OHI Components - Common Features Guide

March 07, 2016

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Table of Contents

1 Data Model

1.1 Activity and Data File Set Model

1.1.1 Activity Model

This entity holds the types of activities that are used for bulk data processing. All activity types are seeded and are not user maintainable.

Activity Type

Field	Description
Code	Unique code of the activity type
Description	Description of the activity type
Active Indicator	Indicates if the activity type is active and can be used
Display in UI Indicator	Indicates if activities of an activity type can be viewed in the UI or not
Top Level Indicator	Indicates if the activity type can be started directly
Type Level	Level on which the activity can be started • GL-Global • TS-Transaction Set • BF- Base Financial Object
Dynamic Record Definition	Association to the dynamic record definition that defines the fields that are available as parameters for activities of an activity type and for the parameter sets created for that activity type

An activity record is a control record for tracking specific execution details of an activity type.

Activity

Field	Description
Activty	The activity that spawned this activity
Activity Type	The activity type to which this activity belongs to
Description	Description of the activity
Internal Remark	Unstructured remarks about the activity
Extra Info	Data with additional processing information
Origin	The origin of the activity • I - Integration Point • M - Manual • S - Spawned
Run Datetime	Date and time on which the activity was started

Status	Status of the activity • BE - Business error Business errors are foreseen situations in the processing of an activity, foreseen meaning that the situation is identified in the processing description and leads to a specified error. Recovery of a business error normally requires changes to the configuration and/or reference data. • CB - Completed with business errors This status is set for a spawning activity if one or more of the spawned activities has a Business Error and none has a Technical Error. • CT - Completed with technical errors This status is set for a spawning activity if one or more of the spawned activities has a Technical Error. • CO - Completed This status is set if the activity itself and all of the spawned activities are completed without any (business or technical) errors. • IN - Initial • IP - In process • TE - Technical Error Technical errors are unforeseen situations in the processing of an activity, caused by for example coding errors or unavailability of system services or resources. Recovery of a technical error caused by erroneous dynamic logic requires debugging the dynamic logic code.Recovery of other technical errors mostly requires actions at system operating level like enabling the system services or resources.
Parameter Set	Reference to the parameter set that was used to start the activity
Dynamic Record	Reference to the dynamic record that is added to an activity based on the activity type of the activity.

Constraints:

- An activity cannot reference both a parameter set and have a parameter values dynamic record.

Activity can cause activity messages.

Activity Message	
Field	**Description**
Code	Code of the message
Element Id	This is for logical grouping of the messages by referencing them to the unit of processing for the activity
Subject Id	Reference to subject involved
Table	Reference to the entity for which the message is meant for, e.g. a relation. Subject Id is always used in combination with a reference to entity, where subject id holds the id within the context of that entity

1.1.2 Data File Set Model

A Data file set is a collection of the data file(s) which are either uploaded to or generated for extraction by OHI Claims application.

Data File Set	
Field	**Description**
Code	The unique identification code of the data file set
Description	Description of the data file set
Indicator Locked	Indicates if the data file set is available for modification or not

Constrains:

- Data file set that is locked cannot be modified.
- Data file set that is locked cannot be picked up for processing by an activity type.

Data File	
Field	**Description**
Code	The unique identification code of the data file within a data file set
Description	Description of the data file
Content Length	Size of the data file
File Data	The contents of the data file
File Path	The path of the uploaded file
MIME Type	The content type of the data file
Data File Set	Reference to the data file set to which the data file belongs

1.2 Translation

This chapter describes multi language support in OHI Claims. Use cases are included to describe installation in multiple languages. This page also describes the translation of seed data and business data.

1.2.1 Concepts

OHI Claims supports multiple languages. It will display texts and messages in the **Display Language** chosen by the end user in the User Preferences Page.

This applies to several categories of items:

1. Boilerplate. This refers to all 'fixed' items on pages like prompts, headings, titles and menu options.
2. Messages. Text of warnings and error messages.
3. Business data. Many entities in OHI Claims have translatable text items. Examples: Code and Description of Tag Types, Display Name of Fields etc.

Every entity that contains translatable attributes, is implemented as a base table and a translations table. The non-translatable columns in the table are stored in the base table, that is given a _B suffix. The translatable columns are stored in the translations table, that is given a _TL suffix. The BASE_TABLE_ID stored in the _TL links a translation to the base. The _TL table is striped by a LANGUAGE column.

Example: MESSAGE entity

Non translatable items of a MESSAGE are stored in OHI_MESSAGES_B. The message attribute is translatable, and thus implemented in the OHI_MESSAGES_TL. (This example does not show all columns of _B and _TL, but only a few to clarify the concept)

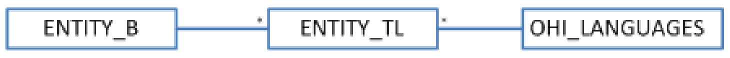
Base Table OHI_MESSAGES_B

ID	CODE	SEVERITY
1	OHI_DUMMY_001	ERROR
2	OHI_DUMMY_002	WARNING

Translation Table OHI_MESSAGES_TL

ID	LANGUAGE	SOURCE_LANG	MESSAGE
1	en__OHI	en__OHI	Maximum exceeded
1	en	en__OHI	Maximum exceeded
1	nl	nl	Maximum overschreden
2	en__OHI	en__OHI	No permission was granted for this
2	en	en__OHI	No permission was granted for this
2	nl	nl	Er is hier geen toestemming voor gegeven

The ID column in the _TL table refers to the base row in the _B table, and both the LANGUAGE and SOURCE_LANG column in the _TL table refer to a table with supported languages.



When displaying data from a _TL table in a screen, the value shown is taken from the row where language equals the Display Language the user has set in the User Preferences page.

SOURCE_LANG determines the source language of a _TL row. In the example above, the rows in OHI_MESSAGES_TL for language 'en' have SOURCE_LANG 'en__OHI', meaning that these rows are copied from the language 'en__OHI' during installation and not translated/changed yet. When the user changes a translatable item, the SOURCE_LANG is set to the current language of the user. This has been done for the _TL rows for language 'nl' in the above example.

So, the Language column in a TL table is used to select the row that matches the Display Language of the current user. The Source_lang column determines the language from which the translatable items originate.

1.2.1.1 Supported languages

The OHI_LANGUAGES table contains the list of supported languages. Per language, several indicators define the possible roles of a language. See table below:

Indicator Column	Purpose	Allowed to change by customer	Number of languages with value 'Y' for this Indicator
ind_default	the language in which translatable items are shown when no user preference is set.	Yes. Should be set to an installed language	1
ind_ohi_specific	'Y' for languages for which seed data is delivered.	No.	1
ind_installed	'Y' for languages in which the application can run.	Only before installation	2 or more

1.2.1.2 Seed Data

Both Boilerplate and Messages are delivered as seed data upon installation. For Boilerplate, the customer cannot enter new rows. Customers can add new messages. Seeded messages are marked as OHI Specific and cannot be changed by the customer.
Seed data is only delivered in supported languages. Currently, this is only English. To facilitate translation of terminology, a dedicated language is created for seed data delivery: OHI English (code=en__OHI). Upon installation, the seed data is loaded in OHI Claims under language

'English' and 'OHI English'. Customers can translate seed data in the 'English' language, but not in the 'OHI English' language.

1.2.2 Use Cases

1.2.2.1 Installation in English

Customers want to run OHI Claims in English only. Before installation, the following values should be set in OHI languages.

Code	Ind Default	Ind OHI Specific	Ind Installed
en	Y	N	Y
en__OHI	N	Y	Y
all other languages	N	N	N

During installation, the seeded value for OHI English will be copied to English.
Take for example the message GEN_UINT_002, with text 'Validation errors found, changes are not saved.' Seed data is delivered for language 'en__OHI'.
After installation, the contents of the OHI_MESSAGES_TL table are as follows:

Code	Source Language	Language	Message
GEN_UINT_002	en__OHI	en__OHI	Validation errors found, changes are not saved.
GEN_UINT_002	en__OHI	en	Validation errors found, changes are not saved.

Code is not really a column of OHI_MESSAGES_TL, but a foreign key to the _B table. It is shown this way to clarify the concept.

Exactly the same concept applies to boilerplate texts.

1.2.2.2 Installation in English and French

Customer wants to run OHI Claims in English (default) or French. Before installation, the following values should be set in OHI languages.

Code	Ind Default	Ind OHI Specific	Ind Installed
en	Y	N	Y
en__OHI	N	Y	Y
fr	N	N	Y
all other languages	N	N	N

During installation, the seeded value for OHI English will be copied to English and French.
After installation, the contents of the OHI_MESSAGES_TL table are as follows:

Code	Source Language	Language	Message
GEN_UINT_002	en__OHI	en__OHI	Validation errors found, changes are not saved.
GEN_UINT_002	en__OHI	en	Validation errors found, changes are not saved.
GEN_UINT_002	en__OHI	fr	Validation errors found, changes are not saved.

Notice the value en__OHI of the Source Language column, meaning that the records for en and fr are copied from the OHI English value.

1.2.2.3 Translation of Seed Data

After installation, the seed data can be translated either because of the use of a different language, or because the customer uses business terminology that deviates from the seeded data. For example the seeded data uses the term 'Provider Group' whereas the customer uses the term 'Network' for the same concept.

Translation can be done as follows:

1. Set the Display Language in the User Preferences to the language for which seed data should be translated.
2. Use the Bulk Translation function to translate multiple boilerplate texts and messages in one go. This option is most appropriate when translating business terminology.
3. Use the Messages and Boilerplate functions to translate records one by one.

See the next three use cases for details.

1.2.2.4 Translation of messages one by one

Example for translation of GEN_UINT_002 to French:

1. Set the Display Language to 'French'
2. Go to the Setup Messages page and query GEN_UINT_002.
3. The Messages page will display the French record with the non-translated Message 'Validation errors found, changes are not saved.'
4. User changes the message text to 'Les erreurs de validation trouvé, les changements ne sont pas enregistrées.' (translated by Google).

The contents of the OHI_MESSAGES_TL table is now as follows:

Code	Source Language	Language	Message
GEN_UINT_002	en__OHI	en__OHI	Validation errors found, changes are not saved.
GEN_UINT_002	en__OHI	en	Validation errors found, changes are not saved.
GEN_UINT_002	fr	fr	Les erreurs de validation trouvé, les changements ne sont pas enregistrées.

Only the French row is changed. Notice the change of the Source Language column.

1.2.2.5 Translation boilerplate entries one by one

Steps to translate menu- and display titles; use 'Network' instead of 'Provider Group':

1. Set the Display Language to 'English'
2. Go to the Setup Boilerplate Texts page and query the boilerplate texts with code like 'rel_providergroup_title%'.
3. The Messages page will display the matching boilerplate texts entries with values in OHI-English.
4. User changes the text values of the boilerplate texts by replacing the term 'Provider Group' by 'Network'.

The contents of the OHI_MESSAGES_TL table is now as follows:

Code	Source Language	Language	Text Value
REL_PROVIDERGROUP_TITLE_SINGULAR	en__OHI	en__OHI	Provider Group
REL_PROVIDERGROUP_TITLE_SINGULAR	en	en	Network

REL_PROVIDERGROUP_TITLE_SINGULAR	en__OHI	fr	Provider Group

1.2.2.6 Bulk translation

Instead of translation of boilerplate text entries and messages one by one, bulk translation can translate multiple items at once. When translation of business terminology is needed, it is most likely that multiple messages and boilerplate text entries have to be changed.

Steps to use 'Network' instead of 'Provider Group' in boilerplate and messages:

1. Set the Display Language to 'English'
2. Go to the Bulk Translation page.
3. Enter '%provider group%' as search value. All matching messages and boilerplate entries are shown.
4. Enter 'provider group' for Term From and 'network' for Term To and press translate.
5. All occurrences of 'provider group' in the matching messages and boilerplate entries are now replaced by 'network'. These changes are not committed yet.
6. Review the changes and make corrections if necessary.
7. Commit the changes by pressing 'Save'.

1.2.2.7 Reinstallation

During reinstallation, only rows with Source Language 'en__OHI' will be overwritten. Translated values with other source language values will be retained.

1.2.2.8 Creation of Business Data

When new business data is created, text values are duplicated to the other installed languages using the same approach as during installation.

Example:

- en__OHI, en and fr are installed languages.
- User has Display Language set to French.

User creates a new Tag Type 'Personne très Importante'. The contents of the REL_TAG_TYPES_TL table are now as follows:

Code	Source Language	Language	Description
PTI	fr	fr	Personne très Importante
PTI	fr	en	Personne très Importante.
PTI	fr	en__OHI	Personne très Importante

1.2.2.9 Translation of Business Data

After creation of the tagtype using the steps in the previous paragraph, the user switches to English and changes the code and description to VIP, Very Important Person.

The contents of the REL_TAG_TYPES_TL table are now as follows:

Code	Source Language	Language	Description
PTI	fr	fr	Personne très Importante
VIP	en	en	Very Important Person.
PTI	fr	en__OHI	Personne très Importante

From the UI, it is not possible to see which items are translatable and which are not. This query lists all items in _TL tables:

```
set pagesize 100
set linesize 100
set escape off

select table_name, column_name from all_tab_columns where table_name like '%\_TL' escape '\'
and column_name not in ('BASE_TABLE_ID','ID','LANGUAGE','SOURCE_LANG')
order by table_name, column_name
/
```

2 Integration Concepts

OHI Components applications are designed to operate as a component in a component-based or service oriented architecture. This guide provides an overview of the integration capabilities of OHI Components applications. Additional information with respect to OHI Components applications integration points is available in other guides, for example:

- See the Installation Guide for service endpoint URLs and configuration options for these
- See the Security Guide for guidance on properly securing integration endpoints

2.1 Integrating with OHI Components Applications

This chapter introduces techniques and patterns used for integrating with OHI Components applications. The following principles played an important role for designing the integration services:

- Use of well-known, proven standards that are widely supported by tools and middleware of many vendors in the IT industry
- Avoid interoperability issues, allowing customers to use OHI Components applications with the IT services they already own and operate

Common integration techniques used in OHI Components applications are:

- File-based integration is used for importing or exporting large sets of data. Examples include imports of ICD-9 or ICD-10 code systems and export of financial messages that are generated in OHI Components Claims or OHI Components Value-Based Payments.
- SOAP web services are used for exchanging business data with other applications in a health insurance payer's IT landscape. For SOAP web services in OHI Components applications the 'service contract' is defined by Oracle in the form of a WSDL that defines the service operations and XSDs that define the message payloads. Oracle also defines the WSDL and XSDs for web services that are called from OHI Components applications.
- RESTful style web services are also available for interacting and integrating with OHI Components applications. Formal specifications for RESTful services is available in the form of RAML documents that are bundled with each application.

Going forward, any new services will be developed as RESTful services, no new SOAP services will be added.

Oracle develops SOAP web services with JAX-WS, the reference implementation for developing SOAP web services in Java whereas RESTful services are constructed using JAX-RS, the reference implementation for developing RESTful web services in Java.

All web services in OHI Components applications perform stateless operations.

2.2 Auditing and Exception Handling

In a component architecture it is paramount that interaction between systems can be tracked and audited. The following is a list of system behaviors and features that support auditing and exception handling in OHI Components:

- Messages that indicate any kind of failure are always logged. Messages that confirm a successful result within the context of synchronous interaction are not logged. The overhead for logging these would impact the performance of these relatively lightweight operations. Messages that indicate failure are logged together with data that can help to determine the cause.

- If an OHI Components application cannot deliver a message, it will not retry that operation instantly. This behavior is based on the underlying assumption that a network failure that prevents successful interaction is not going to be resolved instantly. Instead, a task is raised for delivering the message at a later moment in time, to be triggered by a system operator as soon as the network is restored.
- For SOAP services, OHI Components applications can validate if message payloads adhere to the XSD specification. This feature is configurable on a per web service basis and is disabled by default for performance reasons.
- The logging subsystem can be configured to gather message payloads in log files. Specific measures can be taken for logging message payloads that may contain protected health information. Additional details for checking interfaced messages and results of processing these are documented elsewhere in this guide.

3 Soap Integration Points

3.1 Attribute Handling

Each integration point request message contains data values of a top-level entity (e.g. relation or provider) that have been created or updated in a source system. Within each message are several categories of data such as simple entity-level attributes, lists of non-time valid details, and lists of time-valid details. This section describes how each category of data is handled by OHI applications and also covers guidelines for handling differences in data categories between source systems and OHI applications.

The way that an OHI application handles web service requests is based on the principle that the copy of information in the OHI application is to be kept up-to-date with the (master) information in the system of record. It is not required that the OHI application is informed of every update system of record; only the values at the time of creating the web service requests are important. For example, if an interface periodically creates requests for all outstanding additions and updates, only the values of source system data at the time that the interface is run need to be sent. Whether a record has been updated several times or once since the last interface run is irrelevant.

If an existing record is sent again and it contains the exact same data that is already stored in the system, the existing data will not be updated. This means that the audit columns in the database will also remain unchanged. Tracking these messages can be done through the interface messages log.

3.1.1 Single Value Attributes

These are fields that can have only one single value and the value does not have a start and end date. When the application creates a new record, single value attributes are handled as follows: if the attribute is not included in the request, then the corresponding attribute in the new record will be set to null; if the attribute is included in the request, then the corresponding attribute in the new record will be set to the specified value.

When the application updates a record, a single value attribute is handled as follows: if the attribute is not included in the request, then the existing value in the application remains untouched; if the attribute is included in the request, then the attribute value is updated with the specified value.

For example, consider a new relation being added in the system of record. Because the OHI application keeps a local copy of relation records, the system of record sends the following request to the OHI application:

```
<relation
  code="1333"
  name=Jones
>
  <personDetails
    firstName="John"
  >
>
```

Since this is the first time that the relation with code 1333 is being sent, the OHI application creates a new relation record with only code, name, and firstName having values. All other attributes in the new record in the OHI application will be null. The relation is updated in the source system; the first name is changed from "John" to "Jonathan". The system of record sends the following request to the OHI application:

```
<relation
  code="1333"
>
  <subTypeDetails>
```

```
        <person
          firstName="Jonathan"
        >
      </subTypeDetails>
    </relation>
```

Since there is already a relation with this code, the OHI application will update the relation record with code = 1333 setting <person firstName> to Jonathan. <Relation name> will not be changed or set to null.

External Interface Design Notes

If a given type of data is a single value attribute in an Integration Point message definition and a time valid detail list in the source system, the external interface is expected to only send the latest / current value.

If a given type of data is a single value attribute in an Integration Point message definition and a non time valid detail list or otherwise not a one-to-one match in the source system, the external interface will require (dynamic) logic needed to determine what value should be provided.

3.1.1.1 Amount and Currency

Amount and currency are two attributes that belong together, so they are always represented together in a separate element. For example:

```
    <authorizedAmount
      amount="100"
      currencyCode="USD"
    />
```

If the element is not included in the request, then the existing values for amount and currency in the application remain untouched; if the element is included in the request, then the values are updated with the specified values. In order to send in an update that clears the values, the update request should include the element without any attributes and values (empty element).

3.1.2 Non Time Valid Details

This category covers a list of details (of single values or of detail records) that are details of a parent entity but that are not time valid (i.e. each detail record does not have start and end dates), such as relation titles or tags.

When the OHI application creates new records (for an entity with a non time valid detail list):

- If the detail list element is not included, no details are added
- if a detail list element is included, one detail record for each included detail item is created

When the OHI application is updating an existing record (for an entity with a non time valid detail list):

- If the list element is not included, no changes to current details are made
- If a list element is included, included detail items completely replace current detail records (in effect, all current detail records that have not been resent (i.e. for which there is no detail item with exactly matching values) are removed and one detail record for each included detail item (that is not a 'resend') is created)
- If an empty list element is included, all current detail records are removed

3.1.3 Time Valid Details

This category refers to a list of details within a parent entity that is time valid (i.e. each item of the list has a start and an end date). The items of the list have one or more single value attributes (in addition to the start and end dates) and may themselves contain a list of details. For example, a product (parent record) may have multiple provider groups (time valid details). The link between each product and provider group has a start and an end date.

Lists of details may have a 'detail functional key'. A detail functional key refers to a key of list items / detail records that is unique within a list of details at a specific point in time. In effect a history of values is maintained per detail functional key. An example of a list of details with a detail functional key is relation addresses. In this case, address type is the detail functional key and there may only be one address of each type current at any point in time; however, there may be more than one address current at the same point in time if they are of different types. An example of a list of details without a detail functional key is marital status of relations. In this case, there may only be one marital status current at any given point of time. The following sections describe how the various conditions related to time valid details are handled.

3.1.3.1 Parent Record Creation

When the OHI application creates a new record (for an entity with a time valid list of details),

- if the list element is not included, no details are added,
- if a list element is included, one detail record is created for each item of the list.

3.1.3.2 Parent Record Update

When the OHI application updates an existing record (for an entity with a time valid list of details), one of the following situations applies:

- no list element is included
- the list element is included without list items and without a resendFromDate
- the list element is included without list items, but with a resendFromDate
- the list element is included with list items

In the first two situations, no changes to current details are made. In the third and fourth situation, i.e., a list with items or/and a resendFromDate, the OHI application updates the list on the existing record. How that detail list is updated depends on several conditions related to the message contents and what is already stored in the OHI application. This decision table indicates for the various combinations of conditions which updates are made. Each of these scenarios is described separately following the table.

	Conditions			**Expected Results**			
	Start Date Match	**Starts During Period**	**Starts Before Last Period**	**End Overlap Period**	**Update Matched Period**	**Create Period**	**Delete Later Period(s)**
1.	Y	N	Y		X		X
2.	Y	N	N		X		
3.	N	Y	Y	X		X	X
4.	N	Y	N	X		X	
5.	N	N	Y			X	X
6.	N	N	N			X	

Essentially, the detail list in a parent record update replaces the detail list of the existing record, going forward from a specific date. That date can either be explicit in the update request (the attribute 'resendFromDate') or, if not made explicit, it defaults to the earliest start date of the detail list in the update request.

For example, suppose a relation has had a number of addresses starting on from the 1st of Jan 2008. The relation is updated with a list of addresses in which the first address starts on the 1st of Jan 2009. The update does not specify an explicit resendFromDate, so it defaults to the 1st of Jan 2009. The existing address history between Jan 1st 2008 and Dec 31st 2008 remains intact.

The following two diagrams illustrate the scenarios using marital status. Scenario 1 through 6 presume no explicit resendFromDate is specified. Scenario 7 clarifies what happens in the event that the resendFromDate is explicit.

'Create' indicates what has already been created and is present in the OHI application when the 'Update' is received. The result indicates what the situation will be in the OHI application once the update is processed.

3.1.3.2.1 Matching Start Date (Scenarios 1 and 2)

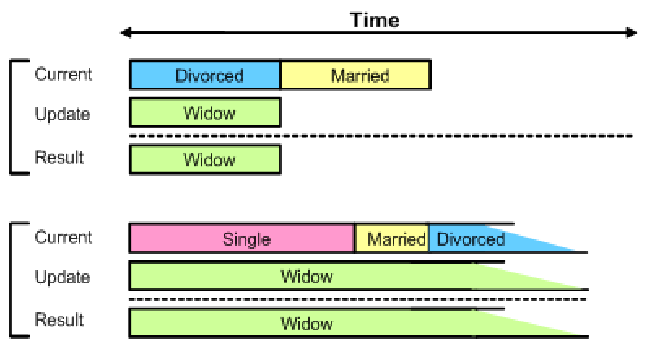


Figure 3-1: Examples of scenario 1

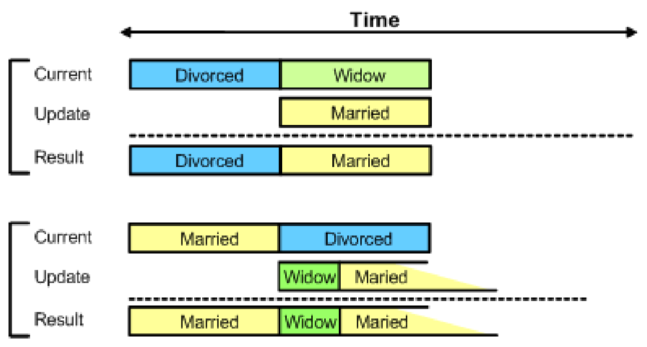


Figure 3-2: Examples of scenario 2

Detail Functional Key

If a new item is received in a message and there is a record present with the same detail functional key and a matching start date, this record will be updated with new values from the message (if any of the values differ). Any record with the same detail functional key that starts after the start date of the new item that has not been resent will be deleted. Note that end date is treated in the same way as other fields.

No Detail Functional Key

If a new item is received in a message and there is a record present with the same start date, this record will be updated with new values from the message (if any of the values differ). All records that start after the start date of the new item that have not been resent will be deleted. Note that end date is treated in the same way as other fields.

Examples

An address (that is already stored in the OHI application) is corrected in the source system by changing it from

```
Site (S), 122 Jefferson Ave, New York (2005-1-1 - 2008-6-30)
```

to

```
Site (S), 128 Jefferson Ave, New York (2005-1-1 - 2008-6-30)
```

The external interface needs to send the latest details:

```
<address
  type="S"
  houseNumber="128"
  startDate="2005-1-1"
  endDate="2008-6-30"
/>
```

The OHI application will then update its previously stored copy of the address.

An address (that is already stored in the OHI application) is ended in the source system by changing it from

```
Site (S), 1422-3rd Ave, New York (2008-1-1 - )
```

to

```
Site (S), 1422-3rd Ave, New York (2008-1-1 - 2009-8-31)
```

The external interface needs to send these details:

```
<address
  type="S"
  startDate="2008-1-1"
  endDate="2009-8-31"
/>
```

The OHI application will then update its previously stored copy of the address with the new end date.

Note: In this example, <address type> is the detail functional key.

3.1.3.2.2 Values Updated (starting a new period during an existing period) (Scenario 3 and 4)

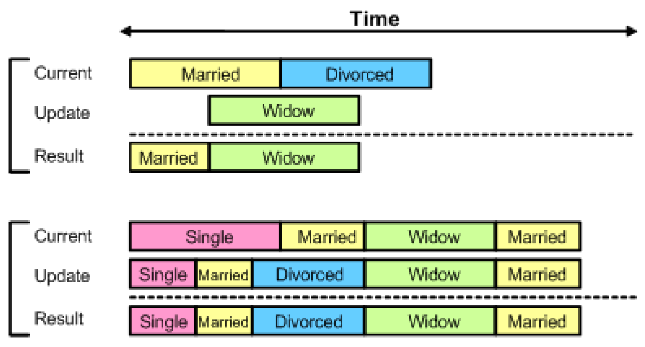


Figure 3-3: Examples of scenario 3

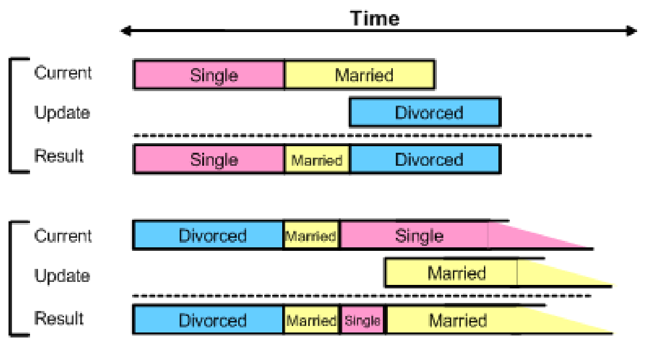


Figure 3-4: Examples of scenario 4

Detail Functional Key

If an item is received with a start date that is after the start date of the last record with the same detail functional key and the last record has no end date , the last record will have its end date updated to the day before the start date of the new detail record and a new record will be created with the new details from the message.

Likewise, if an item is received with a start date that is between the start date and end dates of a record with the same detail functional key:

- this record will have its end date updated to the day before the start date of the new detail record
- any record with the same detail functional key that starts after the start date of the new item that has not been resent will be deleted
- the new address will be stored

No Detail Functional Key

If an item is received with a start date that is after the start date of the last record and the last record has no end date , the last record will have its end date updated to the day before the start date of the new detail record and a new record will be created with the new details from the message.

Likewise, if an item is received with a start date that is between the start date and end dates of a record:

- this record will have its end date updated to the day before the start date of the new detail record
- any record that starts after the start date of the new item that has not been resent will be deleted
- the new address will be stored

Example

A new address is recorded in the source system when there is already an address (for the same detail functional key, in this case address type) stored in the OHI application.

Address before adding new address:

```
Site (S), 1223-9th Ave, New York (2008-7-1 - )
```

New address with effective date 2009-8-1

```
Site (S), 544-E 35th Street, New York (2009-8-1 - )
```

The external interface only needs to send the new address details:

```
<address
  type="S"
  street="E 35th Street"
  houseNumber="544"
  startDate="2009-8-1"
/>
```

The OHI application will then update its previously stored copy of the address with an end date and store the new address.

```
Site (S), 1223-9th Ave, New York (2008-7-1 - 2009-7-31)
Site (S), 544-E 35th Street, New York (2009-8-1 - )
```

Alternately, the external interface could send both addresses with both changed and unchanged details:

```
<address
  type="S"
  street="9th Ave"
  houseNumber="1223"
  city="New York"
  startDate="2008-7-1"
  endDate="2009-7-31"
/>
<address
  type="S"
  street="E 35th Street"
  houseNumber="544"
  city="New York"
  startDate="2009-8-1"
/>
```

This would have the same result in the OHI application.

Example 1

A new (past) address is recorded in the source system when there are already addresses (for the same detail functional key, in this case address type) stored in the OHI application.

Addresses (in both the source system and the OHI application) before adding new address:

```
Site (S), 122 Washington St, New York (2007-1-1 - 2008-4-30)
Site (S), 185-7th Ave, New York (2008-5-1 - )
```

Addresses after adding new address:

```
Site (S), 122 Washington St, New York (2007-1-1 - 2008-1-31)
Site (S), 342 E 46th Street, New York (2008-2-1 - 2008-4-30)
Site (S), 185-7th Ave, New York (2008-5-1 - )
```

The external interface needs to send the new addresses and all addresses that start after it:

```
<address
  type="S"
  street="E 46th Street"
  houseNumber="342"
  city="New York"
  startDate="2008-2-1"
  endDate="2008-4-31"
/>
<address
  type="S"
  street="7th Ave"
  houseNumber="185"
  city="New York"
  startDate="2008-5-1"
/>
```

The OHI application will then update the end date of the address that end after the start date of the new address and replace later addresses with the newly provided addresses resulting in:

```
Site (S), 122 Washington St, New York (2007-1-1 - 2008-1-31)
Site (S), 342 E 46th Street, New York (2008-2-1 - 2008-4-31)
Site (S), 185-7th Ave, New York (2008-5-1 - )
```

Alternately, the external interface could also send the address that needed to be updated with the new end date:

```
<address
  type="S"
  street="Washington St"
  houseNumber="122"
  city="New York"
  startDate="2008-2-1"
```

```
    endDate="2008-4-31"
  />
  <address
    type="S"
    street="E 46th Street"
    houseNumber="342"
    city="New York"
    startDate="2008-2-1"
    endDate="2008-4-31"
  />
  <address
    type="S"
    street="7th Ave"
    houseNumber="185"
    city="New York"
    startDate="2008-5-1"
  />
```

with the same result in the OHI application.

Example 2

A new maritalStatus is added 'within' the periods of a previous status:

Original maritalStatus list:

```
  Single (2007-1-1 - 2008-4-31)
  Married (2008-5-1 - )
```

List after adding new status

```
  Single  (2007-1-1 - 2007-10-31)
  Married (2007-11-1 - 2008-1-31)
  Single  (2008-2-1 - 2008-4-30)
  Married (2008-5-1 - )
```

The external interface will need to send all periods starting with the period that has been added:

```
  <maritalStatus
    status="Married"
    startDate="01-11-2007"
    endDate="31-01-2008"
  />
  <maritalStatus
    status="Single"
    startDate="01-02-2008"
    endDate="31-04-2008"
  />
  <maritalStatus
    status="Married"
    startDate="2008-5-1"
   >
```

The OHI application will then end the first status and replace the later statuses with the new ones from the message.

3.1.3.2.3 Late Addition of a Detail Item (Scenario 5)

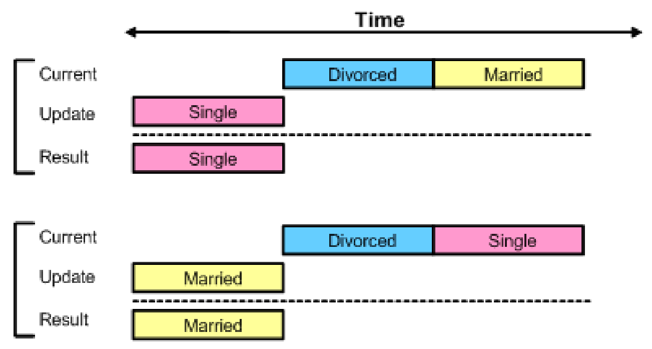


Figure 3-5: Examples of scenario 5

Detail Functional Key

If an item is received with a start date before the start date of the last record with the same detail functional key and it does not start between the start and end dates of another record with the same detail functional key:

- any record with the same detail functional key that starts after the start date of the new item that has not been resent will be deleted
- the new address will be stored

No Detail Functional Key

If an item is received with a start date before the start date of the last record and it does not start between the start and end dates of another record:

- any record that starts after the start date of the new item that has not been resent will be deleted
- the new address will be stored

External Interface Design Notes

In effect, what this approach means is that when a time valid detail record is sent, all subsequent detail records (for the same detail functional key if applicable) need to be sent as well.

If a given type of data is a time valid detail list in an Integration Point message definition and a single value attribute (set of attributes) in the source system, the external interface is expected to send the current values in the time valid detail list with a fixed start date (earlier than would ever be referred to) and a null end date.

If a given type of data is a time valid detail list in an Integration Point message definition and a non time valid detail list or otherwise not a one-to-one match in the source system, the external interface is expected to apply (dynamic) logic needed to determine what value should be provided.

3.1.3.2.4 No Overlapping Dates and No Later Periods (Scenario 6)

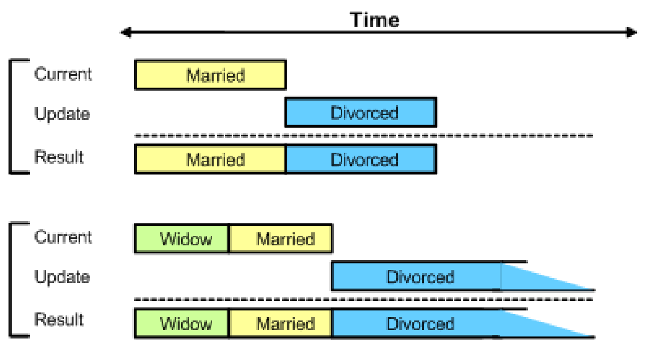


Figure 3-6: Examples of scenario 6

Detail Functional Key

If a new item is received in a message and for the item:

- there are no records present with the same detail functional key,
- there are records present with the same detail functional key but the new item starts later then any of these records ends,

a detail record is created for the item.

No Detail Functional Key

If a new item is received in a message and:

- there are no current items, or
- all current items end before the new item starts,

a detail record is created for the item.

3.1.3.2.5 Explicit Resend Date (Scenario 7)

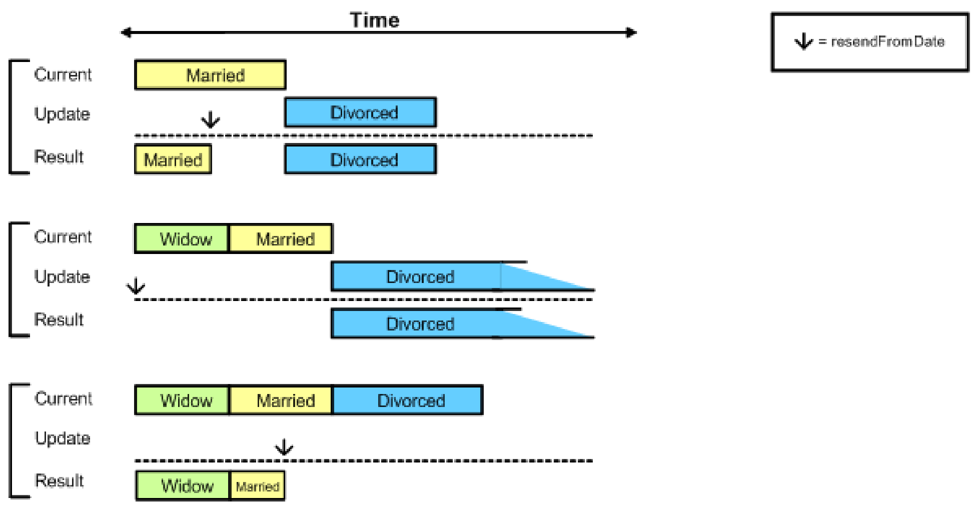


Figure 3-7: Examples of scenario 7

In scenario 1 through 6, the detail item list is updated going forward from the start date of the first list item in the update request. As a consequence, the updates in these scenarios always entail the creation of a new item on that date going forward. In order to update a detail item list in such a way that a list item is removed without replacement, the list header element must specify a resendFromDate. This has the following effect:

- All existing list items that start on or later than the resendFromDate are deleted
- All existing list items that start before and end on or later than the resendFromDate are ended one day prior to the resendFromDate
- All existing list items that end before the resendFromDate remain untouched
- The existing list is updated with the list items in the update request.

In the event that one of the list items in the update request has a start date prior to the specified resendFromDate, the resendFromDate is ignored and the update will be accordance with scenario 1 through 6. In the event that the update request only specifies the list header element with a resendFromDate, the existing list is simply updated in accordance with the bullets above, but no new items are added to that list as a result of the update.

Note that lists with and without detail functional key are handled in the same way, i.e., a specified resendFromDate removes all items from that date going forward.

External Interface Design Notes

When using the resendFromDate all items of a given list for all detail functional keys with a start date on or after the resendFromDate need to be resent. This is different than in the other scenarios where only items of a single detail key need to be resent.

Example 1

Original maritalStatus list:

```
Single (2007-1-1 through 2008-4-31)
Married (2008-5-1, no end date )
```

Update request message:

```
<maritalStatusList
  resendFromDate="2007-07-01>
  <maritalStatus
    status="Married"
    startDate="2007-11-01"
  />
</maritalStatusList>
```

This update will remove all marital status information going forward from the resendFromDate, i.e., 2007-07-01. This will cause the existing item "Single" to be end dated on 2007-06-30, and will entirely remove the existing item for "Married". A new "Married" item is added to the list. List after the update:

```
Single  (2007-1-1 - 2007-6-30)
Married (2007-11-1, no end date)
```

3.2 Dynamic Free Fields, Codes and Records

This section describes the default behavior of integration points that accept dynamic field, code and record values. The configuration of the relevant dynamic field usages drive how the integration point processes a request with new dynamic field values. The driving aspects are the usage type (field, code or record), whether the field can have multiple values, whether the field vale is time valid and whether the field values have a key identifier.

Dynamic field values can be created and updated through the <dynamicField> element. The name attribute in this element identifies the field that is created or updated. If an update request does not include a <dynamicField> element for a dynamic field usage with existing values, then those existing values are not touched. In other words, the default behavior for an integration point is that an update request without a <dynamicFields> element leaves all existing dynamic field values intact.

This section describes update behavior in particular, since the initial insertion of new values is the same across all types and usage configuration.

3.2.1 Free Field Values

Like any dynamic field, free fields can be configured to have multiple values and these values can be configured to be time valid. Free field values are either dates, numbers or character strings; they never have a functional identifier. This makes the update behavior straightforward; sending in (a) value(s) for a single-value non-time-valid field, a multi-value non-time-valid field or a multi-value time-valid fields always results in a full replacement of all the existing values.

Single-value time-valid fields have slightly different behavior; sending in a new value for such a field will delete any values with a later start date, will cut off any overlapping value with an earlier start date, but will not touch any existing value that is end-dated before the start date of the new value.

The following pseudo XML reflects the structure of a single-value time-valid free field

```
<dynamicFields>
  <dynamicField
    name="occupation"
  >
    <value
      startDate="2010-05-01"
      endDate="2012-09-30"
    >Baker</value>
```

Suppose that, before the request is processed, the following values exist for the "occupation" field:

Occupation		
Value	**Start**	**End**
Butcher	2005-01-01	2009-12-31
Courier	2010-01-01	2011-05-31
Shopkeeper	2011-06-01	n/a

The result of the update request would be:

Occupation		
Value	**Start**	**End**
Butcher	2005-01-01	2009-12-31
Courier	2010-01-01	2010-04-30
Baker	2010-05-01	2012-09-30

The "Shopkeeper" value is removed because it started after the "Baker" value in the update request. The "Courier" value is cut off one day prior to the "Baker" value to make sure that there is no overlap in time. The "Butcher" value remains untouched.

For time-valid dynamic fields, the startDate attribute in the <value> element is required; the XML schema definition does not enforce this because the same element is also used for non-time-valid fields.

The value of a dynamic free field can be a date, number or character string. Since the schema has no knowledge of the configuration of the field, the content of the <value> element is validated by the integration point logic. Date values must be sent in according to the xsd:date format, i.e., YYYY-MM-DD. The decimal character for number values must be accordance with the installation settings. If an integration point fails to parse a dynamic field value as a date, the error message GEN-PROC-001 (Value provided is not of type Date) is generated. If an integration point fails to parse a dynamic field value as a number, the error message GEN-PROC-002 (Value provided is not of type Number) is generated.

The schema definition does not except empty <value> elements. In order to sent in an update that clears all values for a particular dynamic field, the update request should include a <dynamicField> element without a <value> element, as reflected in the following pseudo code:

```
<dynamicFields>
  <dynamicField
    name="occupation"/>
```

The following is an example for the update of a single-value non-time-valid dynamic field with usage name "comment":

```
<dynamicFields>
  <dynamicField
    name="comment"
  >
    <value>Need to verify spelling of surname</value>
  </dynamicField>
```

3.2.2 Code Values

Update requests for code dynamic fields are handled similar to those for free fields, with the exception of multi-valued code fields. The reason is that a code always has key field; in other words, it is possible to pinpoint the exact code that you want to update, allowing a more subtle update feature.

Sending in a new value for a single-value non-time-valid code field results in the replacement of the old value with the new. Sending in a new value for a single-value time-valid code field will delete any values with a later start date, will cut off any overlapping value with an earlier start date, but will not touch any existing value that is end-dated before the start date of the new value.

Sending in (a) value(s) for a multi-valued non-time-valid code field can only ever result in the addition of a new code value to the field. This happens when the update includes a code value that is not among the existing values. Existing values (regardless of whether they are included in the update) are left untouched.

Sending in (a) value(s) for multi-valued time-valid will add new code values if the value is not among the existing code values for that field. In case the same code is among the existing values, then the values (for the same code) with a later start date are deleted, the values for the same code with an overlap and an earlier start date are cut off. Code values that are not in the update request are left untouched.

The following pseudo XML reflects the structure of a single-value time-valid code field:

```
<dynamicField
  name="primaryDiagnosis"
>
  <value
    startDate="2010-05-23"
    flexCodeDefinitionCode="ICD09_V_DIAGNOSES"
  >V51
  </value>
</dynamicField>
```

The flexCodeDefinitionCode attribute is used to disambiguate between code definitions. This is a fairly exceptional situation, since it is unusual for different code systems (used in the same context) to have overlapping codes. For example, suppose the field refers to a diagnosis code and the value is V51. Since both the ICD09 and the ICD10 code systems know a diagnosis code V51, the <value> element needs to specify which of the two systems is the intended one.

The following XML shows an example update for a single-value time-valid code field with consecutive values in time:

```
<dynamicField
  name="occupation"
  >
  <value
    startDate="2005-01-01"
```

```
      endDate="2007-12-31"
    >NURSE</value>
    <value
      startDate="2008-01-01"
    >GP</value>
  </dynamicField>
```

3.2.3 Dynamic Records Values

Many of the entities that can be sent in through an integration point can be extended with dynamic fields and / or dynamic records. The values for these records and fields can be set through the integration points as well. Dynamic record values are set by using the <dynamicRecordTables> element. The following pseudo XML snippets illustrate the structure of this element.

```
  <dynamicRecordTables>
    <dynamicRecordTable
      name
    >
      <row
        startDate
        endDate
        indDeleted
   >
        <column
          name
          flexCodeDefinitionCode
        />
          value
        </column>
```

The name in the <dynamicRecordTable> element maps on to the dynamic record usage name. Each <row> represents a dynamic record value. The startDate and endDate are used in case the dynamic record usage is set up to be time valid.

The indicator indDeleted of a row can be used to delete specific dynamic records. This is only possible for dynamic record definitions where a particular column is set up to be 'key'. The dynamic record to delete is then determined by matching the key column value in the request to the key column values present in the database; any dynamic record with the same key value (and the same parent data element) will be deleted. If the dynamic record definition does not specify a key column, indDeleted cannot be used (if sent in anyway it will be disregarded).

Each <column> element sets a value in a single dynamic record. The name attribute maps on to the dynamic record field usage code. The field usage can refer to either a field (such as a date or a number) or a code definition (such as a diagnosis or a procedure). If it refers to a code definition, the <column> value maps on to the key field of the code definition. The schema definition does not allow <column> elements without a value; if a column within a record should remain blank, no <column> element for that column should be included.

All integration points that accept dynamic record tables include a de-duplication feature. For each <row> element, the application checks to see if an identical <row> exists in the message; if it does, then the duplicate is removed. If the dynamic record definition specifies a key field, then de-duplication is based on the key field value alone. For time valid dynamic records, de-duplication is based on the combination of the key field and the start date. If no key field is specified, a <row> is considered a duplicate when *all* field values in the record are the same as in another record.

The following table describes the update behavior of dynamic records through integration points. The assumption is that one or more dynamic record values are already present in the database when the request is processed. The behavior differs based on whether the dynamic record usage allows for multiple values, time valid records and/or has a functional key field.

Specifying a <dynamicRecordTable> element without any <row> element will clear all dynamic records for that particular usage. In case the request does include (a) <row> element(s) then the following logic applies:

Multi	Time	Key	Update behavior

N	N	N	Replace the existing record with the new
N	N	Y	Replace the existing record with the new
Y	N	N	Replace all existing records with the new record(s)
Y	N	Y	Existing records with a key value that matches the key of a record in the request are replaced. Existing records with a key value that is not in the request remain untouched.
N	Y	N	Existing records with the same or later start date are removed The existing overlapping record with an earlier start date is cut off All other existing records remain untouched
N	Y	Y	Existing records with the same or later start date are removed The existing overlapping record with an earlier start date is cut off All other existing records remain untouched
Y	Y	N	The earliest start date of the records in the request message is considered to be the as-of-date from which point onwards only the records included in the request message are valid. Existing records that overlap in time are ended or deleted automatically: • Existing records that start before and end on or later than the as-of-date, are ended one day prior to the as-of-date. • Existing records that start on or later than the as-of-date are deleted. All other existing records remain untouched

Y	Y	Y	Existing records with the same key value and with the same or later start date are removed The existing overlapping record with the same key value and with an earlier start date is cut off All other existing records remain untouched

3.3 File Based Integration

In the following cases, data can be loaded into OHI Claims using files (bulk or batch processing):

- Diagnoses
- Procedures
- Product Definitions
- Consumption Batch Import
- Fee Schedules
- Provider Pricing Clauses

3.3.1 File Import Batch Processing Request

Batch processing of the contents of a file starts with placing the file that is to be processed in a 'input file directory' and by subsequently informing OHI Claims that a file is ready for processing. The latter is done using the File Import Web Service for which the WSDL is typically available at the following address:

```
http://machine.domain:port/ohi-web-services/FileImportService/
fileImport.wsdl
```

The *fileImportRequest* method of the File Import Web Service takes a message that adheres to the FileImport.xsd specification. A sample message looks like this:

```
<fileImportRequest xmlns="http://healthinsurance.oracle.com/ws/
fileimport/v3">
    <filePath>/diagnoses/2009/diagnosesCodesOct.xml</filePath>
    <importProcess>diagnosesImport</importProcess>
    <responseFilePath>/diagnosesResponses/diagnosesCodesOct.xml</
responseFilePath>
    <successFilePath>/successDiagnosesResponses/
diagnosesCodesOct.xml</successFilePath>
    <failureFilePath>/failureDiagnosesResponses/
diagnosesCodesOct.xml</failureFilePath>
</fileImportRequest>
```

Explanation of the elements in the message:

- The required element *filePath* specifies the relative path to the file that needs to be processed. It will be prepended with the value of system property *ohi.ws.fileimport.filesrootdirectory*.
- The required element *importProcess* identifies the process to be used for importing the data. If an invalid process name is used then the system reports error GEN-FILE-013 in the file import response message. The following import processes are supported:
    - diagnoses
    - procedures
    - productDefinition
    - writeConsumptionBatchRequest

- feeScheduleBatchRequest

- The required *responseFilePath* element specifies the relative path to a response file that is generated by OHI Claims as a result of processing the data in the file that is to be processed. It will be prepended with the value of system property *ohi.ws.fileimport.filesrootdirectory*.
- The optional *successFilePath* element specifies the relative path to which the input file that was processed will be moved by OHI Claims after successfully processing the data in the input file. The relative path will be prepended with the value of system property *ohi.ws.fileimport.filesrootdirectory*. If the *successFilePath* element is not specified, OHI Next does not move the input file after its contents were successfully processed.
- The optional *failureFilePath* element specifies the relative path to which the input file that was processed is moved by OHI Claims after ***un***successfully processing the data in the input file. The relative path will be prepended with the value of system property *ohi.ws.fileimport.filesrootdirectory*. If the *failureFilePath* element is not specified, OHI Claims does not move the input file after its contents were ***un***successfully processed.

In case given directories are not available, the system will attempt to create these. A file import job will not be started when the *responseFilePath* did not exist and could not be created; the response message will indicate a failure. When either a *successFilePath* or *failureFilePath* is specified and these did not exist or could not be created, the file import job will not be started either. Again, the response message will indicate the failure.

As soon as the message is received by OHI Claims, it will be queued for processing. File import messages are processed in the order of delivery.

3.3.1.1 Allowed characters for File Paths

For security reasons, not all characters that can normally be used in file names are allowed. Characters that are allowed in a file name are:

- platform-specific file separator, i.e. "/" on Unix and "\" for Windows platforms
- alphanumeric characters: a - z, A - Z, 0 - 9
- spaces
- dots
- hyphens and underscores

If any other character is detected, a GEN-FILE-009 error is returned and the File Import process fails.

If the input file cannot be detected or is not readable, a GEN-FILE-004 error is returned and the File Import process fails.

3.3.2 Response File and Process Completion Notification

With batch processing, there is one response per batch that is submitted even though a batch request message may in effect consists of multiple (main-level) items that are relatively independent of each other. Each response message indicates how many (main-level) items were successfully processed and contains task messages for the individual (main-level) items that have them. Batch response messages are stored in a file and referenced from a response notification message.

As a result of File Import batch processing, OHI Claims creates an XML response file containing the results of the import process. When the import process completes and the response file is generated, OHI Claims delivers a fileImportResponse message that adheres to the FileImportResponse.xsd specification. A sample message looks like this:

```
<fileImportResponse xmlns="http://healthinsurance.oracle.com/ws/
fileimport/v3">
     <result>S</result>
</fileImportResponse>
```

The *result* element specifies the result of processing the input file. Its value can be either S (for success) or F (for failure).

The system uses WS-Addressing properties to determine the Web Service endpoint to which the response message should be delivered and how it relates or correlates to the initial request message. OHI Claims supports WS-Addressing version 1.0 (May 2006). In accordance with that standard, the response message is delivered to the Web Service endpoint that is identified by the ReplyTo address property in the initial SOAP request message. The RelatesTo property is populated with the value of the MessageID that was specified in the initial SOAP request message.

The Web Service for delivering the process completion notification is not delivered as part of OHI Claims.

An import fails when either:

- An exception occurred during processing of the contents of a file
- The input file could not be moved or an exception occurred when moving the file

3.3.3 Interface Task Log

The results of the batch process are maintained in a Task Message Log. This log can be accessed through a look-up page. Access to data for a certain File Import batch job are accessible by process name or through the MessageID. The response file is created from the information in the Task Message Log.

OHI Claims does not clean the Task Message Log automatically.

3.4 Service Based Integration

Next to File Based Integration (page 31), OHI Components applications support service based integration using the following messaging patterns:

- OHI Components Web Services
  - Synchronous message processing (request-response): standard two-way message exchange where a response message immediately follows the inbound request (using the same socket for communication).
  - Asynchronous message processing: an inbound request is not immediately processed; the response message is delivered after processing to the endpoint address specified by the client at the time of sending the request.
- OHI Components as Client (calling external services)
  - Synchronous Message Processing (request-response): similar to inbound requests, OHI Components applications expect the response to immediately follow its outbound request (using the same socket for communication).
  - Asynchronous message processing: an outbound request will not be processed immediately; the response message should be delivered after processing to the endpoint address that is specified by OHI Components applications at the time of sending the request.

3.4.1 OHI Components Web Services

3.4.1.1 Synchronous Message Processing

If a request is received and the *ohi.ws.<integration_point>.request.validate* property for the specific Integration Point is set to true, the message is validated to check if it adheres to the XSD specification. If the request is not valid, a SOAP fault will be returned and the message is not processed. The validity check is not performed if the *ohi.ws.<integration_point>.request.validate* property is set to false.

A synchronous request is processed immediately. The result of processing the request message is returned to the client immediately after processing. The format of the response message adheres to the message specification that is dictated by the WSDL specification of the web service.

As the response message contains details of the results of processing the request, the Task Message Log is not written to in case of synchronous message processing.

Examples of synchronous web services are the Relation integration point and the Provider integration point.

3.4.1.2 Asynchronous Message Processing

Also in case of an asynchronous request, the message is validated to check if it adheres to the XSD specification. If not, a SOAP fault will be returned immediately. The message is not processed.

Valid requests are queued and will be processed in the order in which these were queued as soon as system resources are available.
Similar to File Based Integration (page 31), WS-Addressing is used to determine:

- To which endpoint URI the response message should be delivered. This is determined by the client and passed as the WS-Addressing replyTo address in the request message.
- How the request and response are correlated. The response message contains the WS-Addressing relatesTo identifier that the client program can use to correlate the response with the original request. The relatesTo attribute holds the client-generated messageId value that was passed in the request.

Passing a valid replyTo address URI and a messageId that is unique for the specific IP is the responsibility of the calling system.

In case of asynchronous message processing, result messages are written to the Task Message Log. These are passed in the response message and may also be retrieved using the messageId value.

An example of an asynchronous web service is the ClaimsIn integration point.

3.4.2 OHI Components as Web Service Client (outbound requests)

For external SOAP Web Services that are called from OHI Components applications the contracts are specified by Oracle. In order for the system to successfully connect to an external service, the contract must be implemented and the Web Service must be available.

3.4.2.1 Synchronous Message Processing

OHI Components applications execute synchronous calls to external services (outbound requests). The endpoint for the Web Services is specified as property in the application's properties file. The format of an endpoint property is *ohi.<integration_point>.endpoint.request*. For a synchronous request OHI Components applications expect an immediate response.

Other properties relevant for synchronous outbound requests are the following:

- ohi.ws.client.connectiontimeout: the timeout period that OHI Components applications use to establish a connection to an external service. It is specified in milliseconds; a value of 0

means never timeout, in that case OHI Components applications will wait (indefinitely) until the connection is established.
For example, a value of 6000 means that OHI Components applications will wait for 6000 milliseconds to establish a connection. If a connection is not established before that period expires, OHI Components applications will flag the service as being unavailable. The task for which the request needed to be send ends in an error state and can be retried / recovered from the "*View Technical Errors*" screen.

- ohi.ws.client.readtimeout: once a connection is established and the request is sent, this property specifies the timeout period OHI Components applications will wait for the server to respond to the request. It is specified in milliseconds; a value of 0 means never timeout, in that case OHI Components applications will wait (indefinitely) until a response is received. For example, a value of 6000 means that OHI Components applications will wait for a response for 6000 milliseconds after the connection was established. If a response is not received before that period expires, OHI Components applications will flag the service as being unavailable. The task for which the request needed to be send ends in an error state and can be retried / recovered from the "View Technical Errors" screen.
- ohi.ws.client.retrytimeout: OHI Components applications keep track of the state of external web services. In case a request to an external service fails (either the connection times out or the response is not received before the readtimeout expires) OHI Components applications register the service as being unavailable. OHI Components applications will not attempt to send other requests to the same service within the specified retrytimeout time frame (measured from the moment the service failed for the first time).
  If a service is registered as being unavailable,OHI Components applications send a notification to inform a system administrator so that appropriate action can be taken. Note: if a service is not available for a long time and the service is used with a high frequency, a low value for this property effectively means that OHI Components applications will send many notifications.
  A value of 0 means that OHI Components applications attempt to send each request.
  If the external service is not available, that means that the time that is set for the ohi.ws.client.connectiontimeout property is lost for each attempt. On the other hand, if the value for this property is large and the volume of requests is high then many tasks will end up in an error state (and have to be retried / recovered from the "View Technical Errors" screen).

An example of a synchronous outbound web service is the Claim Event integration point.

3.4.2.2 Asynchronous Message Processing

OHI Components applications can also execute asynchronous calls to external services (outbound requests). As is the case for synchronous services, the endpoint for an asynchronous external Web Services is specified as property in the application's properties file. The format of an endpoint property is ohi.<integration_point>.endpoint.request.

In accordance with the WS-Addressing standard, for an asynchronous request OHI Components applications expect the external system to deliver the response to the OHI Components application endpoint for which the URL is given in the request using the same messageId.

For asynchronous requests, the ohi.ws.client.connectiontimeout and ohi.ws.client.retrytimeout properties apply.

3.5 Interface Messages Log

3.5.1 Result Messages

Result messages refer to errors, warnings, or informational messages that result from batch, asynchronous and synchronous single message processing. Result messages are categorized as internal or external and are written to the Interface Messages Log (see below). The message code and text of external result messages may be included in response messages.

Each result message corresponds to a message definition that specifies the standard message text and possibly substitution parameters. Result messages for batch processes may be at batch level (for messages that do not apply to a specific main-level request message element) or at main-level element level (in which case a main-level element code / id is included as a substitution parameter so that messages can be matched to a main-level element). Likewise, messages that are specific to an item of a list are expected to include code / id as a substitution parameter so that messages can be matched to the item that they relate to.

Result message definitions include a notification indicator that indicates if, when a result message corresponding to the definition occurs during batch processing, a failure notification should be sent.

Message definitions are maintained in table OHI_MESSAGES.

3.5.2 Interface Messages Log

An interface message log is maintained in the database. It is written to during batch, (asynchronous and synchronous) single message processing and for some UI related operations. For each operation an interface message entry is created which may have zero, one or more details. The structure of the interface messages log is listed in the following sub-paragraphs.

3.5.2.1 Interface Message

Interface Messages carry the following details:

Attribute	Asynchronous Single Message	Batch	Synchronous Single Message
Subtype	ASYNC	BATCH	SYNC
Task Id	Identifier of the system task that was created for processing the request	n/a	n/a
Job Instance Id	n/a	Reference to the job execution id	n/a
Correlation Id	WS-Addressing messageId as given by the client that called the web service for invoking the asynchronous process (external identifier)	WS-Addressing messageId as given by the client that called the web service for invoking the batch process (external identifier)	n/a
Service Name	IP name. This is the same name that appears for the service in the WebLogic Console.	n/a	IP name. This is the same name that appears for the service in the WebLogic Console.
Operation Name	IP operation name as it appears in the WSDL.	Batch job name	IP operation name as it appears in the WSDL.
Request Received Time	The time that processing of the request started	The time that processing of the batch request started	The time that processing of the request started
Request Processed Time	The time that processing of the request finished	The time that processing of the batch request finished	The time that processing of the request finished
Request Message	Original request message payload that was received	Original request notification message (containing link to file that contains the actual request message) that was received.	Original request message payload that was received
Response Message	Response message that was sent.	Response notification message (containing link to file that contains the actual response message) that was sent.	Response message that was sent.

Result Code	'F' if message not processed successfully, 'S' if processed successfully.	'F' if message not processed successfully, 'S' if processed successfully.	'F' to indicate that message is not processed successfully. Synchronous messages that were processed successfully are not logged.

3.5.2.2 Interface Message Details

Zero, one or more interface message details may be logged for an interface message. The structure of the interface message details is listed in the following table:

Attribute	Description
Interface Message Id	Reference to Interface Message
Element Id	From message elementId attribute (only if result message relates to a main-level element in a batch file - otherwise n/a)
Message Code	Code of the message
Message Text	Descriptive text of the message with values substituted for the placeholders
Message Detail	Additional 'technical' detail (e.g. stack trace). Optional.

Processing of a message failed if a message code is logged that is classified as Fatal.

Note that in case of synchronous single messages only failures are logged. This is done to make message processing more efficient. The idea of not logging successfully processed messages is that the absence of a failure implies success. If processing the request was successful, the audit columns for the entity reflect the change.

3.5.3 Interface Messages Log UI page

To track message processing read-only UI pages are provided:

- An overview page that shows two sections with aggregated information, one for service messages and one for batch messages.
- Two pages that provide message details service and batch processes respectively.

The overview page allows filtering by date range; by default it will query results for the last day. The numbers are aggregated by service name, operation name and subtype. From the overview page it is possible to click through to a page that shows details for all requests of the selected IP or Batch process. The latter page also provides access to the request and response message payloads.

Messages resulting from UI operations are stored in the interface message log but are not visible through the interface messages log UI pages. Instead these are shown in specific UI pages.

3.6 Data Set Operations Integration Point

This integration point allows the user to generate the payload for a particular data set and or/and start the import for a data set. The purpose of this integration point is to allow the coordination of data sets between application environments without having to access the application screens.

This section refers to the environment that loads the data set as the 'target' environment. The environment on which the data set is generated is referred to as the 'source' environment.

This integration point supports the handling of data sets that belong to the definitions CLAIMS CONFIGURATION, CLAIMS_PROVIDERS, CLAIMS_PROCEDURES and CLAIMS_DIAGNOSES.

This integration points consists of six operations:

- Stop Dequeue
- Start Dequeue
- Build Data Set
- Save to File
- Import From File
- Import From Environment

3.6.1 Operation Requests

3.6.1.1 Stop Dequeue

This request stops claims from entering the claims flow. Note that claims that are already in the flow continue to be processed.

```
<stopDequeue/>
```

3.6.1.2 Start Dequeue

This request tells the application to accept new claims for processing.

```
<startDeqeueue>
```

3.6.1.3 Build Data Set

This request is sent to the source environment. It creates (or overwrites) the XML payload for a data set.

```
<buildDataSet
  dataSetDefinitionCode
  dataSetCode
  inclusionDate
 />
```

The application sends back a web service response either when the build fails or when the payload generation is complete.

```
<buildDataSetResponse>
  <resultMessages
    result
  >
    <resultMessage
      code
    > message text
```

The following error messages can be returned in the web service response:

Code	Sev	Message
OHI-IP-DATA-001	Fatal	Unknown combination of data set code {code} and data set definition code {code}
GEN-MIGR-008	Fatal	It is not possible to start a new build while another build or import is in progress
GEN-MIGR-010	Fatal	It is not possible to start a build with empty data set

3.6.1.4 Save to File

This request is sent to the source environment. It saves the data set to the specified file path.

```
<saveToFile
```

```
    dataSetCode
    dataSetDefinitionCode
    filePath
  />
```

The application sends back the following response:

```
  <saveToFileResponse>
    <resultMessages
      result
    >
      <resultMessage
        code
      > message text
```

Code	Sev	Message
GEN-FILE-001	Fatal	Directory does not exist or is not accessible: {directory}
OHI-IP-DATA-001	Fatal	Unknown combination of data set code {code} and data set definition code {code}
OHI-IP-DATA-003	Fatal	This data set has not been built yet.

3.6.1.5 Import From File

This request is sent to the target environment. It starts an import with the file's content as the payload.

```
  <importFromFile
    filePath
    dataSetDefinitionCode
    responseFilePath
  />
```

The application sends back a response either when the import fails or when the import is complete. The web service response only contains messages in case the input file cannot be read or the response file not be written to.

```
  <importResponse>
    <resultMessages
      result
    >
      <resultMessage
        code
      > message text
```

The following error messages can be returned in the web service response. These are messages that prevent the import from happening.

Code	Sev	Message
GEN-FILE-001	Fatal	Directory does not exist or is not accessible: {directory}
GEN-FILE-004	Fatal	Input file path does not exist {inputFilePath}
GEN-FILE-009	Fatal	File / Directory contains non permitted characters
GEN-MIGR-007	Fatal	Import file must have a .zip extension
GEN-MIGR-009	Fatal	It is not possible to start a new import while another build or import is in progress

OHI-IP-DATA-004	Fatal	Unknown data set definition code {code}

The response file contains the messages that relate to the imported content.

3.6.1.6 Import From Environment

This request is sent to the target environment. It starts the retrieval of the data set from the source environment and starts the import on the target environment.

```
<importFromEnvironment
  sourceEnvironment
  dataSetDefinitionCode
  dataSetCode
  responseFilePath
/>
```

The sourceEnvironment attribute should contain the SID of the source environment database

The matching response only relays messages of which the cause prevents the import from happening. Error messages that relate to specific items in the payload end up in the response file.

```
<importResponse>
  <resultMessages
    result
  >
    <resultMessage
      code
    > message text
```

The following error messages can be returned in the web service response.

Code	Sev	Message
GEN-FILE-001	Fatal	Directory does not exist or is not accessible: {directory}
GEN-FILE-009	Fatal	File / Directory contains non permitted characters
GEN-MIGR-009	Fatal	It is not possible to start a new import while another build or import is in progress
OHI-IP-DATA-001	Fatal	Unknown combination of data set code {code} and data set definition code {code}
OHI-IP-DATA-002	Fatal	Unknown source environment {sourceEnvironment}

The response file contains the messages that relate to the imported content.

3.6.2 Response File

The response file contains the exact same messages as displayed in the pop-up in page FN0040 Inbound Data Sets - Batch process message details.

```
<importResponseFile>
 <resultMessages
    result
  >
    <resultMessage
      code
    > message text
```

Each data set definition has a specific set of error messages that can occur. Typically these messages indicate there is an incompatibility between the items in the payload and the items that already exist on the target environment.

For example, the claims configuration data set can return the following messages:

Code	Sev	Message
CLA-MIGR-001	Info	Disabled fee schedule line ({fee schedule code}, {line procedures}, {line start date})
CLA-MIGR-002	Info	Disabled product benefit specification ({product code}, {benefit specification code}, {start date})
GEN-MIGR-001	Fatal	Cannot find {entity} with key {code or usage name} used by {entity, key}
GEN-MIGR-002	Fatal	Cannot find {entity} with Code {code} and Flex Code Definition Code {definition code} used by {entity, key}
GEN-MIGR-003	Fatal	Cannot find OhiTable with Name {name} used by {entity, key}
GEN-MIGR-004	Fatal	Cannot find Signature with Name {name} and Subtype {subtype} used by {entity, key}
GEN-MIGR-005	Fatal	Cannot find usage for dynamic field or record {usage name} on {table name} used by {entity, key}
GEN-MIGR-006	Fatal	The dynamic field usage ({name}, {table name}) is not compatible with the existing configuration

3.7 Result Messages

3.7.1 Indicating Success or Failure

In the responses of Integration Points, the common element <resultMessages> is included to indicate whether or not the request message was processed successfully. This element has an attribute called 'result' which can be either 'S' to indicate success or 'F' to indicate failure.

For example, an Integration Point's response message to a successfully processed request message contains:

```
<resultMessages result='S'/>
```

This element is included in the root of the Integration Point's response element. It may be included multiple times, if the request message contained multiple 'transaction units'. The response message may then (e.g.) indicate that some transaction units were processed successfully while processing of others failed.

3.7.2 Result Messages

Inside the element <resultMessages> (described above) there may be zero, one or more result messages. The purpose of these messages is to clarify why a request message was not processed successfully. The message text contains the message text in which the substitution parameters have been set.

For example, the Claims In Integration Point's response message to a request message which failed to process successfully could contain :

```
<resultMessages result='F'>
    <resultMessage code='CLA-IP-CLAI-010'>
```

```
        CLA-IP-CLAI-010: The specified product code ABC is unknown
      </resultMessage>
  </resultMessages>
```

Result messages are common or specific:

- **Integration Point Specific messages** can only occur in the response of a specific Integration Point. Such messages are described in the description of the concerned Integration Point. The example above shows a result message specific to the Claims In Integration Point.
- **Messages common across Integration Points** can occur in the responses of many Integration Points. These messages relate to common functionality, like the use of dynamic fields. These message are described below.

Messages common across Integration Points:

Code	Severity	Message
GEN-ACRE-001	Fatal	Access restriction code {code} is unknown. Request cannot be processed
GEN-TRAS-001	Fatal	A reference may only be provided in combination with a transaction source
GEN-TRAS-002	Fatal	Transaction source code {code} is unknown
GEN-CURR-001	Fatal	Currency code {code} is unknown
GEN-DYNA-001	Fatal	The dynamic field: {dynamicFieldUsageName} should have unique values. There is already a record with value: {value}
GEN-DYNA-002	Fatal	{dynamicFieldUsageName}: There is already a value present in this period of time ({startDate} - {endDate})
GEN-DYNA-003	Fatal	Cannot insert same value for the flex code in case the dynamic field is multivalue
GEN-DYNA-004	Fatal	Dynamic field {dynamicFieldUsageName} is not time valid. Request cannot be processed
GEN-DYNA-005	Fatal	Dynamic field {dynamicFieldUsageName} may have only one single value. Request cannot be processed
GEN-DYNA-006	Fatal	Dynamic field flex code definition code {code} is unknown. Request cannot be processed
GEN-DYNA-007	Fatal	The flex code {code} is unknown to dynamic field {dynamicFieldUsageName}. Request cannot be processed
GEN-DYNA-008	Fatal	Dynamic field name {dynamic field usage name} is unknown. Request cannot be processed
GEN-DYNA-010	Fatal	{dynamicFieldUsageName} should have a value and a {startDate}
GEN-DYNA-011	Fatal	{dynamicFieldUsageName} value {value} does not belong to flex code definition {code}
GEN-DYNA-012	Fatal	{dynamicFieldUsageName}: the same value {value} is present in another period ({startDate} - {endDate})
GEN-DYNA-013	Fatal	Only one value allowed for {dynamicFieldUsageName}
GEN-DYNA-015	Fatal	The field {dynamicFieldUsageName} is not allowed to be empty
GEN-DYNA-016	Fatal	Dynamic Record Definition with type {dynamicFieldUsageName} could not be found
GEN-DYNA-017	Fatal	Dynamic record definition {dynamicFieldUsageName} does not define field {flexCodeFieldUsageCode}
GEN-DYNA-018	Fatal	Usage {dynamicFieldUsageName} can only have a record when the condition defined evaluates to true
GEN-DYNA-019	Fatal	Usage {dynamicFieldUsageName} should have at least one record
GEN-DYNA-020	Fatal	Key field {flexCodeFieldUsageCode} should have unique value for Dynamic record {dynamicFieldUsageName}

GEN-DYNA-021	Fatal	Dynamic Records should specify the value for key field. Dynamic Record {dynamicFieldUsageName} does not specify value for key {flexCodeFieldUsageCode}
GEN-DYNA-022	Fatal	Dynamic record {dynamicFieldUsageName} is not time valid. Request cannot be processed
GEN-PROC-001	Fatal	Value provided is not of type Date
GEN-PROC-002	Fatal	Value provided is not of type Number

Besides these messages, business rule messages may also occur in the responses of Integration Points. Business rule messages are raised in the validation layer of OHI Components applications and are common to both the User Interface Pages and the Integration Points. For example, business rule GEN-TMVL-001: "The start date should lie before the end date for {dynamicFieldUsageName}".

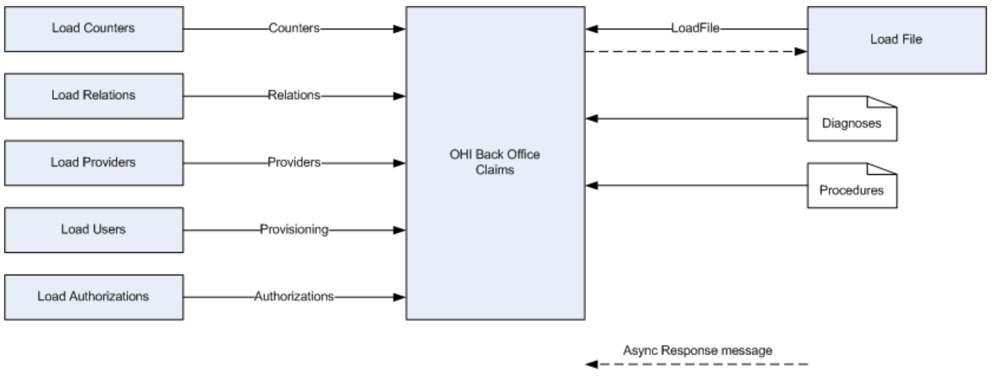
Lastly, technical error messages may be returned through the responses of Integration Points. For example, database message GEN-ORA-01400: "DESCR" column is mandatory for table "PRI_FEE_SCHEDULES".

3.8 Integration Testing

Testing an application in a Services based environment requires the services to be available during certain tests. These services can be provided by real systems, or by simulation. A system that simulates an external system is referred to as a test double or mock service.
Before the OHI Claims flow can be tested, a certain amount of data will need to be available in OHI Claims. In this section, these are categorized as Prerequisite services. The services that are required during Flow testing are categorized as Operational services.

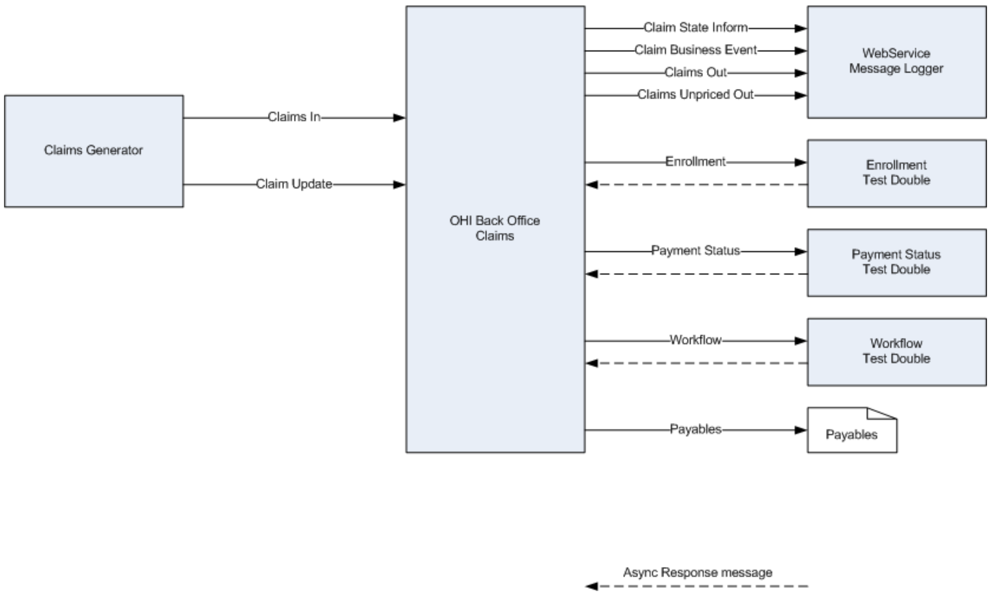
3.8.1 Environment for Prerequisite Services

The diagram below shows the services that can be tested individually. They are not necessarily connected to the Claims flow.



3.8.2 Environment for Operational Services

The diagram below shows the services required for testing the Claims flow.



3.9 Web Service Versioning

When customers start to integrate with OHI Components applications Web Services, there is a need for versioning these. Parts of a Web Service that are likely to change and for which version control needs to be put in place are:

- The (abstract) WSDLs; these are typically fairly stable.
- The XML Schema content that describes the types for the service's message definitions; these are more likely to change.

3.9.1 Compatibility

A new version of a Web Service contract that continues to support client software that was designed to work with the previous version is said to be backward-compatible. Examples of backward-compatible changes to an XSD are:

- The addition of an optional element.
- Changing an existing element from being required to optional.

These have no impact on existing consumers of the service.

If a contract changes in such a way that it can no longer be used by existing consumers of the service without making changes to the consumer programs, it is an incompatible change. Examples of incompatible changes are:

- Renaming or removing an existing WSDL operation.
- Adding a new required XML Schema element or attribute to a message definition.
- Renaming an optional or required XML Schema element or attribute in a message definition.
- Removing an optional or required XML Schema element or attribute from a message definition.

3.9.2 Flexible Versioning Strategy

The selected versioning strategy is known in the literature as Flexible. Any incompatble change results in a new version of the service contract and the contract is designed to support backwards compatibility. Any change that breaks the existing contract results in a new version.

For XML Schema's and WSDLs versions are identified using a major and minor version in notation "major.minor". Conventions according to the compatibility guarantee:

- A minor version is expected to be backward compatible with other minor versions that are associated with a major version.
- A major version breaks backward compatibility.

The versioning strategy for SOAP services used by OHI Components applications can be characterized as follows:

- A compatible change leads to a minor change of the version number. The namespace(s) remain unchanged, hence supporting backward compatibility.
- An incompatible change leads to a major change of the version number.
- Major versions will be identified in XML namespaces of top-most XML schemas (and later WSDLs). As a result, a major change will require applications that use the schema or web service to be upgraded.

Version identification example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dt="http://healthinsurance.oracle.com/datatypes" xmlns="http://
healthinsurance.oracle.com/fileimport/v1"
targetNamespace="http://healthinsurance.oracle.com/fileimport/v1"
elementFormDefault="qualified" attributeFormDefault="unqualified"
 version="1.0">
```

For a compatible change, only the version number will be affected whereas the namespace declarations remain unchanged:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dt="http://healthinsurance.oracle.com/datatypes" xmlns="http://
healthinsurance.oracle.com/fileimport/v1"
targetNamespace="http://healthinsurance.oracle.com/fileimport/v1"
elementFormDefault="qualified" attributeFormDefault="unqualified"
 version="1.1">
```

An incompatible change results in changes to both the version number as well as the version identifier in the namespaces:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dt="http://healthinsurance.oracle.com/datatypes" xmlns="http://
healthinsurance.oracle.com/fileimport/v2"
targetNamespace="http://healthinsurance.oracle.com/fileimport/v2"
elementFormDefault="qualified" attributeFormDefault="unqualified"
 version="2.0">
```

Note that the DataTypes schema that holds re-usable simple and complex types, does not have a version identifier. The types in there are re-used often and existing types are expected to be stable. Given the amount of re-use in all OHI SOAP Services, it is not likely for types declared in DataTypes.xsd to be removed.

3.9.3 Resolving Version Conflicts across Releases

Consider the following versions for the ClaimImport.xsd in distinct releases of an application:

Release	XSD Version
2.12.1.0.0	1.0
2.12.2.0.0	1.1

That means that the ClaimImport.xsd was changed (backward compatible or minor change) from release 2.12.1.0.0 to 2.12.2.0.0. If another minor change is required on both these versions that could (in theory) result in the following situation:

Release	XSD Version
2.12.1.0.1	1.1
2.12.2.0.1	1.2

To reflect the change in the ClaimImport.xsd in both versions of the application the minor version indicator in either release would be incremented. The result is that the ClaimImport.xsd in releases 2.12.2.0.0 and 2.12.1.0.1 has the same version number; that would wrongfully indicate that the XSDs in these versions are similar.

This (rare) versioning conflict is resolved by introducing a third indicator, referred to as revision in notation "major.minor.revision". Thus, the result would be:

Release	XSD Version
2.12.1.0.1	1.0.1
2.12.2.0.1	1.2

4 HTTP API Integration Points

4.1 HTTP API resources in an OHI Components application

The context root for any HTTP API resource in an OHI Components application is "/api". An overview of all HTTP API resources available in a running system is available under "/api/doc".

The page that is generated also provides access to RAML specifications for each resource. RAML stands for **RESTful API Modeling Language**. A RAML document is a formal specification of an HTTP API that is readable by both humans and computers. In many cases the RAML specification contains usage examples for the methods that are supported by the API.

4.2 Attribute Handling

Each integration point request message contains data values of a root element (e.g. organization or organization provider) that have been created or updated in a source system. Within each message are several categories of data such as simple entity-level attributes, lists of non-time valid details and lists of time-valid details, dynamic fields and dynamic records. This section describes how each category of data is handled by OHI applications and also covers guidelines for handling differences in data categories between source systems and OHI applications.

The way that an OHI application handles requests is based on the principle that the copy of information in the OHI application is to be kept up-to-date with the (master) information in the system of record. It is not required that the OHI application is informed of every update in the system of record; only the values at the time of creating the requests are important. For example, if an interface periodically creates requests for all outstanding additions and updates, only the values of source system data at the time that the interface is run, need to be sent. Whether a record has been updated several times or once since the last interface run is irrelevant.

If an existing record is sent again and it contains the exact same data that is already stored in the system, the existing data will not be updated. This means that the audit columns in the database will also remain unchanged.

4.2.1 Single Value Attributes

Single value attributes are fields that can have only one single value and the value does not have a start and end date. Single value attributes can be represented in the requests as attributes (for example name in the organizationProvider request) or as elements (for example parentOrganizationProvider in the organizationProvider request). When the application creates a new record, single value attributes are handled as follows:

- Represented as attributes: if the attribute is not included in the request, then the value of the corresponding attribute in the new record will be set to null; if the attribute is included in the request, then the corresponding attribute in the new record will be set to the specified value.
- Represented as elements: if the element is not included in the request, then the value of the corresponding attribute in the new record will be set to null; if the element is included in the request, then the corresponding attribute in the new record will be set to the specified value.

When the application updates a record, a single value attribute is handled as follows:

- Represented as attributes: if the attribute is not included in the request, then the existing value in the application remains untouched; if the attribute is included in the request, then the attribute value is updated with the specified value. In order to send in an update that clears the value, the update request should include the attribute with an empty value.

- Represented as elements: if the element is not included in the request, then the existing value in the application remains untouched; if the element is included in the request, then the attribute value is updated with the specified value. In order to send in an update that clears the value, the update request should include the element without any attributes and values (empty element).

For example, consider a new organization being added in the system of record. Because the OHI application keeps a local copy of organization records, the system of record sends the following request to the OHI application:

```
<organization
  code="1333"
  name="Jones Administration"
  outputLanguage="en"
  businessPhoneNumber="0301111111"
/>
```

Since this is the first time that the organization with code "1333" is being sent, the OHI application creates a new organization record (relation of subtype organization) with only code, name, language and business phone number having values. All other attributes in the new record in the OHI application will be null. The organization is updated in the source system; the business phone number is changed from "0301111111" to "0301111112". The system of record sends the following request to the OHI application:

```
<organization
  code="1333"
  businessPhoneNumber="0301111112"
/>
```

Since there is already an organization with this code, the OHI application will update the organization record with code "1333", setting the business phone number to "0301111112". The name and the language will not be changed or set to null.

External Interface Design Notes

If a given type of data is a single value attribute in an Integration Point message definition and a time valid detail list in the source system, the external interface is expected to only send the latest / current value.

If a given type of data is a single value attribute in an Integration Point message definition and a non time valid detail list or otherwise not a one-to-one match in the source system, the external interface will require (dynamic) logic needed to determine what value should be provided.

4.2.1.1 Amount and Currency

Amount and currency are two attributes that belong together, so they are always represented together in a separate element. For example:

```
<authorizedAmount
  amount="100"
  currencyCode="USD"
/>
```

If the element is not included in the request, then the existing values for amount and currency in the application remain untouched; if the element is included in the request, then the values are updated with the specified values. In order to send in an update that clears the values, the update request should include the element without any attributes and values (empty element).

4.2.2 Details

Details can be non time valid (i.e. each detail record does not have start and end dates) such as provider titles. Details can also be time valid (i.e. each item of the list has a start and an end date), such as provider group affiliations. Both types of details are handled as follows:

When the OHI application creates new records (for an entity with a detail list):

- If the detail list element is not included, no details are added
- If a detail list element is included, one detail record for each included detail item is created

When the OHI application is updating an existing record (for an entity with a detail list):

- If the list element is not included, no changes to current details are made
- If a list element is included, included detail items completely replace current detail records (in effect, all current detail records that have not been resent (i.e. for which there is no detail item with exactly matching values) are removed and one detail record for each included detail item (that is not a 'resend') is created)
    - Note that completely replacing an existing list can have a different meaning depending on the root element. Provider group affiliations can for example be specified for root element organization provider or for root element provider group. If specified for an organization provider, it means that all provider group affiliations of that provider (in all provider groups) are replaced by the new list. If specified for a provider group, it means that all provider group affiliations of that provider group are replaced by the new list.
- If an empty list element is included, all current detail records are removed.

4.2.3 Dynamic Fields and Records

This section describes the default behavior of integration points that accept dynamic field and record values. The configuration of the relevant dynamic field usages drive how the integration point processes a request with new dynamic field and record values. The driving aspects are the usage type (field, code or record), whether the field can have multiple values and whether the field value is time valid.

4.2.3.1 Dynamic Fields

Dynamic field values can be created and updated through integration points. How the values are supplied in the request messages depends on the way the dynamic fields are configured in the application. Dynamic fields can be configured as:

- Single-Value Non-Time-Valid
- Single-Value Time-Valid
- Multi-Value Non-Time-Valid
- Multi-Value Time-Valid

4.2.3.1.1 Single-Value Non-Time-Valid Fields

Message Definition

A single-value non-time-valid free field is represented as an attribute of the element it belongs to, with the name of the attribute being the same as the corresponding dynamic field usage name. The same applies to a single-value non-time-valid flex code field that is configured as a flex code definition.
A single-value non-time-valid flex code field that is configured as a flex code set is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element has flexCodeDefinitionCode as attribute and the dynamic field value as text content.

Update Behavior

When the application updates a record, a single-value non-time-valid free field and a single-value non-time-valid flex code field that is configured as a flex code definition are handled as follows: if the attribute is not included in the request, then the existing value in the application remains untouched; if the attribute is included in the request, then the value is updated with the specified value. In order to send in an update that clears the value, the update request should include the attribute with an empty value.

A single-value non-time-valid flex code field that is configured as a flex code set is handled as follows when updating a record: if the element is not included in the request, then the existing value in the application remains untouched; if the element is included in the request, then the value is updated with the specified value. In order to send in an update that clears the value, the update request should include the sub-element without the flexCodeDefinitionCode attribute and the text content.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- dateOfBirth: single-value non-time-valid free field
- married: single-value non-time-valid flex code field (configured as a flex code definition)
- specializedProcedure: single-value non-time-valid flex code field (configured as a flex code set)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
  nameFormatCode="NFDFLT"
  outputLanguageCode="EN"
  startDate="2010-01-01"
  dateOfBirth="1975-06-06"
  married="Y"
>
  <specializedProcedure
    flexCodeDefinitionCode="CPT_CODES"
  >
    33010
  </specializedProcedure>
</individualProvider>
```

In order to update the married value from Y to N and leave all other fixed field values and dynamic field values intact, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  married="N"
>
</individualProvider>
```

To clear the values of the dynamic fields that were created in the examples above, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  dateOfBirth=""
  married=""
>
  <specializedProcedure/>
</individualProvider>
```

4.2.3.1.2 Single-Value Time-Valid Fields

Message Definition

A single-value time-valid free field is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <value> sub-elements of its own. The <value> sub-element has startDate and endDate as attributes and the dynamic field value as text content.

The same applies to a single-value time-valid flex code field that is configured as a flex code definition.
A single-value time-valid flex code field that is configured as a flex code set is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <value> sub-elements of its own. The <value> sub-element has startDate, endDate and flexCodeDefinitionCode as attributes and the dynamic field value as text content.

Update Behavior

When the application updates a record, a single-value time-valid field is handled as follows: if the sub-element is not included in the request, then the existing value(s) in the application remain(s) untouched; if the sub-element is included in the request, then the existing value(s) is/are replaced by the value(s) specified in the request. In order to send in an update that clears the existing value(s), the update request should include the sub-element without any <value> sub-elements.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- contractReferences: single-value time-valid free field
- married: single-value time-valid flex code field (configured as a flex code definition)
- specializedProcedures: single-value time-valid flex code field (configured as a flex code set)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
  nameFormatCode="NFDFLT"
  outputLanguageCode="EN"
  startDate="2010-01-01"
>
  <contractReferences>
    <value
      startDate="2010-01-01"
      endDate="2012-06-31"
    >
      1111
    </value>
    <value
      startDate="2012-07-01"
    >
      1112
    </value>
  </contractReferences>
  <married>
    <value
      startDate="2012-01-01"
      endDate="2012-06-31"
    >
      Y
    </value>
  </married>
  <specializedProcedures>
    <value
      flexCodeDefinitionCode="CPT_CODES"
      startDate="2010-01-01"
      endDate="2011-12-31"
    >
      33010
    </value>
    <value
      flexCodeDefinitionCode="ICD10_PROCEDURES"
      startDate="2012-01-01"
    >
      C2161ZZ
    </value>
  </specializedProcedures>
</individualProvider>
```

In order to replace the contract references 1111 and 1112 by contract reference1113 and leave all other fixed field values and dynamic field values intact, the following individual provider request message is used:

```
  <individualProvider
    code="1234567890"
    flexCodeDefinitionCode="US_PROVIDER"
  >
    <contractReferences>
      <value
        startDate="2010-01-01"
        endDate="2012-06-31"
      >
        1113
      </value>
    </contractReferences>
  </individualProvider>
```

To clear the values of the dynamic fields that were created in the examples above, the following individual provider request message is used:

```
  <individualProvider
    code="1234567890"
    flexCodeDefinitionCode="US_PROVIDER"
  >
    <contractReferences/>
    <married/>
    <specializedProcedures/>
  </individualProvider>
```

4.2.3.1.3 Multi-Value Non-Time-Valid Fields

Message Definition

A multi-value non-time-valid free field is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <value> sub-elements of its own. The <value> sub-element has the dynamic field value as text content. The same applies to a multi-value non-time-valid flex code field that is configured as a flex code definition.
A multi-value non-time-valid flex code field that is configured as a flex code set is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <value> sub-elements of its own. The <value> sub-element has flexCodeDefinitionCode as attribute and the dynamic field value as text content.

Update Behavior

When the application updates a record, a multi-value non-time-valid field is handled as follows: if the sub-element is not included in the request, then the existing value(s) in the application remain(s) untouched; if the sub-element is included in the request, then the existing value(s) is/are replaced by the value(s) specified in the request. In order to send in an update that clears the existing value(s), the update request should include the sub-element without any <value> sub-elements.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- contractReferences: multi-value non-time-valid free field
- hobbies: multi-value non-time-valid flex code field (configured as a flex code definition)

- specializedProcedures: multi-value non-time-valid flex code field (configured as a flex code set)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
  nameFormatCode="NFDFLT"
  outputLanguageCode="EN"
  startDate="2010-01-01"
>
  <contractReferences>
    <value>
      1111
    </value>
    <value>
      1112
    </value>
  </contractReferences>
  <hobbies>
    <value>
      Soccer
    </value>
    <value>
      Baseball
    </value>
  </hobbies>
  <specializedProcedures>
    <value
      flexCodeDefinitionCode="CPT_CODES"
    >
      33010
    </value>
    <value
      flexCodeDefinitionCode="ICD10_PROCEDURES"
    >
      C2161ZZ
    </value>
  </specializedProcedures>
</individualProvider>
```

In order to replace the hobbies Soccer and Baseball by hobby Basketball and leave all other fixed field values and dynamic field values intact, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <hobbies>
    <value>
      Basketball
    </value>
  </hobbies>
</individualProvider>
```

To clear the values of the dynamic fields that were created in the examples above, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <contractReferences/>
  <hobbies/>
  <specializedProcedures/>
</individualProvider>
```

4.2.3.1.4 Multi-Value Time-Valid Fields

Multi-value time-valid fields are handled in the same manner as single-value time-valid fields.

4.2.3.2 Dynamic Records

Many of the entities that can be sent in through an integration point can be extended with dynamic records. The values for these records can be set through the integration points as well. How the values are supplied in the request messages depends on the way the dynamic records are configured in the application. Dynamic records can be configured as:

- Single-Value Non-Time-Valid
- Single-Value Time-Valid
- Multi-Value Non-Time-Valid
- Multi-Value Time-Valid

4.2.3.2.1 Single-Value Non-Time-Valid Records

Message Definition

A single-value non-time-valid dynamic record is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element has the dynamic record flex code field usages (representing the columns of the dynamic record) that are configured as free fields, as attributes with the names of the attributes being the same as the corresponding flex code field usage names. The same applies to the dynamic record flex code field usages that are configured as flex code definitions. Dynamic record flex code field usages that are configured as flex code sets are represented as sub-elements of the sub-element described above, with the names of the sub-elements being the same as the corresponding dynamic record flex code field usage names. The sub-elements have flexCodeDefinitionCode as attribute and the dynamic field value as text content.

Update Behavior

When the application updates a record, a single-value non-time-valid record is handled as follows: if the sub-element is not included in the request, then the existing value(s) in the application remain(s) untouched; if the sub-element is included in the request, then the existing value(s) is/are replaced by the value(s) specified in the request (attributes that are not in the request are set to null). In order to send in an update that clears the existing value(s), the update request should include the sub-element without any attributes and sub-elements.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- accreditation: single-value non-time-valid dynamic record, having the following dynamic record flex code field usages:
  - accreditationName (configured as a flex code definition)
  - status (configured as a flex code definition)
  - accreditationDate (configured as a free field)
  - accreditedBy (configured as a flex code set)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
  nameFormatCode="NFDFLT"
  outputLanguageCode="EN"
  startDate="2010-01-01"
>
  <accreditation
```

```
    accreditationName="Physical Therapy"
    status="Accredited"
    accreditationDate="2011-01-01"
  >
    <accreditedBy
      flexCodeDefinitionCode="US_PROVIDER"
    >
      1000000000
    </accreditedBy>
  </accreditation>
</individualProvider>
```

In order to replace the accreditation with accreditation name Physical Therapy by an accreditation with accreditation name Speech Therapy and leave all other values intact, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <accreditation
    accreditationName="Speech Therapy"
    status="Accredited"
    accreditationDate="2011-01-01"
  >
    <accreditedBy
      flexCodeDefinitionCode="US_PROVIDER"
    >
      1000000000
    </accreditedBy>
  </accreditation>
</individualProvider>
```

To clear the values of the dynamic record that was created in the examples above, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <accreditation/>
</individualProvider>
```

4.2.3.2.2 Single-Value Time-Valid Records

Message Definition

A single-value time-valid dynamic record is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <record> sub-elements of its own (representing the rows of the dynamic record). The <record> sub-elements have startDate and endDate as attributes. The dynamic record flex code field usages (representing the columns of the dynamic record) that are configured as free fields, are represented as attributes on the <record> sub-elements with the names of the attributes being the same as the corresponding flex code field usage names. The same applies to the dynamic record flex code field usages that are configured as flex code definitions.
Dynamic record flex code field usages that are configured as flex code sets are represented as sub-elements of the <record> elements, with the names of the sub-elements being the same as the corresponding dynamic record flex code field usage names. The sub-elements have flexCodeDefinitionCode as attribute and the dynamic field value as text content.

Update Behavior

When the application updates a record, a single-value time-valid record is handled as follows: if the sub-element is not included in the request, then the existing value(s) in the application remain(s) untouched; if the sub-element is included in the request, then the existing value(s) is/are replaced by the value(s) specified in the request (attributes that are not in the request are set to null). In order to send in an update that clears the existing value(s), the update request should include the sub-element without any <record> sub-elements.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- accreditations: single-value time-valid dynamic record, having the following dynamic record flex code field usages:
  - accreditationName (configured as a flex code definition)
  - status (configured as a flex code definition)
  - accreditationDate (configured as a free field)
  - accreditedBy (configured as a flex code set)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
  nameFormatCode="NFDFLT"
  outputLanguageCode="EN"
  startDate="2010-01-01"
>
  <accreditations>
    <record
      accreditationName="Physical Therapy"
      status="Accredited"
      accreditationDate="2011-01-01"
      startDate="2011-01-01"
      endDate="2011-06-06"
    >
      <accreditedBy
        flexCodeDefinitionCode="US_PROVIDER"
      >
        1000000000
      </accreditedBy>
    </record>
    <record
      accreditationName="Occupational Therapy"
      status="Accredited"
      accreditationDate="2011-07-07"
      startDate="2012-01-01"
    >
      <accreditedBy
        flexCodeDefinitionCode="HEALTH_ACCREDITOR"
      >
        Home Health Accreditation Commission
      </accreditedBy>
    </record>
  </accreditations>
</individualProvider>
```

In order to replace the accreditations with accreditation names Physical Therapy and Occupational Therapy by an accreditation with accreditation name Speech Therapy and leave all other values intact, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <accreditations>
    <record
      accreditationName="Speech Therapy"
      status="Accredited"
      accreditationDate="2011-01-01"
      startDate="2011-01-01"
```

```
        endDate="2011-06-06"
      >
        <accreditedBy
          flexCodeDefinitionCode="US_PROVIDER"
        >
          1000000000
        </accreditedBy>
      </record>
    </accreditations>
  </individualProvider>
```

To clear the values of the dynamic record that was created in the examples above, the following individual provider request message is used:

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
>
  <accreditations/>
</individualProvider>
```

4.2.3.2.3 Multi-Value Non-Time-Valid Records

Message Definition

A multi-value non-time-valid dynamic record is represented as a sub-element of the element it belongs to, with the name of the sub-element being the same as the corresponding dynamic field usage name. The sub-element can have one or more <record> sub-elements of its own (representing the rows of the dynamic record). The <record> sub-elements have the dynamic record flex code field usages (representing the columns of the dynamic record) that are configured as free fields, as attributes with the names of the attributes being the same as the corresponding flex code field usage names. The same applies to the dynamic record flex code field usages that are configured as flex code definitions.
Dynamic record flex code field usages that are configured as flex code sets are represented a sub-elements of the <record> elements, with the names of the sub-elements being the same as the corresponding dynamic record flex code field usage names. The sub-elements have flexCodeDefinitionCode as attribute and the dynamic field value as text content.

Update Behavior

When the application updates a record, a single-value time-valid record is handled as follows: if the sub-element is not included in the request, then the existing value(s) in the application remain(s) untouched; if the sub-element is included in the request, then the existing value(s) is/are replaced by the value(s) specified in the request (attributes that are not in the request are set to null). In order to send in an update that clears the existing value(s), the update request should include the sub-element without any <record> sub-elements.

Examples

Consider the example below of an individual provider request message for the creation of a new individual provider, where the following dynamic field usages are configured on the providers table:

- accreditations: single-value time-valid dynamic record, having the following dynamic record flex code field usages:
  - accreditationName (configured as a flex code definition)
  - status (configured as a flex code definition)
  - accreditedBy (configured as a flex code definition)
  - accreditationDate (configured as a free field)

```
<individualProvider
  code="1234567890"
  flexCodeDefinitionCode="US_PROVIDER"
  name="Smith"
```

```
    nameFormatCode="NFDFLT"
    outputLanguageCode="EN"
    startDate="2010-01-01"
  >
    <accreditations>
      <record
        accreditation="Physical Therapy"
        status="Accredited"
        accreditedBy="Home Health Accreditation Commission"
        accreditationDate="2011-01-01"
      />
      <record
        accreditation="Occupational Therapy"
        status="Accredited"
        accreditedBy="Home Health Accreditation Commission"
        accreditationDate="2011-07-07"
      />
    </accreditations>
  </individualProvider>
```

In order to replace the accreditations with accreditation names Physical Therapy and Occupational Therapy by an accreditation with accreditation name Speech Therapy and leave all other values intact, the following individual provider request message is used:

```
  <individualProvider
    code="1234567890"
    flexCodeDefinitionCode="US_PROVIDER"
    name="Smith"
    nameFormatCode="NFDFLT"
    outputLanguageCode="EN"
    startDate="2010-01-01"
  >
    <accreditations>
      <record
        accreditation="Speech Therapy"
        status="Accredited"
        accreditedBy="Home Health Accreditation Commission"
        accreditationDate="2011-01-01"
      />
    </accreditations>
  </individualProvider>
```

To clear the values of the dynamic record that was created in the examples above, the following individual provider request message is used:

```
  <individualProvider
    code="1234567890"
    flexCodeDefinitionCode="US_PROVIDER"
  >
    <accreditations/>
  </individualProvider>
```

4.2.3.2.4 Multi-Value Time-Valid Records

Multi-value time-valid records are handled in the same manner as single-value time-valid records.

4.2.4 Errors

Error messages related to attribute handling, that are common across integration points, are specified in Response Messages (page 59).

4.3 Response Messages

4.3.1 Indicating Success

OHI HTTP API services make use of HTTP status codes in the HTTP headers to indicate a success. The following table lists these status codes:

Code	Meaning	Description
200	OK	Request succeeded for GET calls. May also be used for DELETE calls that complete synchronously
201	Created	Request succeeded for creation of a resource
202	Accepted	Request accepted for asynchronous POST or DELETE calls
204	No Content	Request succeeded for updating or deleting a resource

In the case of successfully creating a resource next to a 201 status code, the server returns the Location URI. In the HTTP specification, it is optional whether the server sends the representation of the newly created resource. In many cases, it will send a full response body as it may contain further processing instructions for the client in the form of hypermedia links.

In the case of successfully updating a resource, according to the HTTP specification the server has to respond in one of the following ways:

1. Header "HTTP 200 OK", accompanied by a full response body or
2. Header "HTTP 204 No Content", without any response body.

By default, OHI systems will send "HTTP 200 OK" and a full response body. After updating a resource the server does not return a Location URI.

In the case of successfully deleting a resource, according to the HTTP specification the server has to respond in one of the following ways:

1. Header "HTTP 200 OK" if the response includes an entity describing the status; OHI systems use this option if a sub-entity was deleted to return the modified (master) entity.
2. Header "HTTP 202 Accepted" if the action has not yet been enacted; OHI systems use this option if the entity is deleted at a later stage.
3. Header "HTTP 204 No Content" if the action has been enacted but the response does not include an entity; OHI systems use this option if the entity was deleted successfully

Requests that GET one or more resources will have a response structure specific to the integration point.

4.3.2 Links

When responding to requests, OHI Components applications may add URI links to the response message payload. Links have the following attributes:

- rel: description for the type of link
- type: the media type for the link, e.g. 'application/xml'
- href: the actual hypermedia reference that the client may activate

OHI Components applications return relative URI links that start with the path of the RESTful service that generated the response. It is expected that the client prepends these with the protocol identifier (e.g. HTTPS), domain, port and the OHI Components application HTTP API context root (i.e. "/api").

The following link types are distinguished:

Link type	Description
self	Reference to a specific resource, e.g. /api/activities/{activityId}.
file	Reference to a file that will be streamed to the client when the link is activated.
messages	List of messages resulting from processing a request or activity.
first	For pagination of results: for navigation to first 'page'. Only applicable if the current 'page' is not the first.
next	For pagination of results: for navigation to next 'page'. Only applicable if the list contains additional items.
prev	For pagination of results: for navigation to previous 'page'. Only applicable if the current 'page' is not the first.
actions/...	Denotes actions that are executed when the link is activated. For example: 'actions/startprocessing' to start an activity.

4.3.3 Pagination

An HTTP API service response to GET resource request may support pagination to handle large data volumes if specified as a requirement for the integration point. Additional information must be supplied along with the Integration point's URI regarding the "offset" and "limit". By default, the Offset is set to 0 and Limit is set at 50. i.e if only the URI is specified, first 50 resources will be returned in the response along with the links "next" and "prev" as applicable.

Offset: Specifies the nth record from which the subset is desired
Limit: Specifies the number of elements to be returned

Example URI to fetch next 50 records.

```
<Integration Point URI>?offset=51&&limit=50
```

4.3.4 Indicating Failure

An HTTP API service response message to an unsuccessfully processed request message will have the following structure along with appropriate HTTP status codes unless it is explicitly required to have more information to be part of the response of the integration point:

```
<resultMessages result='F'>
    <resultMessage code= ''>
      messageText
    </resultMessage>
</resultMessages>
```

The following table lists the HTTP status codes for indicating failure:

Code	Meaning	Description
400	Bad Request	The request could not be understood by the server due to malformed syntax (e.g. when unmarshalling of the request failed)

401	Unauthorized	Request failed because the user is not authenticated
404	Not found	The server has not found anything matching the Request URI
415	Unsupported Media Type	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method
422	Unprocessable Entity	The syntax of the request is correct (thus a 400 (Bad Request) status code is inappropriate) but the server was unable to process the contained instructions because the data violates business rules
500	Internal Server Error	Something went wrong on the server, check server status and logs and/or report the issue

4.3.5 Failure Result Messages

Result messages are common or specific. Inside the element <resultMessages> (described above) there may be zero, one or more result messages.The purpose of these messages is to clarify why a request message was not processed successfully. The message text contains the message text in which the substitution parameters have been set.

For example, the Activity Integration Point's synchronous response message which failed to start the activity process as activity code was unknown would be:

```
<resultMessages result='F'>
    <resultMessage code='ACT-IP-ACTY-001'>
      ACT-IP-ACTY-001: Activity code abc is unknown
    </resultMessage>
</resultMessages>
```

- **Integration Point Specific messages** can only occur in the response of a specific Integration Point. Such messages are described in the description of the concerned Integration Point.
- **Messages common across Integration Points** can occur in the responses of many Integration Points. These messages relate to common functionality, like the use of dynamic fields. These messages are described below.

Messages common across Integration Points:

Code	Severity	Message
GEN-ACRE-001	Fatal	Access restriction code {code} is unknown. Request cannot be processed
GEN-TRAS-001	Fatal	A reference may only be provided in combination with a transaction source
GEN-TRAS-002	Fatal	Transaction source code {code} is unknown
GEN-CURR-001	Fatal	Currency code {code} is unknown
GEN-DYNA-001	Fatal	The dynamic field: {dynamicFieldUsageName} should have unique values. There is already a record with value: {value}

GEN-DYNA-002	Fatal	{dynamicFieldUsageName}: There is already a value present in this period of time ({startDate} - {endDate})
GEN-DYNA-003	Fatal	Cannot insert same value for the flex code in case the dynamic field is multivalue
GEN-DYNA-006	Fatal	Dynamic field flex code definition code {code} is unknown. Request cannot be processed
GEN-DYNA-007	Fatal	The flex code {code} is unknown to dynamic field {dynamicFieldUsageName}. Request cannot be processed
GEN-DYNA-009	Fatal	{endDate} should be later than or the same as {startDate} for {dynamicFieldUsageName}
GEN-DYNA-010	Fatal	{dynamicFieldUsageName} should have a value and a {startDate}
GEN-DYNA-011	Fatal	{dynamicFieldUsageName} value {value} does not belong to flex code definition {code}
GEN-DYNA-012	Fatal	{dynamicFieldUsageName}: the same value {value} is present in another period ({startDate} - {endDate})
GEN-DYNA-015	Fatal	The field {dynamicFieldUsageName} is not allowed to be empty
GEN-DYNA-018	Fatal	Usage {dynamicFieldUsageName} can only have a record when the condition defined evaluates to true
GEN-DYNA-019	Fatal	Usage {dynamicFieldUsageName} should have at least one record
GEN-DYNA-020	Fatal	Key field {flexCodeFieldUsageCode} should have unique value for Dynamic record {dynamicFieldUsageName}
COD-FCFU-101	Fatal	Dynamic Records should specify the value for key field. Dynamic Record {dynamicFieldUsageName} does not specify value for key {flexCodeFieldUsageCode}
GEN-PROC-017	Fatal	Value {value} is not part of domain {domain}

Besides these messages, business rule messages may also occur in the responses of Integration Points. Business rule messages are raised in the validation layer of the OHI application and are common to both the User Interface Pages and the Integration Points. For example, business rule GEN-TMVL-001: "The start date should lie before the end date for {dynamicFieldUsageName}".

Lastly, technical error messages may be returned through the responses of Integration Points. For example, database message GEN-ORA-01400: "NAME" column is mandatory for table "REL_PROVIDERS".

4.4 Data File Set Integration Point

OHI Claims application supports file based data import. Uploading the files and processing the file contents is a two step process:

1. Load the files using the Data File Set Integration Point.
2. Process the file contents by initiating the proper activity type using the Activity Integration Point or through the UI.

The data file set integration point allows uploading of the files in the following ways:

- Through multiple request -response based conversation mode.
- Through a single request by creating the data file set and loading multiple data files.

4.4.1 Creating a data file set with one or more files in conversation mode (multiple requests).

In this scenario the client performs the following steps:

- Create data file set, optionally with one or more files.
- Add data file to the data file set, if not already created with the data file set.
- Add file content.

4.4.1.1 Step 1: Create Data File Set

This request enables an external system to create a data file set.

4.4.1.1.1 **Request Message**

The create data file set request will have the following structure:

```
<dataFileSet code='' description=''>
</dataFileSet>
```

4.4.1.1.2 **Response Message**

The create data file set success response will have the following structure:

```
<dataFileSet code=''>
  <links>
    <link rel='self' type='application/xml' href='/datafilesets/
{datafilesetcode}'/>
  </links>
</dataFileSet>
```

The URI received in the response could be used to perform further actions on the data file set.

4.4.1.1.3 Step 2: Create data files in a Data File Set.

A data file can be created within a previously created data file set by posting a request to the URI received in the response of "Create Data File Set request" (uri='/datafilesets/{datafilesetcode}).

4.4.1.1.4 **Request Message**

The create file within a data file set request will have the following structure:

```
<dataFile
```

```
    code=''
    description=''
    filePath='' -- File path can be supplied for reference purposes. The
   file content is uploaded through a POST operation
  />
```

4.4.1.1.5 **Response Message**

The create file within a data file set success response will have the following structure:

```
  <dataFileSet code=''>
    <links>
      <link rel='self' type='application/xml' href='/datafilesets/
  {datafilesetcode}'/>
    </links>
    <dataFiles>
      <dataFile code=''>
        <links>
          <link rel='file' type='text/xml' href='/datafilesets/
  {datafilesetcode}/datafiles/{datafilecode}/data'/>
        </links>
      </dataFile>
    </dataFiles>
  </dataFileSet>
```

If the code for the data file set or data file is omitted a system generated number will be used instead.

4.4.1.2 Optional: Create data file set with a data file

Data file can be created directly in Step1. This will create a data file set with one data file or multiple data files. In this case the request - response xml for Step1 will be as mentioned below.

4.4.1.2.1 **Request Message**

The create data file set with a data file request will have the following structure:

```
  <dataFileSet code='' description=''>
    <dataFiles>
      <dataFile code='' description='' filePath=''/>
      ...
    </dataFiles>
  </dataFileSet>
```

4.4.1.2.2 **Response Message**

The create data file set with file success response will have the following structure:

```
  <dataFileSet code=''>
    <links>
      <link rel='self' type='application/xml' href='/datafilesets/
  {datafilesetcode}'/>
    </links>
    <dataFiles>
      <dataFile code=''>
        <links>
          <link rel='file' type='text/xml' href='/datafilesets/
  {datafilesetcode}/datafiles/{datafilecode}/data'/>
        </links>
      </dataFile>
      ...
    </dataFiles>
  </dataFileSet>
```

4.4.1.3 Step 3: Add data to Data File

The payload can then be submitted to the URI which was received as the response of Step 2. i.e '/datafilesets/{datafilesetcode}/datafiles/{datafilecode}/data'. The file structure is described in the specification of the file import based integration points.

4.4.1.3.1 **Response Message**

The add content to data file request's success response will have appropriate HTTP status code in the header and the following structure:

```
<dataFileSet code=''>
  <links>
    <link rel='self' type='application/xml' href='/datafilesets/
{datafilesetcode}'/>
  </links>
  <dataFiles>
    <dataFile code=''>
      <links>
        <link rel='file' href='/datafilesets/{datafilesetcode}/datafiles/
{datafilecode}/data'/>
      </links>
    </dataFile>
  </dataFiles>
</dataFileSet>
```

For all of the above mentioned steps , in case of failure the response will be as specified in the standard structure under "Indicating Failure" in Response Messages[1].

Example : A result message for file content that could not be added as the data file code was unknown will have the following structure.

```
<resultMessages result='F'>
    <resultMessage code='DAT-IP-DAFI-005'>
     DAT-IP-DAFI-005:Data file code abc is unknown to data file set pqr.
    </resultMessage>
</resultMessages>
```

4.4.2 Creating a data file set with multiple files in a single request

As an alternative to creating the data file set in the 'conversational' mode, the data file sets endpoint offers the client the ability to create a data file set with one or more data files, together with the file contents, in one request.
This feature requires the client to POST a so called multipart request message that contains the following parameters:

- The "dataFileSetCode" that contains the unique code for the data file set that will be created.
- One or more "file" parameters that contain the file contents along with file metadata.

Multipart requests can be constructed with various technologies. The following is an example in HTML:

In an HTML based UI the following sample form could be used to create a data file set with code as "myfileSet" and upload two files with code "fileCode1" and "fileCode2" in one request:

```
<html>
  <form name="formtest" action="/filesets/multipart" method="POST"
 enctype="multipart/form-data">
    <input type="text" name="dataFileSetCode" value="myfileSet" />
    <input type="file" name="fileCode1" value="" />
    <input type="file" name="fileCode2" value="" />
    <input type="submit" value="Submit" />
  </form>
</html>
```

The system's response to this request will have the following structure:

```
<dataFileSet code='myfileSet'>
  <links>
    <link rel='self' type='application/xml' href='/datafilesets/
myfileSet'/>
  </links>
  <dataFiles>
    <datafile code='fileCode1'>
      <links>
        <link rel='file' href='/datafilesets/myfileSet/datafiles/
fileCode1/data'/>
      </links>
    </datafile>
    <datafile code='fileCode2'>
      <links>
        <link rel='file' href='/datafilesets/myfileSet/datafiles/
fileCode2/data'/>
      </links>
    </datafile>
  </dataFiles>
</dataFileSet>
```

In case of failure the response will be as specified in the standard structure under "Indicating Failure" in Response Messages.[2]

4.4.3 Other Available Operations

4.4.3.1 Get defined data file sets in the system

This request can be used to fetch the details of all the available data file sets in the system. This is a URI based request (/datafilesets) and supports pagination[3].

4.4.3.2 Get details of a data file set

This request can be used to fetch the details of data file sets in the system. This is a URI based request. The URI must have the following pattern: /datafilesets/{datafilesetcode}

The response to this request will have the following structure:

```
<dataFileSet code='' description='' locked='N'>
  <dataFiles>
    <dataFile code ='' description ='' contentLength='' filePath=''>
      <links>
        <link rel='file' href='/datafilesets/{datafilesetcode}/datafiles/
{datafilecode}/data'/>
      </links>
    </datafile>
   ....
  </dataFiles>
</dataFileSet>
```

4.4.3.3 Get details of a data file

This request can be used to download the contents of a data file in the system. This is a URI based request. The URI must have the following pattern: /datafilesets/{datafilesetcode}/datafiles/{datafilecode}/data

4.4.3.4 Update details of a data file set

This request can be used to update the details of a data file set in the system. The URI will have the following pattern: /datafilesets

The Request / Response xml structures are the same as for creating a data file set.

4.4.3.5 Update details of a data file in a data file set

This request can be used to update the details of a data file in the system. The URI will have the following pattern: /datafilesets/{datafilesetcode}/datafiles/

The Request / Response xml structures are the same as for creating a file within data file set.

4.4.3.6 Delete a data file set

This request can be used to delete a data file set in the system. This is a URI based request. The URI will have the following pattern : /datafilesets/{datafilesetcode}/

The response message structure will be as specified in Response Messages.[4]

4.4.3.7 Delete a data file in a data file set

This request can be used to delete the details of a data file in a data file set in the system. URI will have pattern: /datafilesets/{datafilesetcode}/datafiles/{datafilecode}

The response message structure will be as specified in Response Messages.[5]

For details on how attributes in the request messages are handled, refer to the Attribute Handling[6].

4.4.4 Error Messages

The following error messages, that are specific to the data file set integration point may be returned in the response messages:

Scenario	Message code	Message	Severity
Create a data file set	DAT-IP-DAFI-001	Data file set code {0} already exists	Fatal
Create data file within a data file set.	DAT-IP-DAFI-002	Data file code {0} already exist within the data file set	Fatal
Update/Delete/Get a data file set or Update/Delete/ Get file within a data file set.	DAT-IP-DAFI-003	Data file set code {0} is unknown	Fatal
Update/Delete a data file set or Update/Delete file within a data file set.	DAT-IP-DAFI-004	Data file set code {0} is locked for modification	Fatal
Update/Delete/Get data file within a data file set.	DAT-IP-DAFI-005	Data file code {0} is unknown to data file set {1}	Fatal

4.5 Activity Integration Point

The activity integration point provides the following functions:

- Create an activity with the intention to start it later.
- Create and immediately start an activity.
- Recover an activity.
- Monitor status of an activity.

Creating and Starting an Activity

This request enables an external system to create and start an activity. The URI that is used determines how the system processes the request. Use URI:

- /activities to create an activity but not execute it yet.
- /activities/start to create an activity and immediately execute it.
- /activities/{activity_id}/start to start an activity that was created earlier.

Request Message

The start activity request will have the following structure:

```
<activity
  level=""
  code=""
  description=""
  parameterSetCode="">
    <contextFields>
      <contextField
        name=""
        value=""/>
    </contextFields>
    <parameters>
      <parameter
        name=""
        value=""/>
   </parameters>
</activity>
```

Context Field

Some activity types can only be started in a specified context. This information is provided via the <contextFields> element. The attribute "name" of element <contextField> must point to the context and the attribute "value" must contain the logical key for the context.

Example: activity type SUPERSEDE_REVERSE that is defined at level TS (or TransactionSet) can only be started in context of a financial transaction set. The financial transaction set code must be provided as value to set-up the execution context.

Response Message

If the activity was successfully created but not started (i.e. URI /activities was used) then the response will have the following structure:

```
<activity level="" status="">
  <links>
    <link rel='action/startprocessing' type='application/xml' uri='/
activities/{activityId}/start'/>
  </links>
</activity>
```

Note that the URI in the link can be used by the client to start the activity. Attempt to start an activity that is not in a valid state i.e. already completed activity, or an activity that is currently being processed will have a similar response as that of get activity status.

If the activity was created and executed immediately (i.e. URI /activities/start was used) then the response will have the following structure:

```
<activity level="" status="">
  <links>
    <link rel='self' type='application/xml' uri='/activities/
{activityId}'/>
  </links>
</activity>
```

In case the activity could not be registered in the system, for example because the activity code is unknown, the response will be as specified under "Indicating Failure" topic in Response Messages[7].

Recover an Activity

This request enables an external system to recover a failed activity. This is a URI based request with the following pattern: /activities/{activityid}/recover.

Only activities that failed in status TE/CT can be recovered. Examples of such a failure could be the inability of sending out financial messages or errors in dynamic logic or any other technical error. Attempt to recover an activity that is not in valid state will have a similar response as that of get activity status.

The recover request will have a similar response as that of a start activity.

Status Monitoring

Step 1: Get activity status

This feature provides the ability to fetch the status of an activity. This is a URI based request with the following pattern: /activities/{activityid}.

Response Message

```
<activity
 level=""
 status="">
    <links>
      <link rel='messages' href='/activities/{activityid}/messages'/>
   </links>
</activity>
```

The <resultMessages> element will only be available when the activity has concluded with errors. If the activity has concluded with errors and it is desirable to have the details of the errors then Step 2 must be performed.

Step 2: Get activity result messages

This feature provides the ability to get the result messages of an activity. The URI received from the step 1 should be used i.e /activities/{activityid}/messages. This supports pagination[8].

Response Message

The response will have the following structure:

```
<activityMessages>
  <resultMessages result="" elementId="">
     <resultMessage
       code =""
      >
      messageText
    </resultMessage>
  </resultMessages>
</activityMessages>
```

Element and Attribute

- <resultMessage elementId> its an optional parameter. It provides consolidation of the messages based on its value. The elementId attribute usage for an activity is defined in the description of the activity type. Example: For the provider import activity the element id can be provider code in combination with the flex field code for the provider resource.

Optional: Activity Notification

The activity integration point provides a call back feature to a pre-configured endpoint. This feature provides the following response to the configurable endpoint once the activity triggered by an external system through IP has concluded. The generic notification endpoint can be configured as property 'ohi.activityprocessing.notification.endpoint' in the OHI Components application's properties file. The notification endpoint can be overridden for specific activity types, e.g. specify property 'ohi.activityprocessing.notification.endpoint.SELECT_TRANSACTIONS_IN_SET' to have the system deliver all notifications for activity type SELECT_TRANSACTIONS_IN_SET to a specific endpoint.

Response Message

The response message once the activity has concluded will have the following structure:

```
<activity level="" status="">
   <links>
     <link rel='messages' href='/activities/{activityid}/messages'/>
   </links>
</activity>
```

Note: the response may contain a <dataFileSets> element. Refer to Special parameters for details.

In case of failure to retrieve the status of activity (Example: activity code is unknown) the response will be as specified under "Indicating Failure" topic in Response Messages[9].

4.5.1 Conversation Parameter

The responseDataFileSetCode parameter influences the standard request-response mechanism for an activity, it is therefor referred to as a conversation parameter.

As is the case with any activity parameters, conversation parameters can only be used if they are defined for the activity type. The following is an example of a start activity request that makes use of the responseDataFileSetCode parameter:

```
<activity level="" code="">
  <parameters>
    <parameter name="responseDataFileSetCode" value="RESPONSE_DFS"/>
  </parameters>
</activity>
```

Assuming that the parameters and values are valid, the request in this example creates an activity. As part of the execution a response data file set with code "RESPONSE_DFS" is created. The response to the status monitoring requests i.e. get activity status and the call back response (<resultDataFileset > element will be added to the above response) will have the following structure:

Response Message

```
<activity level="" status="">
   <links>
     <link rel='messages' href='/activities/{activityid}/messages'/>
     <link rel='file' href='datafilesets/{datafilesetcode}/datafile/
{datafilecode}/data'/>
   </links>
</activity>
```

The response file can be downloaded by using "Get details of a data file" request from data file integration point : URI : datafilesets/{datafilesetcode}/datafile/{datafilecode}/data should be use to initiate a request to download the response file.

The response file will have the following structure:

```
<{rootElelement}>
  <resultMessages result="" elementId="">
     <resultMessage
       code =""
      >
      Text Message
    </resultMessage>
  </resultMessages>
</{rootElelement}>
```

The {rootElelement} for the response file is described in the activity type.

4.5.1.1 Error Messages

The following error messages, that are specific to the activity integration point may be returned in the response messages:

Message code	Scenario	Message	Severity
ACT-IP-ACTY-001	Create or Create & Start activity	Activity code {0} is unknown	Fatal
ACT-IP-ACTY-002	Create or Create & Start activity	Activity type level {0} is unknown	Fatal
ACT-IP-ACTY-003	Create or Create & Start activity	Parameter set code {0} is unknown	Fatal
ACT-IP-ACTY-004	Status monitoring and get result messages	Activity Id {0} is unknown	Fatal
ACT-IP-ACTY-005	Create or Create & Start activity	Use either a parameter set or parameters but not both	Fatal

4.5.1.2 Examples

4.5.1.2.1 Create or Start Activity request - SELECT_TRANSACTIONS_IN_SET

```
<activity
  level="GL"
  code="SELECT_TRANSACTIONS_IN_SET"
  description="Select processed transaction in a new set">
    <parameters>
     <parameter   name="financialTransactionSetCode"
 value="FINANCIALTRANSACTIONSRUN_20141107"/>
     <parameter   name="financialTransactionSetDescr" value="Financial
 transactions set for Nov-07, 2014"/>
     <parameter   name="transactionCreatedDateFrom" value="2010-01-01"/>

     <parameter   name="transactionCreatedTimeFrom" value="0800"/>
     <parameter   name="transactionCreatedDateTo" value="2014-11-06"/>
     <parameter   name="transactionCreatedTimeTo" value=""/>
     <parameter   name="paymentDueDateFrom" value="2010-01-01"/>
     <parameter   name="paymentDueDateTo" value="2014-11-14"/>
     <parameter   name="includeUnfinalized" value="Y"/>
     .....
   </parameters>
</activity>
```

4.5.1.2.2 Create or Start Activity request - SUPERSEDE

```
<activity
  level="TS"
  code="SUPERSEDE"
  description="Supersede applicable transaction in the set
 FINANCIALTRANSACTIONSRUN_20141107">
```

```
        <contextFields>
          <contextField
            name="transactionSet"
            value="FINANCIALTRANSACTIONSRUN_20141107"/>
        <contextFields>
    </activity>
```

4.5.1.2.3 Create or Start Activity request - PROVIDER_IMPORT

```
    <activity
      level="GL"
      code="PROVIDER_IMPORT"
      description="Processed data file set 001V11">
       <parameters>
         <parameter   name="dataFileSetCode" value="001V11"/>
         <parameter   name="responseDataFileSetCode" value="response001V11"/
    >
       </parameters>
    </activity>
```

4.6 File Based Import

OHI Components applications support file based data import for the following entities:

- Import Providers and Provider Groups
- Import Relations

Uploading files and processing the file contents is a two step process:

1. Upload the file using Data file set integration Point[10]
2. Process the file contents by initiating the proper activity type using the Activities Integration Point[11].

    - Provider Import activity type to process the file based request for the provider integration point.
    - Relation Import activity type to process the file based request for relation integration point.

Parameters

The parameters for the file import based activity are as follows:

- Data File Set Code
  This parameter indicates the data file set that needs to be processed.

The activity can be created only when the Indicator Locked on the data file set code is 'N' otherwise an error ACT-VL-FIAT-001 is raised.

Once the activity picks up a data file set for processing (i.e. starts an created activity), it places a lock on it by updating the indicator Locked to 'Y'. If the indicator Locked is already found to be 'Y' then an error ACT-VL-FIAT-001 is raised. Activity must unlock the data file set in case if the processing completes with the error(s).

- Response Data File Set Code
  The data file set code that should be used for the data file set of the response file. If not provided the system generates a UUID based value as data file set code.

Data File Validation

The first step in every data file import activity is to check whether the datafile that is provided through the parameters is valid for the specific data import. The following checks are executed:

1. Does the datafile exist (i.e. is there a datafile in the datafile set)?
2. Is the datafile empty?
3. Is the datafile of the correct type for the import activity (i.e. does it have the correct root element)?

Error Message

The file import based activity can result in the following messages:

Code	Sev	Message
ACT-VL-FIAT-001	Fatal	Data file set {code} is locked and cannot be picked up for processing.
ACT-VL-DAFI-001	Fatal	No data file exists in datafile set {code}.
ACT-VL-DAFI-002	Fatal	Data file {code} is empty.
ACT-VL-DAFI-003	Fatal	Data file {code} is of incorrect type for this integration point.

4.6.1 Example : Provider Import

4.6.1.1 Step 1: Create data file set with a data file

4.6.1.1.1 **Request Message**

The create data file set with a data file request will have the following structure:

```
<dataFileSet code='ProviderImport_20141210_01' description ="Provider
 Import">
  <dataFile
    code='IndividualProviderFile1'
    descr='Individual Provider File 1'
    filepath='system1/tmp/Providers/1_1212121_v2.xml' -- This can be the
 reference path of the external system. This is for information only.
  />
</dataFileSet>
```

4.6.1.1.2 **Response Message**

```
<dataFileSet code='ProviderImport_20141210_01'>
  <links>
    <link rel='self' type='application/vnd.ohi-xml' uri='/datafilesets/
ProviderImport_20141210_01'/>
  </links>
  <dataFiles>
    <dataFile code='IndividualProviderFile1'>
      <links>
        <link rel='content' type='text/xml' uri='/datafilesets/
ProviderImport_20141210_01/datafiles/IndividualProviderFile1/data'/>
      </links>
    </dataFile>
  </dataFiles>
</dataFileSet>
```

4.6.1.2 Step 2: Upload Content

Use the URI /datafilesets/ProviderImport_20141210_01/datafiles/IndividualProviderFile1/data and upload the file having the structure as specified in Provider Integration point.

4.6.1.2.1 **Response Message**

The add content to data file request's success response will have an appropriate HTTP status code in the header and the following structure:

```
<dataFileSet code='ProviderImport_20141210_01'>
  <links>
    <link rel='self' type='application/vnd.ohi-xml' uri='/datafilesets/
ProviderImport_20141210_01'/>
  </links>
  <dataFiles>
    <dataFile code='IndividualProviderFile1'>
      <links>
        <link rel='content'  uri='/datafilesets/
ProviderImport_20141210_01/datafiles/IndividualProviderFile1/data'/>
      </links>
    </dataFile>
  </dataFiles>
</dataFileSet>
```

4.6.1.3 Step 3: Initiate the Provider Import Activity using Create Activity (/activities)

```
<activity level="GL" code="PROVIDER_IMPORT" description="Processed data
 file set 001.11">
    <parameters>
      <parameter   name="dataFileSetCode"
 value="ProviderImport_20141210_01"/>
      <parameter   name="responseDataFileSetCode"
 value="responseProviderImport_20141210_01"/>
    </parameters>
</activity>
```

4.6.1.3.1 **Response Message**

The start activity response will have an appropriate HTTP status code in the header and the following structure:

```
<activity code='PROVIDER_IMPORT' status ="IN">
  <links>
    <link rel='action/startprocessing' type='application/vnd.ohi-xml'
 uri='/activities/12345/start'/>
  </links>
</activity>
```

4.6.1.4 Step 4: Start Provider Import Activity

Use the URI /activities/12345/start to start the activity.

4.6.1.4.1 **Response Message**

The start activity response will have an appropriate HTTP status code and location in the header and will have the following structure:

```
<activity
 code="PROVIDER_IMPORT"
 status="IP">
</activity>
```

4.6.1.4.2 Step 5: Get Status

Use the URI /activities/12345 to get the status of the activity

4.6.1.4.3 **Response Message**

When the activity has not concluded the response structure will be

```
<activity code="PROVIDER_IMPORT" status="IP" />
```

When the activity has concluded with business errors the response structure will be

```
<activity code="PROVIDER_IMPORT" status="CB">
   <links>
     <link rel='messages' uri='/activities/12345/messages'/>
     <link rel='datafilesets' uri='datafilesets/
responseProviderImport_20141210_01/datafiles/67890/data'/>
   </links>
</activity>
```

4.6.1.5 Step 6: Get result messages

You can get more information about the result message by sending a request to URI /activities/12345/messages

4.6.1.5.1 **Response Message**

```
<activityMessages>
  <resultMessages result="F" elementId="1234 A">
    <resultMessage code ="REL-IP-PROV-031">
      REL-IP-PROV-031: Country code AA is unknown
    </resultMessage>
  </resultMessages>
  <resultMessages result="F" elementId="1235 B">
    <resultMessage code ="REL-IP-PROV-031">
      REL-IP-PROV-031: Country code AA is unknown
    </resultMessage>
  </resultMessages>
</activityMessages>
```

4.6.1.6 Step 7: Get response file

You can download the response file by using "Get details of a data file" request from data file integration point with URI - datafilesets/responseProviderImport_20141210_01/datafiles/67890/data

4.6.1.6.1 **File Details**

```
<individualProvidersResponse>
  <resultMessages result="F" elementId="1234 A">
     <resultMessage code ="REL-IP-PROV-031">
      REL-IP-PROV-031: Country code AA is unknown
     </resultMessage>
  </resultMessages>
  <resultMessages result="F" elementId="1235 A">
     <resultMessage code ="REL-IP-PROV-031">
      REL-IP-PROV-031: Country code AA is unknown
     </resultMessage>
  </resultMessages>
</individualProvidersResponse>
```

4.7 Seed Data

4.7.1 Activity Types

Code	Description	Type Level	Top Level?	UI?	Dynamic Record Definition	Common / Claims
PROVIDER_IMPORT	Imports the providers contained in the data files of the data file set.	GL	Y	Y	FILE_IMPORT	Common
RELATION_IMPORT	Imports the relations contained in the data files of the data file set.	GL	Y	Y	FILE_IMPORT	Common

4.7.2 Flex Code Field Usages

4.7.2.1 Dynamic Record Definition FILE_IMPORT

Name	Field Code	Pick List	Code Def	Mandatory?	Sequence	Display name
dataFileSetCode	C100			Y	1	Data File Set Code
responseDataFileSetCode	C100			N	2	Response Data File Set Code

1. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
2. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
3. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY.html
4. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
5. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
6. http://slcibah.us.oracle.com:8888/OHI-Main/13662-DSY
7. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
8. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY.html
9. http://slcibah.us.oracle.com:8888/OHI-Main/13887-DSY
10. http://slcibah.us.oracle.com:8888/OHI-Main/13891-DSY
11. http://slcibah.us.oracle.com:8888/OHI-Main/13894-DSY

5 Integrating multiple OHI Applications

5.1 Data Replication

Introduction

The focus of this part of the document is synchronization or replication of data between various OHI Components applications.

An example of this is the replication of person data in OHI Enterprise Policy Administration (OHI Policies) to OHI Claims. Both applications make use of the same component or sub system for maintaining that data. Customers could use the Persons HTTP API message based service or the Data File Set interface for maintaining person data in both OHI Policies and OHI Claims.

To prevent customers from being burdened with having to synchronize person data across multiple OHI applications, Oracle provides capabilities for replicating these changes to other (OHI) applications in an automated fashion. In this scenario, OHI Policies acts as the system of record or source system for the person data. Whenever a person or a detail of a person, like an address, is changed in OHI Policies (the source system) it creates a replication event for that member. Target systems, like OHI Claims in this example, subscribe to receive these replication events and use these as a trigger to automatically retrieve the changes.

Entities for which Data Replication between OHI Components applications is supported

The following table provides an overview of the entities for which Data Replication can be enabled and the versions of the OHI Components applications in which this feature was first provided.

Entity	Source System	Target System
Persons	OHI Policies 2.16.1.0.0	OHI Authorizations 2.16.1.0.0
Persons	OHI Policies 2.16.1.0.0	OHI Claims 2.16.1.0.0
Authorizations	OHI Authorizations 2.16.1.0.0	OHI Claims 2.16.1.0.0

Oracle identifies the systems that qualify as source and target systems and which versions of these systems are certified to exchange data using the data replication mechanism.

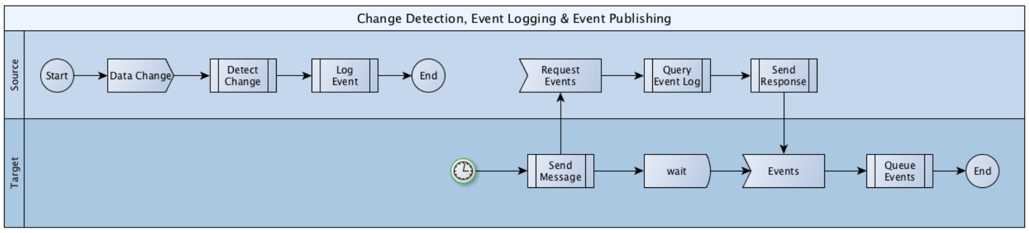
In the following paragraphs the concepts of this data replication mechanism are explained as well as the way to configure eligible OHI Components applications to act as either source or target systems.

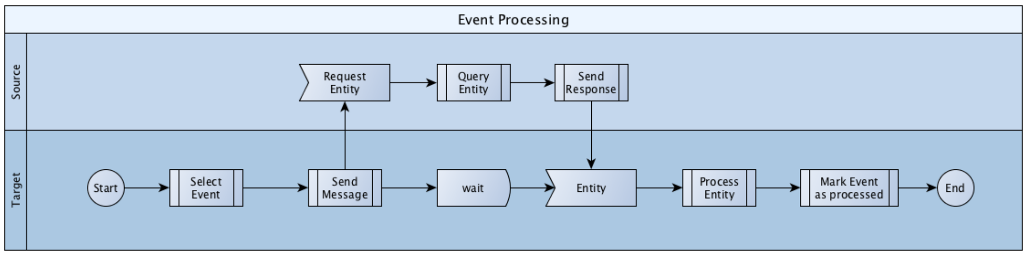
Data Replication Overview and Concepts

This paragraph provides a high-level overview of the data replication mechanism, which can be broken down into the following major areas:

- Change detection: the source system is responsible for detecting changes to entities that need be replicated to target systems. These replication events include insert, update and delete operations.
- Event logging: the source system maintains a log of the replication events that need to be replicated.
- Event retrieval / publishing: events are returned by the source system per request of target systems. For that purpose, the source system exposes an HTTP API resource that target systems can use to retrieve replication events.
- Event processing: the target system processes the events to update its local representation of the entities.

The following pictures provide a high-level overview of the solution, identifying processing in source and target systems:





Change Detection and Change Logging in Source Systems

Changes are tracked at the 'root aggregate' entity level, i.e. for a person and not for a person's address. Changes to details propagate up to the root aggregate. Changes are tracked for each database transaction in the system and for each transaction new replication events are created; replication events that were created in previous transactions are never updated.

The following table lists examples of changes and the resulting data replication events:

Change in the source system in one transaction	Data Replication event(s) created by the system
A new person is added with an address	One Insert event for the person
The last name of an existing person is changed	One Update event for the person
The last name for the same person is changed again	One Update event for the person
For three persons the last names are changed	Three Update events, one for each person
A new address is added for an existing person	One Update event for the person
The name of an existing person is changed and a new address is added for the same person	One Update event for the person
An address is removed from the person's list of addresses	One Update event for the person
A person and all its associated data (like addresses and, marital statuses) is removed	One Delete event for the person
Three persons and their associated data are removed	Three Delete events, one for each person

Configuration options in the Source System

Change detection and logging in a source system is automatically activated when the system is started. It is driven by seeded configuration that is provided by Oracle. Source systems expose an HTTP API resource 'api/replicationevents' that target systems use to retrieve replication events (i.e. the source system does not publish or push replication events). The same resource supports the following operations:

Operation	Description
GET api/replicationevents/entities	Returns an overview of the entities and their configuration details for which the system is able to track changes.
GET api/replicationevents/entities/{entity}	Returns configuration details for a specific business entity.

PUT api/replicationevents/{entity}	To change the configuration for the given entity. It can be used to enable or disable replication event logging for the given entity. Any other elements cannot be changed, attempts to change these will simply be ignored.

For example, to disable change detection and logging for the "persons" entity execute the following:

```
curl -H 'Content-Type: application/xml' -H 'Accept: application/xml' -X
 PUT -d
'<sourceReplicationEventLogConfiguration enabled="N" />'
http://localhost:8001/api/replicationevents/persons
```

To re-enable change detection and logging for the "persons" entity execute the following:

```
curl -H 'Content-Type: application/xml' -H 'Accept: application/xml' -X
 PUT -d
'<sourceReplicationEventLogConfiguration enabled="Y" />'
http://localhost:8001/api/replicationevents/persons
```

Remarks:

- Oracle assumes that this configuration is very static. It may take up to 15 minutes before the system picks up a change to this configuration.
- Once change detection and logging for an entity is disabled the HTTP API resource must be used to enable it again, e.g. if it is disabled restarting the system will not re-enable change detection and logging.

Retrieving Replication Events that were logged by a Source System

A GET operation on the 'api/replicationevents/{entity}/events' resource provides access to the Replication Events that a source system logged for a given entity. It accepts the following parameters:

Parameter	Description
Timestamp (default: null)	Identification of the last replication event entry that was already received by the target system. The source system will return replication event entries with a timestamp that is larger than the given timestamp.
Limit (default: null)	For pagination of replication events.
Timestamp Threshold (default: null)	The same timestamp could occur multiple times (i.e. multiple events were logged at the same time). It might be the case that the target system did not receive all events logged for that timestamp. The Timestamp Threshold informs the source system exactly how many replication events that were logged for the business entity at the given timestamp were already received by the target system. If more replication events were logged for the business entity at the given timestamp the source system will return all these replication events again.

Remarks:

- The source system paginates responses. If the page size limit is not specified in the request it will return a maximum of 1000 replication events (configurable by setting system property 'ohi.ws.replicationevents.pagination.limit'). If the number of events that were logged for the business entity for a specific timestamp exceeds the pagination limit, the latter will be ignored. If additional results are available, the response contains a link that the client can use to get the next set.
- The media type / payload format is 'application/xml'.

- For each business entity, the response contains (absolute) URIs that point to the source system's resource for retrieving the entity, e.g. 'http://host:port/api/persons/123' for retrieving the person that can be identified with id '123' in the source system. The target system will use the URIs to get the updated business entity.

Configuring an OHI Application Target System to retrieve Replication Events from a Source System

An OHI Components application that can retrieve replication events from a source system (or from multiple source systems) comes with the basic setup and processing logic for that purpose. However, by default, retrieving replication events from a source system is not enabled. Customers that want to enable replication of data to a target system must configure the base URI for each entity for which data replication is supported in the target system.

The following table lists the system properties for currently supported data replication use cases:

Entity	Source System	Target System	System Property	Sample Value
Persons	OHI Policies	OHI Authorizations	ohi.ws.sourcesystem .persons.baseurl	http://host:port/ policies
Persons	OHI Policies	OHI Claims	ohi.ws.sourcesystem .persons.baseurl	http://host:port/ policies

Note: the values in the System Property and Sample Value columns is formatted for readability, these should normally be specified as a single string, without line breaks.

An OHI Components application that is preconfigured to retrieve change events from a source system checks for each entity if the associated system property is given a valid URI value. If that is the case, the configuration for retrieving replication events for that specific entity from a source system is enabled.

If the configuration was enabled for at least one entity, the system starts a Data Replication Global activity that frequently runs to collect replication events and make sure that these are processed. The interval for running the Data Replication Global activity is configurable, it can be influenced by setting system property 'ohi.datareplication.activity.runinterval' (value in seconds, by default it is 600).

The Data Replication Global activity executes the following workflow for every entity for which the configuration is enabled:

- If none is running yet, the Data Replication Global activity spawns an instance of the Replication Event Retriever Global activity for each business entity for which data replication is enabled. It is the responsibility of this activity to copy replication events from the source system verbatim, i.e. it does not check for possible duplicates.
  The entity for which this activity instance should retrieve changes is passed to it as input parameter. The Replication Event Retriever Global activity retrieves replication events from the configured source system, using the GET operation on the source system's 'api/replicationevents/{entity}/events' resource.
  It stores the replication events returned by the source system in a local events table from which these are processed. As long as the response message indicates that additional replication events are available the Replication Event Retriever will continue to retrieve and store these.
- If none is running yet, the Data Replication Global activity spawns an instance of the Replication Event Processor Global activity. The Replication Event Processor processes events for a given entity. The processing algorithm is explained in the following paragraph.

Activities and their status can be tracked from the Activities page in OHI Components applications. Instances of the Replication Event Retriever Global Activity and the Replication Event Processor Global activity that completed successfully will be removed periodically if the Data Purge routine (see below) is activated.

Processing Replication Events in a Target System

The source system created new replication events for any change to a specific entity. The Replication Event Retriever Global activity retrieved all these replication events from the configured source system.

An overview of the data attributes in a target system is provided in the following table:

Attribute	Data Type	Required?	Description
Entity	varchar2(30)	Yes	Entity.
Source System	varchar2(30)	Yes	The source system.
Source System Version	varchar2(30)	Yes	Source system version.
Source Subject UUID	varchar2(36,0)	Yes	Unique identifier of the entity as provided in the source system at the time it was created.
Logged Timestamp	timestamp	Yes	Identifies when the replication event was logged in the source system.
Operation	varchar2(1)	Yes	As logged by the source system. Possible values: I; U; D (Insert, Update, Delete respectively).
Retrieved Timestamp	timestamp	Yes	Identifies when the replication event was retrieved.
Business Entity URI	varchar2(200)	No	URI for retrieving the up to date entity from the source system. Empty in case of Delete events.
Activity Id	number(30,0)	No	Reference to the activity that processes the entity.
Processed Timestamp	timestamp	No	Identifies when the replication event was processed.
Processed Result	varchar2(2)	No	Superseded (SU) / Completed (CO) / Errored (ER)

First, the processor checks if dynamic logic functions are available for all replication events for an entity that are not processed yet. If that is not the case then it raises an activity message REP-CTET-001, 'Dynamic Logic function with signature {0} and code {1} not found' and the activity will end with status Completed with Errors. The use case for dynamic logic functions is explained below.

If the activity has validated that dynamic logic functions are available it will process the replication events that were retrieved. The replication events in the target system's database may contain duplicate entries or events that make no sense to even process. An example of the latter is a series of update events for a specific entity that are followed by a delete event; in that case it makes no sense to update the entity in the target system because it is already clear that it will eventually be deleted. Therefore processing replication events in a target system requires consolidation ('rolling up the changes') to prevent replication of the same entity multiple times and to prevent doing unnecessary work.

The algorithm will be explained in terms of the data that OHI Components applications store for replication events that are retrieved from a source system.

The algorithm is as follows:

- First, 'roll up' changes by marking events for an entity as Superseded and set the Processed Timestamp using the following rules:

    - Any events for an entity with the same Source Subject UUID prior to a delete operation for that specific entity are marked as Superseded. Only the delete event for that entity remains to be processed.
    - Any insert or update events except the last one for an entity with the same Source Subject UUID for which there is no delete event are marked as Superseded. Only the last insert or update event for that entity remains to be processed.
- Loop over (remaining) non-processed replication events (for which the Processed Timestamp is null) to process these:
    - Insert a new or update an existing local representation by using the Business Entity URI to retrieve the entity from the source system and calling existing import functions on existing services with the data that was retrieved.
    - In case of a delete operation the entity is queried using the Source Subject UUID. Processing is as follows:
      If the entity could not be resolved using the Source Subject UUID then the processed result in the Target Replication Events entity will be set to Completed.
      If the object could be resolved it will be deleted along with all its details.
      If the entity could be resolved but the delete operation fails, for example because there are references to the business entity in the target system, then a non-fatal message will be added to the activity and the Processed Result will be set to Completed.

If processing of an event results in an error then:

- The processed result for that specific replication event is set to Errored.
- The status of the activity that processed the entity is set to Completed with Errors so that it can be recovered after the problem was investigated and fixed.
- More recent events for an entity with the same Business Entity URI will not be processed.
- Processing will continue for events of the same entity but with a different Business Entity URI. This is similar to importing a file: in that case every element is processed independent from other elements.

Overcoming Model and Setup Differences between OHI Applications

The data replication mechanism can handle differences between entities in various applications:

- Entity models may have application specific differences and the models may evolve over time. Moreover, changes to the entity model may not be applied to all OHI Components applications at the same time. Similarly, customers may uptake new versions of applications at different times.
- The setup may differ from application to application. For example, the person entity in OHI Policies is likely to be extended with dynamic fields that are not comparable to the dynamic fields for a person in OHI Claims. Other examples that may differ for the person entity include (but are not limited to) the code to identify a Dynamic Logic script for formatting a person's name and the access restriction code for viewing a person's (contact) details.

Dynamic fields (and values) that are specified for the entity in the source system will be ignored when importing the entity in the target system. Dynamic field matching is based on the usage name of the dynamic field. The data type of the dynamic field is not taken into account. The following table lists how OHI Applications deal with a number of sample set up scenarios:

Dynamic Field set up in Source System	Dynamic Field set up in Target System	Result when importing entity
Dynamic field with name 'foo' and value 'bar'	Dynamic field with name 'foo' does not exist	Dynamic field is ignored
Dynamic field with name 'foo', type String and value 'bar'	Dynamic field with name 'foo', but of type Date	Import fails

Dynamic field with name 'foo' does not exist	Mandatory dynamic field with name 'foo' exists	Import fails unless dynamic logic is used to populate the field

A dynamic Logic function is used to overcome differences between entities in source and target applications. For each entity, a Data Replication Transformation Dynamic Logic function must be configured for the combination of OHI Components applications (versions) that is certified to exchange data using the data replication mechanism.

The following tables list the Data Replication Transformation Dynamic Logic configuration for OHI Authorizations:

Entity	Source System	Target System	Dynamic Logic Code	Dynamic Logic Signature
Persons	OHI Policies 2.16.1.0.0	OHI Authorizations 2.16.1.0.0	POL216100_AUT216100	Persons Data Replication Transformation

and for OHI Claims:

Entity	Source System	Target System	Dynamic Logic Code	Dynamic Logic Signature
Persons	OHI Policies 2.16.1.0.0	OHI Claims 2.16.1.0.0	POL216100_CLA216100	Persons Data Replication Transformation
Authorizations	OHI Authorizations 2.16.1.0.0	OHI Claims 2.16.1.0.0	AUT216100_CLA216100	Authorizations Data Replication Transformation

For changes to the factory models of entities, Oracle provides Data Replication Transformation Dynamic Logic scripts as sample data. Customers can use these as a starting point and freely adapt these to fit their entity specific configurations.

Dynamic Logic Signature

Data Replication Transformation Dynamic Logic scripts are Dynamic Logic functions with an entity-specific signature named "<Entity> Data Replication Transformation" and the following input parameters:

In / Out	Name	Type	Description
In	xmlEntity	GPathResult	The xml payload representation of the entity that was retrieved from the source system. The GPathResult is constructed using an XmlSlurper. In Groovy, this is the most efficient way for "reading" an xml document.
In	entity	OHI Domain class	The OHI domain class that was constructed from the entity that was retrieved from the source system.

The following sample Dynamic Logic sets the value of dynamic field 'field' that is defined in the target system to the value of the dynamic field 'anotherField' that is defined in the source system:

```
entity.field = xmlEntity.@anotherField
```

Data Purging

Data purging capabilities are available to periodically clean up

- Source Replication Events;
- Target Replication Events;
- Activities that were used to import changed entities in the target system and that completed successfully;
- Activities that were used to import changed entities in the target system, that completed with errors, but that were not associated with any replication events yet (e.g. because it failed fast when not all dynamic logic functions were properly configured).

In-line with existing data purging capabilities, the routine for cleaning up replication events is implemented as a PL/SQL database operation:

```
rep_data_purge_pkg.purge_data
(p_purge_days_source in integer
,p_purge_days_target in integer
);
```

The input parameters define the retention period for the data expressed in days. The minimum retention period is 30 days. A value smaller than 30 for any of the input parameters results in an error.

The Operations Guide explains how to set up frequent and automated purging of technical data.

FIXME

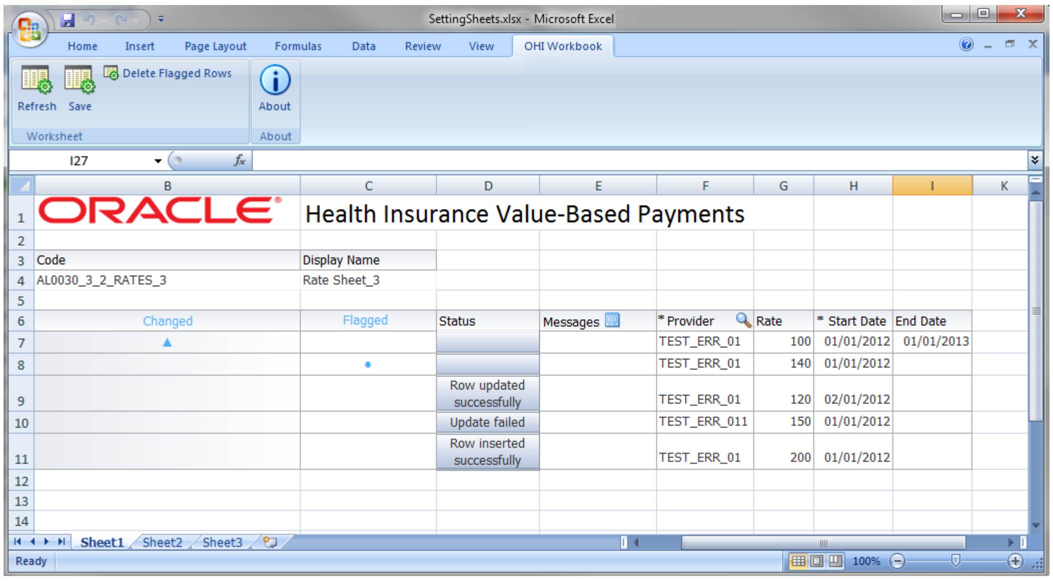
Update page 9683-DSY

6 User Interface Pages

6.1 Desktop Integration

By clicking the Edit in Excel button, it is possible to edit (insert, update or delete) rows in Excel using the ADF Desktop Integration functionality. This functionality is only available on selected user interface pages.

6.1.1 Mock-up



	B	C	D	E	F	G	H	I	K
1	ORACLE	Health Insurance Value-Based Payments							
2									
3	Code	Display Name							
4	AL0030_3_2_RATES_3	Rate Sheet_3							
5									
6	Changed	Flagged	Status	Messages	* Provider	Rate	* Start Date	End Date	
7	▲				TEST_ERR_01	100	01/01/2012	01/01/2013	
8		●			TEST_ERR_01	140	01/01/2012		
9			Row updated successfully		TEST_ERR_01	120	02/01/2012		
10			Update failed		TEST_ERR_011	150	01/01/2012		
11			Row inserted successfully		TEST_ERR_01	200	01/01/2012		
12									
13									
14									

6.1.2 OHI Workbook

The Excel sheet holds a separate OHI Workbook tab in the Excel header area. This tab includes multiple buttons in the following sections:

- Worksheet
- About

6.1.2.1 Worksheet Section

The following buttons are available in this section:

- Refresh
- Save
- Delete Flagged Rows

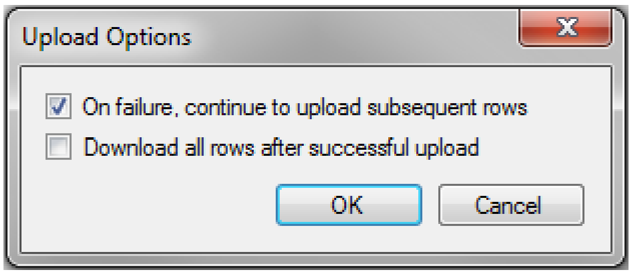
6.1.2.1.1 Refresh

The refresh button cancels all changes entered in the Excel sheet since the last save action (after user confirmation) and gets the current (saved) state from the database, based on certain search criteria (specific for every page).

]

6.1.2.1.2 Save

The save button uploads all rows with a checked Changed field (this field is explained in the Columns chapter). When clicking the save button a dialog is opened with the following options:


Upload Options

[✓] On failure, continue to upload subsequent rows

[ ] Download all rows after successful upload

Continue Upload

This option is checked by default.

- If checked, the subsequent batches of rows will be committed (inserts and updates) if failures occur in a batch of rows
- If unchecked, the subsequent batches of rows will not be committed (inserts and updates) if failures occur in a batch of rows

Download Rows

This option is unchecked by default.

- If checked, all rows that meet the search criteria are downloaded after all rows with a checked Changed field have been successfully committed (inserts and updates)
  - The rows are not downloaded if one or more failures occur
- If unchecked, the rows are not downloaded after all rows with a checked Changed field have been committed (successfully or unsuccessfully)

6.1.2.1.3 Delete Flagged Rows

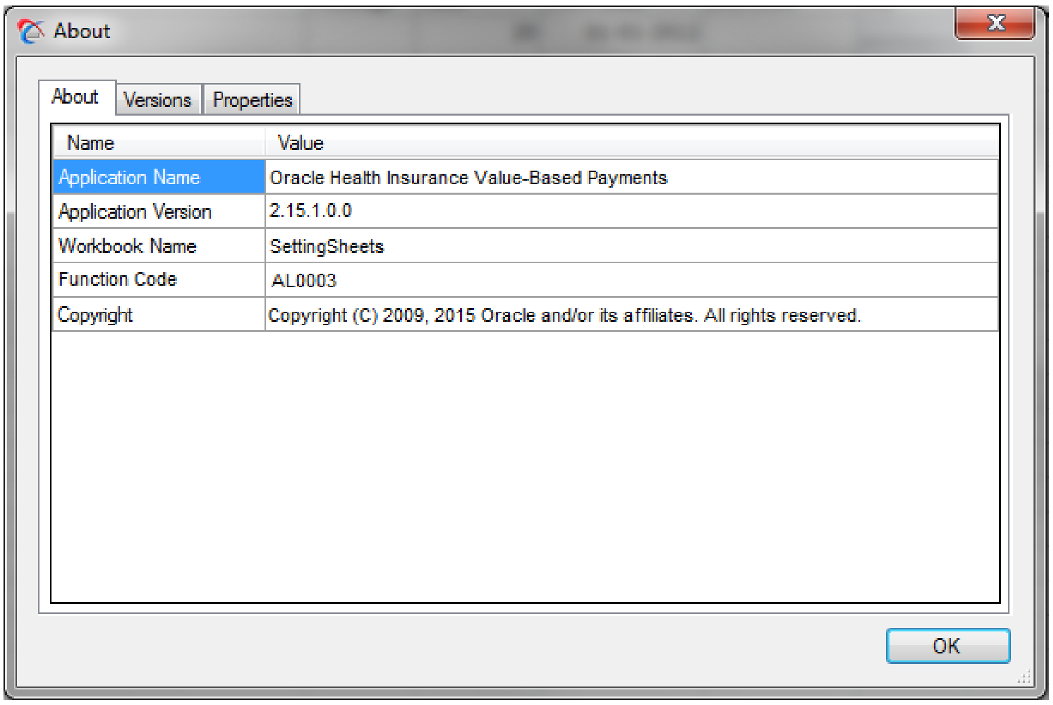
The delete flagged rows button deletes all rows with a checked Flagged field (this field is explained in the Columns chapter).

6.1.2.2 About Section

The following button is available in this section: About.

6.1.2.2.1 About

When invoked, this action launches an About dialog that displays information about the OHI application (name and version) and the specific workbook (name and function code). It also displays technical information about the versions of supporting software and the properties.



Name	Value
Application Name	Oracle Health Insurance Value-Based Payments
Application Version	2.15.1.0.0
Workbook Name	SettingSheets
Function Code	AL0003
Copyright	Copyright (C) 2009, 2015 Oracle and/or its affiliates. All rights reserved.

6.1.3 Columns

The following types of columns are displayed (in specified order):

- Standard Desktop Integration Columns
- Page Specific Fixed Columns
- Page Specific Dynamic Columns

6.1.3.1 Standard

The following columns are displayed (in specified order):

- Changed
- Flagged
- Status
- Messages

6.1.3.1.1 Changed

This field is used to indicate if a row will be committed (insert or update) when the Save button in the OHI Workbook tab is clicked (as described in the OHI Workbook chapter).

- The field is automatically checked:
  - For newly inserted rows
  - For existing rows that are updated (if one or more of the Page Specific fields are updated)
- The field can also be manually checked or unchecked by the user by double-clicking on the field

Note that if an existing row with a checked Changed field is committed, an update is performed on the existing row in the application even if nothing changed on the row.

6.1.3.1.2 Flagged

This field is used to indicate if a row will be deleted when the Delete Flagged Rows button in the OHI Workbook tab is clicked (as described in the OHI Workbook chapter).

- The field is unchecked for newly inserted rows
- The field can be manually checked or unchecked by the user by double-clicking on the field
- Note that the standard Excel method of simply deleting rows from the sheet is not supported

6.1.3.1.3 Status and Messages

These fields are used to show the results of commits after the Save or Delete Flagged Rows button in the OHI Workbook tab is clicked (as described in the OHI Workbook chapter).

After committing, one of the following statuses can be displayed in the field:

- Row inserted successfully
- Row updated successfully
- Insert failed
- Update failed
- Delete failed

Successful Insert

The 'Row inserted successfully' status is displayed when a new row is inserted successfully.

Successful Update

The 'Row updated successfully' status is displayed when an existing row is updated successfully.

Failed Insert

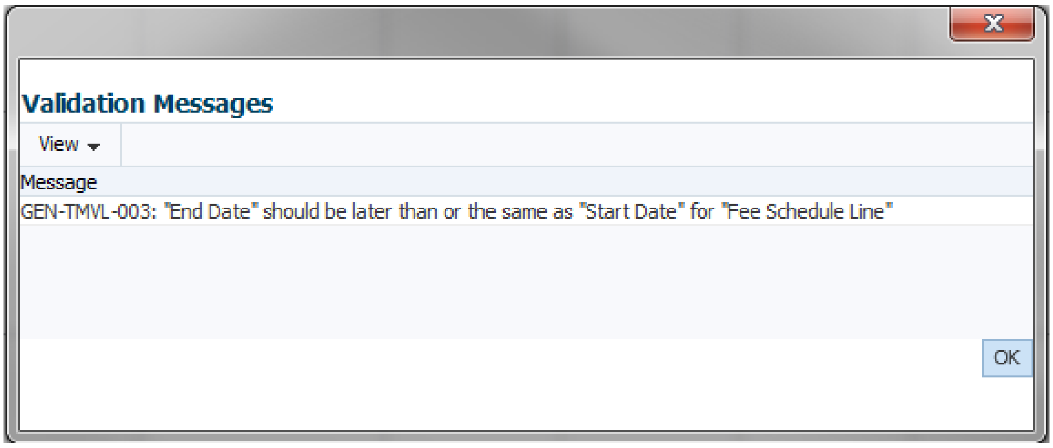
The 'Insert failed' status is displayed when insert of a new row failed, because of errors on that row. Double-clicking on the 'Messages' field opens a Validation Messages dialog where the error messages are displayed (see below). Note that if the insert fails because of errors in other rows in the batch (while on this row there are no errors) the status field is empty.

Failed Update

The 'Update failed' status is displayed when update of an existing row failed, because of errors on that row. Double-clicking on the 'Messages' field opens a Validation Messages dialog where the error messages are displayed (see below). Note that If the update fails because of errors in other rows in the batch (while on this row there are no errors) the status field is empty.

Failed Delete

The 'Delete failed' status is displayed when delete of an existing row failed, because of errors on that row. Double-clicking on the 'Messages' field opens a Validation Messages dialog where the error messages are displayed (see below). Note that if the delete fails because of errors in other rows in the batch (while on this row there are no errors) the status field is empty.



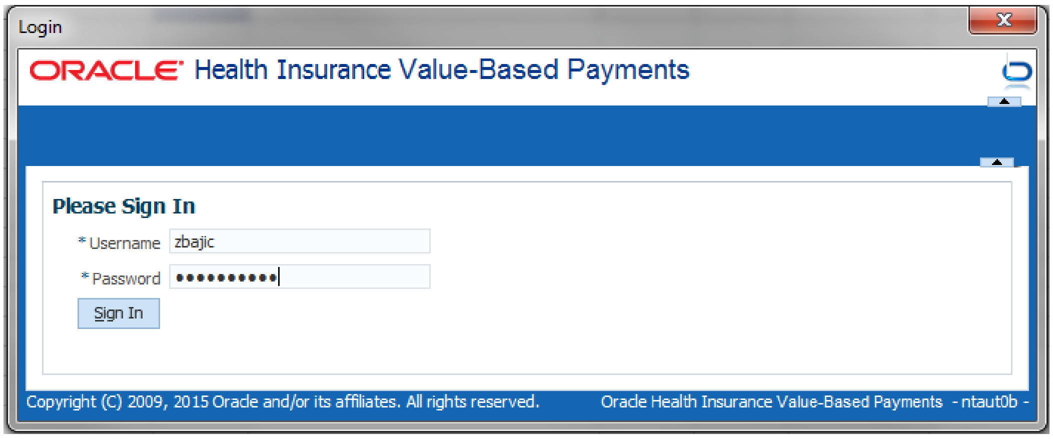
Message
GEN-TMVL-003: "End Date" should be later than or the same as "Start Date" for "Fee Schedule Line"

6.1.3.2 Page Specific

Page specific fixed columns are the fixed fields columns that are specific for every page. Page specific dynamic columns are the dynamic fields columns that are specific for every page. Note that all dynamic fields that are configured for the applicable table (even the ones that are configured to be in the overflow) are displayed as columns.

6.1.4 User Access

When the user clicks on the Edit in Excel button, Excel is opened and a Login dialog is prompted:


Login
ORACLE Health Insurance Value-Based Payments
Please Sign In
Sign In
Copyright (C) 2009, 2015 Oracle and/or its affiliates. All rights reserved.
Oracle Health Insurance Value-Based Payments - ntaut0b -

* Username zbajic
* Password ••••••••••

If the user enters valid credentials and the user has an access restriction grant with retrieve rights for the specific function, the Excel sheet is loaded with all rows that meet the search criteria.

6.1.5 Not Supported

Editing multiple sheets for the same page, at the same time and by the same user is not supported.

6.2 Search Function

This page describes the different types of search that are available in UI pages. It describes the standard behavior, certain pages may have deviations from these standards.

By entering search criteria, the user can restrict the rows visible in a page to only the rows that match these criteria. In general, pages in OHI Claims support two search modes: the quick search and the advanced search.

Both search modes can be used on top level data and child data. Take for example the Countries page. Top level data consists of Countries. For each country, a list of Country Regions is shown in the lower section of the screen. Both the Country section and the Country Region section have their own quick- and advanced search capabilities. Searching for child data is always in the context of a current parent. For example: when US is selected as country in the upper part of the screen, additional search criteria can restrict the Country Regions of country US to those that match the criteria. It is never possible to search children of multiple parents.

6.2.1 Quick Search

This is the default search mode - no additional action is needed to enable the quick search. It can filter on only one field. The list of available filters is limited to fields that are visible as columns.

6.2.2 Advanced Search

Characteristics of Advanced Search:

- It is accessed by clicking on the "advanced search" link
- Can search on all items that are available in the page: both items directly visible and items in an inline overflow.
- Supports entering multiple search criteria at once. Multiple criteria are combined using the AND operator. Only rows that match all entered criteria are shown.
- Performs only a search on items for which a criteria is entered.
- After performing an advanced search, the page will return to quick search mode.

6.2.3 Search Types

Depending on the item type, a different type of search item is shown in the page. Also the search operation performed depends on the item type. See table below.

Item Type	**Search Item Rendered**	**Search Value to be entered**	**Search Operation performed**
Numeric	Input text	Numeric constants	item = search value
Date	Input date (with datapicker)	Date constants	item = search value

Boolean	Drop down with values:
	- Empty
	- Yes
	- No
		Select one of the options	if Yes or No selected:
			item=search value.
			no action otherwise.

Domain Based	Drop Down with domain values and empty value	Select one of the options	if empty selected: no action. otherwise: item = search value.
Normal alphanumeric field. See the first note below	Input text	Alphanumeric constants, including wildcard.	item like search value.
Startdate and Enddate	As of Date field	Date constants	startDate <= search value <= enddate

Information in this table applies to both quick and advanced search.

Searching on an alphanumeric field is case-insensitive, search value ABC will find value Abc or abc.

Though like is the search operation for alphanumeric fields, the user has to enter the wildcard. So search value ABC will not find ABCD. Search value ABC% will.

6.2.4 From/To Search

OHI Claims has an upper limit for the number of rows that can be shown at once. (Currently this is 200). A user might want to see the next set of rows.
Example: the Countries page will show countries up to Suriname (SR). The user wants to see the Countries that alphabetically come after Suriname. To facilitate this, From/To search items are created for the descriptor item of a page. The descriptor item is the item that uniquely identifies the row. For countries this is the code. So for code two search items exist: Code From and Code To. To retrieve the countries from Suriname onwards, the user has to enter SR in Code From and leave Code To empty. From/To search is only available in Advanced Search.