Oracle Health Insurance Back Office

Business Event Framework within

Oracle Health Insurance Back Office

version 2.1

Part number: E51467_01

December 2013

CHANGE HISTORY

Release	Version	Changes
10.13.1.0.0	2.0	New version of the Business Event Framework manual
10.13.3.0.0	2.1	Changes adapted in Dynamic pl/sql definition. New additions in ALG_EVENT_INTERFACE_PCK

Contents

1. Introduction

This document describes the Business Event Framework developed for OHI Back Office.

Specific *hooks* are required in the OHI Back Office application for customers to develop custom event handling using the OHI Back Office database. The Business Event Framework can be used to signal specific events in the OHI Back Office application. These events can arise from creating or modifying data or by the passing of time. The framework is also used to define how an event should be handled.

An example of how the Business Event Framework can be used is when a member supplies the health care payer with their change of address after relocating. The health care payer has an integrated customer relationship management (CRM) system and uses the change of address event to automatically trigger an update to the CRM database.

2. Overview

The purpose of the Business Event Framework is to facilitate custom development for signaling and responding to business events occurring in the OHI Back Office application.

Since the majority of custom development for OHI Back Office implementations is PL/SQL based, the framework is implemented in PL/SQL.

The Business Event Framework provides two options both for signaling and responding. These options can be combined for each business event to create the most suitable environment for handling the event.

2.1. Signaling Events

The Business Event Framework offers two options for signaling events and both are described in this section.

2.1.1. Detected Events

Detected events are events that are signaled by querying the data in one or more tables. A decision to register the event is based on the results of the query. The event is registered based on the data that was found at the moment the data was queried. This moment can be controlled by scheduling Process Business Events (SYS5001S) batch (see Starting Business Event).

For example, a relation record is updated at 08.30, 11.15 and 14.50 hours. When the batch is scheduled to run at 15.00, the data from the last modification (14.50) will be evaluated. The data for the record as at 09.00 or 12.00 cannot be signaled by a detected event.

Detected events are best used in situations where the intermediate modifications are not important or where the passing of time is the trigger for the event.

2.1.2. Triggered Events

Triggered events are signaled the moment they occur. Using database triggers an event is evaluated and registered. Unlike the detected events, intermediate changes can be signaled. In the relation record example, which is updated at 08.30, 11.15 and 14.50, a triggered event can be registered for all three updates.

Events can be signaled separately based how the data is modified, for example insert, update or delete.

Triggered events are used to signal data modifications immediately.

2.2. Responding to Events

The Business Event Framework offers two options for responding to events and both are described in this section.

2.2.1. Batch Response

To process signaled events in a batch the signaled events must be stored in an OHI Back Office table. The moment the event is signaled, either through a detected event or a triggered event, the event is saved to table ALG#EVENTS. The Process Business Events (SYS5001S) batch handles the events. The batch can be scheduled to run at the correct intervals (see Starting Business Event).

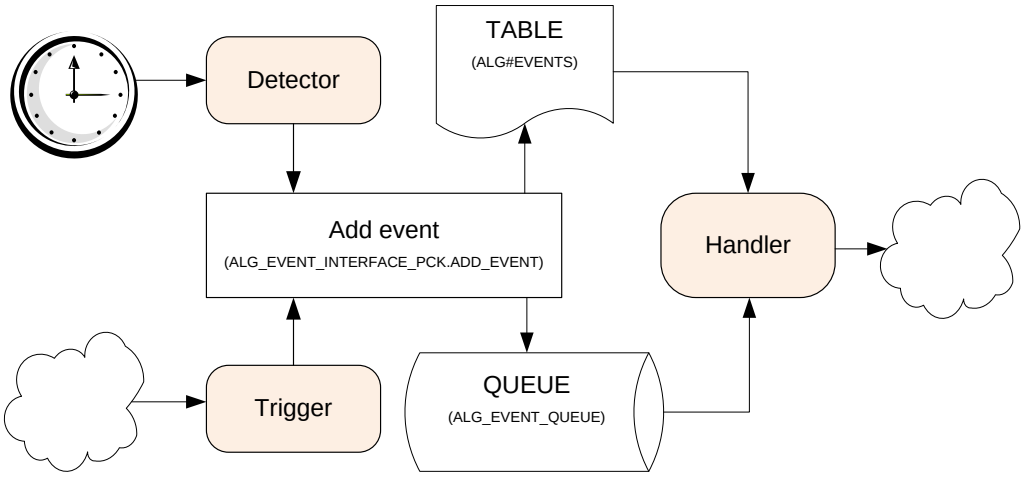
> **Note: No duplicate events**
>
> Duplicate events will not be stored when saving events to a table. In case a relation or policy is signaled multiple times for the same event, the table will hold only one occurrence of the event. If the same data manipulation type is performed twice for an event on a table and record, only the first will result in an event.
>
> After successful processing the event, the same event could be detected again.

2.2.2. Near Real Time

When an event should be signaled the moment it occurs, events can be stored to a queue. An OHI Back Office background process is continuously listening to the queue. Events are taken from the queue and processed immediately.

2.3. Combining Signaling and Response Types

Four definitions result from the two types of events together with two storage options. This section describes the situations where each definition can be used.



2.3.1. Detected Events, Storing to a Table

This event definition is suitable when there is no urgency to act on specific events and individual data changes are not important. For example, there is an event that produces an overview of all policies modified in the previous week. A record of all the individual modifications does not have to be kept. A check on the last date the record was updated is sufficient in this example.

Detected events are also the only events able to act on situations not triggered by data manipulation but the passing of time. For example, a member reaches 18 years of age or a record having a specific status for a number of days. Triggered events are not suitable for this since no data is changed and therefore no database trigger will signal the event.

2.3.2. Triggered Events, Storing to a Table

This event definition is suitable when the action of the event has no urgency but the individual data modifications are important. For example, a triggered event can be used when an event should be registered when a policy reaches the final status. A detected event is less suitable for this because at the time the detection batch is running the policy could have been updated to another status. This results in the policy being skipped by the detection run and no event is registered.

2.3.3. Detected Events, Storing to the Queue

Although technically possible, this type of event is not practical. Detected events are processed in the same batch run. There is not much difference between the moment an event is registered and the moment it is processed. Therefore processing these events using the queue will not provide much of an advantage. The queue will have a large load to process when lots of events are detected.

When multiple occurrences of the same event are required storing to the queue should also be used.

2.3.4. Triggered Events, Storing to the Queue

This event type is best suited when individual updates are important and immediate action is required. For example, the member should receive a welcome email when their policy reaches the final status.

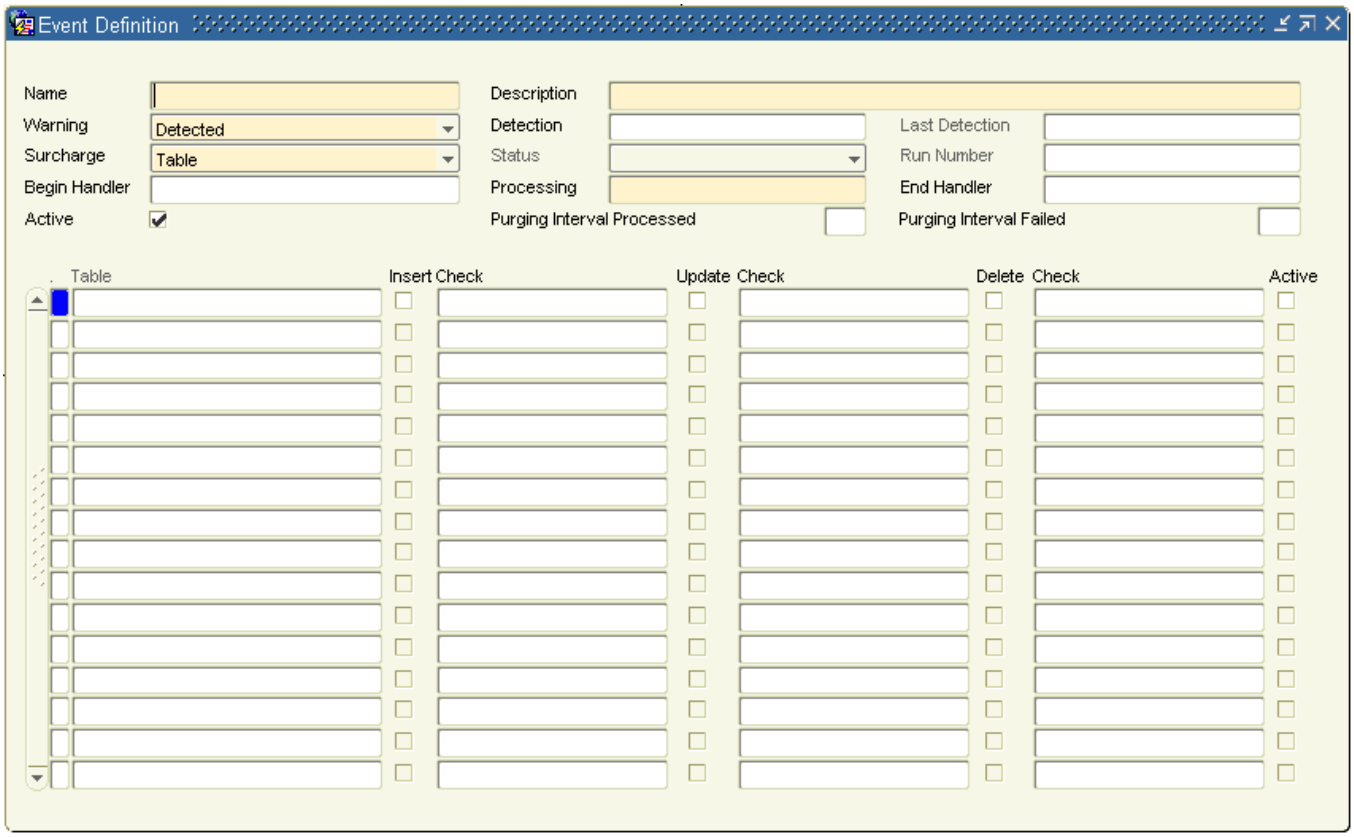
3. Framework Components

This chapter describes all the components within the OHI Back Office application for setup, registering and responding to business events.

3.1. OHI Back Office Windows

3.1.1. Event Definition

The Event Definition (SYS1149F) window is used for defining an event in OHI Back Office.



Data in Event Definition Block

Field	Description
Name	The name of the event, maximum length 30 characters.
Description	The description of the event, maximum length 100 characters.
Type	How the event is signaled, allowable values Detected and Triggered.
Detector	The (package) procedure for registering this event. Only applicable for detected events.
Last Detection	The timestamp of the last processing run. Only applicable for detected events.
Storage	Where are signaled events stored, allowed values Table and Queue.
Status	The status of a processing run. Only applicable for detected events.
Run Number	The last number of the processing run. Only applicable for detected events.
Begin Handler	The (package) procedure for the begin handler. Only applicable for events with storage set to Table.
Handler	The (package) procedure for handling the event. Applicable for all events.
End Handler	The (package) procedure for the end handler. Only applicable for events with storage set to Table.
Active	Indicates whether the event is active or not.
Purge Interval Success	The purge interval for events that have been successfully processed. Only applicable for events with storage set to Table.
Purge Interval Failure	The purge interval for events that have failed. Only applicable for events with storage set to Table.

The event tables block is only applicable for triggered events.

Data in Event Tables Block

Field	Description
Table	Holds the name of the table the event is designed for.
Insert	Indicates whether events should be signaled when a new record in this table is created.
Evaluation function	The name of the dynamic PL/SQL definition used to evaluate the event. Only allowed in case the Insert indication is checked.
Update	Indicates whether events should be signaled when a record in this table is updated.
Evaluation function	The name of the dynamic PL/SQL definition used to evaluate the event. Only allowed in case the Update indication is checked.
Delete	Indication whether events should be signaled when a record in this table is deleted.
Evaluation function	The name of the dynamic PL/SQL definition used to evaluate the event. Only allowed in case the Delete indication is checked.

The event can be fine-tuned with the evaluation functions to only signal the desired situation. See the next section for a more detailed description of these functions.

The window is accessible by following the menu path:

⇒ System

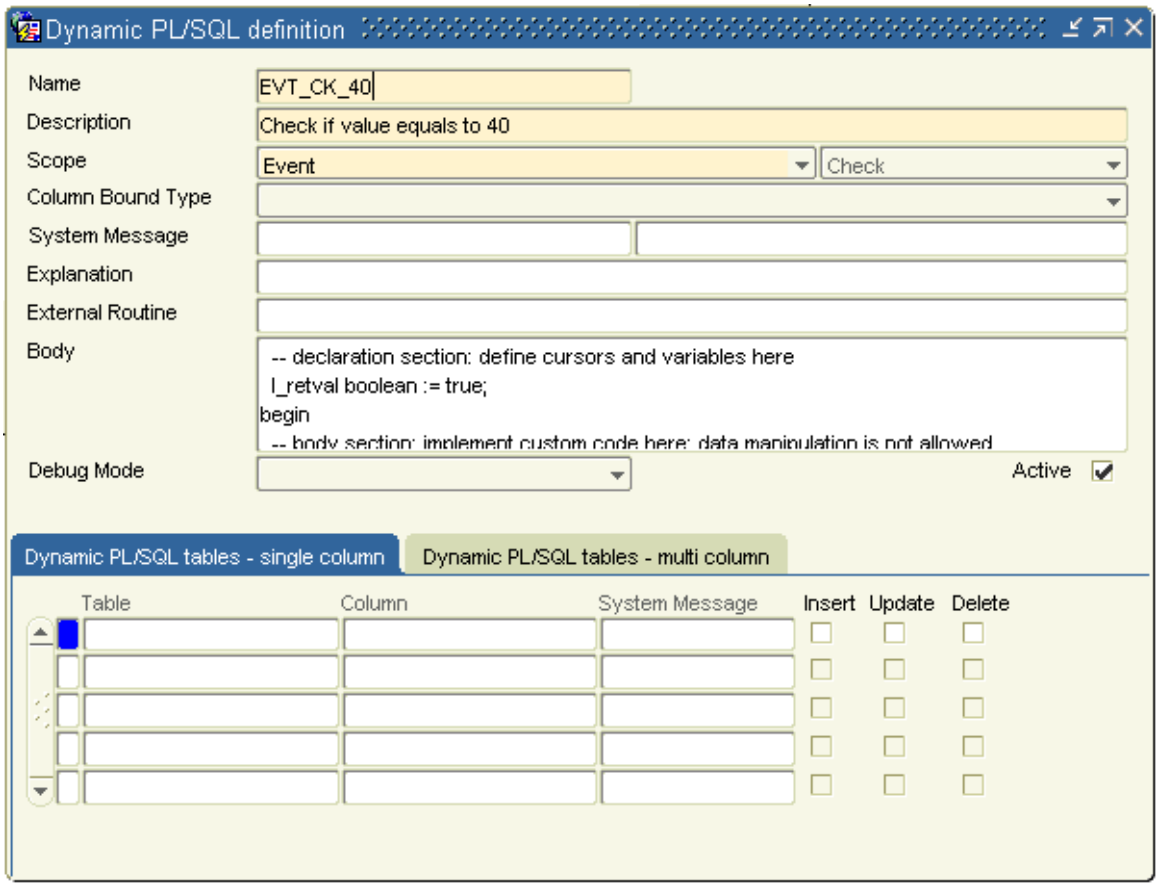
⇒ Management

⇒ General

⇒ Event Definition

3.1.2. Dynamic PL/SQL Definition

The Dynamic PL/SQL Definition (SYS1139F) window is used to maintain the dynamic PL/SQL functions that are used to fine-tune the registering of a specific event.



The Scope must be set to Event for PL/SQL definitions used within the Business Event Framework. This window is also used to maintain PL/SQL definitions used elsewhere within OHI Back Office. Only the fields applicable for the Business Event Framework are described.

Data in the PL/SQL Definition Block

Field	Description
Name	The name of the PL/SQL definition, maximum length 20 characters.
Description	The description of the PL/SQL definition, maximum length 50 characters.
Scope	Should be set to Event to be able to select the definition in the Event Definition (SYS1149F) window.
Body	The actual code of the PL/SQL definition. The event will be registered when the function returns a true value.

The PL/SQL Body contains the actual code used to evaluate whether an event should be registered. The code must return a Boolean value to indicate this. In case true is returned the event will be signaled. In case the function returns false, it will not. Two record variables are available for the old and new values (`old_rec` and `new_rec`). These can be used to evaluate a specific situation, for instance only signal events in case the new status is equal to D.

The two tabs are not applicable for the Business Event Framework.

The window is accessible by following the menu path:

⇒ System

⇒ Management

⇒ General

⇒ Dynamic PLSQL Definition

Note: 'Active' indication

When committing a dynamic PL/SQL definition with the indication 'Active' checked, OHI Back Office will try to validate the code of the Body section. When creating a new PL/SQL definition the table that will be used is unknown to OHI Back Office. Therefore the 'Active' indication should not be checked when first creating the PL/SQL definition. After linking it to a table in the Event Definition (SYS1149F) window it can be turned on.

3.2. Event Definition Package

The `ALG_EVENT_INTERFACE_PCK` can be used to define event definitions. This offers the same functionality as the OHI Back Office Event Definition window with the exception of defining the tables for a Triggered event. The functionality for installing and de-installing an event is available for backward compatibility.

The package also holds procedures that are used for event handling and several utilities.

See Appendix 1 for a full description of the parameters for each procedure and function in the package.

3.2.1. Event Definition

- `install`
  Available as a procedure and a function returning the ID of the event. This can be used for the event definition. When the given event already exists (based on the name of the event) it will update the event definition, otherwise a new event definition will be registered with the values supplied.
- `deinstall`
  This procedure is available twice. Once to remove an event with a given name and once to remove it based on the ID of the event definition.

3.2.2. Event Handling

- `add_event`
  Three procedures are available to store an event to the table. One receives the name of the event as a parameter, the second the ID of the event definition. The parameter code holds the identification of the record in OHI Back Office that caused the event.
  The third procedure stores an event to the Business Event Framework queue. It receives one parameter of type:

`alg_ede_payload_tp`

To be able to change the storage clause of an event from table to queue the code should be a string with the following format:

table_id##record_id##dml_type, where dml_type can be 'I'(Insert), 'U' (Update) or 'D' (Delete)

- `purge_all_events`
  Available twice, based on the name of the event definition and based on the ID of the event definition. It will remove all events and event errors for the given event.
- reapply_failed_event
  Available twice, based on the name of the event definition and based on the ID of the event definition. It will change the status of a event stored in the table from 'Failed' to 'New'. This procedure should be called from within the detector plugin. Providing a specific event will reset only the provided event for the given event definition. When no event is provided all failed events for the given definition will be reset.

3.2.3. Utility

The package has two utilities.

- `type_payload_to_code`
  Can be used to transform object type `alg_ede_payload_tp` to the code parameter of the `add_event` procedure.
- `code_payload_to_type`
  Available twice, used to convert the code parameter of the `add_event` procedure to object type `alg_ede_payload_tp`. Available with the name and the ID of the event definition.

3.3. Event Handling Package

Events are handled by the framework package `ALG_EVENT_PCK`. This is an internal OHI Back Office package and is therefore not available for custom development. It contains the same functions and procedures as the `ALG_EVENT_INTERFACE_PCK`.

3.4. Process Business Event Batch

The Process Business Events (SYS5001S) batch has been developed to support starting the Business Event Framework by the OHI Back Office batch scheduler. The batch is needed to signal Detected events and to process events which are stored in the `ALG#EVENTS` table. The batch can be scheduled using OHI Back Office Submit Batch Request (SYSS003F) window. It has the name of the event as a parameter allowing for different run intervals per defined event.

3.5. Background Process

Background process `OHI_EVENT_JOB_x` is used to handle events with storage set to queue. The process is started and stopped simultaneously with the OHI Back Office batch process.

The process monitors the Business Event Framework queue. Events are taken from the queue and processed using the `ALG_EVENT_PCK` package.

With the Back Office parameter 'No. of processes for event framework' the number of processes listing to the event queue can be set.

3.6. Business Event Framework Tables

The following database tables and object types are used for handling events:

- ALG_EVENT_DEFINITIES (event definitions)
- ALG_EVENT_INIT_WIJZIGINGEN (table involved in a triggered event definition)
- ALG#EVENTS (events)
- ALG#EVENT_ERRORS (error messages)
- ALG_EDE_PAYLOAD_TP (event payload for queues)

The table columns are described in Appendix 2.

4. Building Your Own Business Events

First the business event should be analyzed to determine the best suited registering and handling types. Triggered events are best suited when the event signals data manipulation and it is important to signal each individual action. Detected events can be used for end-of-day status reports or for events not caused by data changes but by the passing of time.

The storage of the event should be set to Queue when, as soon as the event is signaled, immediate action is required. It can be set to Table when the action to the event is less urgent and can occur at a scheduled times.

4.1. Detected Events

In the Event Definition window set the Type to Detected. The Detector field is mandatory for this type of event.

4.1.1. Detector

The field holds the (package) procedure, which is used to register the business event. The procedure receives the timestamp of the last time it was started and the name of the business event. The Business Event Framework will commit after executing the detector.

For example where an event should count the number of policies, the detector in the event definition could be:

```
my_event_pck.detect_nr_policies
```

The procedure definition could look like:

```
procedure detect_nr_policies
( pi_event_name   in alg_event_definities.naam%type
, pi_start_date   in date
);
```

Each event occurrence can be stored using the `add_event` procedure in the `ALG_EVENT_INTERFACE_PCK` package.

4.1.2. Adding Events

Detected events should either be saved to the ALG#EVENTS table or to the Business Event Framework queue. This can be done by calling the `add_event` procedure in the `ALG_EVENT_INTERFACE_PCK` package.

Dependent on the storage clause for the event the appropriate `add_event` can be called. For events stored in the table this would be:

```
alg_event_interface_pck.add_event
( pi_name    in alg_event_definities.naam%type
, pi_code    in alg#events.code
, pi_date    in alg#events.master_date%type
);
```

Or:

```
alg_event_interface_pck.add_event
( pi_ede_id  in alg_event_definities.id%type
, pi_code    in alg#events.code
, pi_date    in alg#events.master_date%type
);
```

For events stored in the queue this is:

```
alg_event_interface_pck.add_event
( pi_ede_payload  in alg_ede_payload_tp
);
```

If the storage type of an event is modified in the Event Definition (SYS1149F) window the `add_event` will continue to work and the received parameters will be converted to match the storage type. Although it can have a (minor) impact on performance it is not necessary to change the detector-program code.

4.1.3. Example

The following code shows an example of an event to signal all new relations created since the last time this event was processed.

```
procedure my_detector
( pi_event_name  in alg_event_definities.naam%type
, pi_start_date  in date
) is
  cursor c_events
  ( vi_date_from  date
  ) is
  select rel.id
  from rbh_relaties  rel
  where rel.creatie_moment >= c_events.vi_date_from;

  l_tab_id     alg_tabellen.id%type;
  l_dml_type   varchar2(1) := 'I';

begin

  -- Determine table id
  l_tab_id := rbh_rel_capi.g_tab_id;

  for r_rec in c_events ( pi_start_date )
  loop

    -- Store to a table
    alg_event_interface_pck.add_event
    ( pi_name   => pi_event_name
    , pi_code   => r_rec.id
    );

  end loop;

end my_detector;
```

4.2. Triggered Events

In the Event Definition window set the Type to Triggered. For events of this type the Detector field is not available since the event is signaled using OHI Back Office database triggers.

4.2.1. Tables

The second block is only available for triggered events. The table of the event can be defined and the action on the table can be set using the Insert, Update or Delete indications.

4.2.2. Fine Tuning

Evaluation functions are available for defining additional criteria for registering an event. These functions can be set up in the Dynamic PL/SQL Definition window.

4.2.3. Example

A triggered event can be set up to signal all policies that reach a final status. Since policies cannot be created with the final status, the only action to monitor is update. To prevent registering other updates to the policy the following dynamic PL/SQL can be created, the scope of the dynamic PL/SQL should be set to Event. The body can hold:

```
  -- declaration section: define cursors and variables here
  l_retval boolean := true;
begin
  -- body section:
  -- return boolean value --
  return new_rec.status  = 'D' and
         new_rec.status <> old_rec.status;
end;
```

Since this PL/SQL definition will be linked to the `VER_POLISSEN` table in the event definition window the `new_rec` and `old_rec` variables will hold all fields available in that table.

4.3. Batch Handled Events

Events with storage clause set to table will be handled by the Process Business Events batch.

4.3.1. Begin Handler

The (package) procedure defined for the begin handler in the event definition is called once. This can be used for example to open a file for writing log messages. The framework will commit after executing the begin handler.

The (package) procedure receives the following parameters:

- The name of the event

- The run number of the process
- The date of the last processed run

```
my_event_pck.begin_handler
( pi_name           in varchar2
, pi_run_nr         in number
, pi_date_detection in date
);
```

4.3.2. Handler

The handler is called for each instance of the event. The framework will commit after executing the handler. The handler (package) procedure receives the following parameters:

- The code of the event
- The date that was passed when registering the event
- The date the event was registered

```
my_event_pck.handler
( pi_name           in varchar2
, pi_date_source    in date
, pi_date_detection in date
);
```

4.3.3. End Handler

The end handler is called once after processing all events. For example this can be used to save information about the process run such as the total number of events processed, the number of failed events or close the file opened in the begin handler. The framework will commit after executing the end handler.

```
my_event_pck.end_handler
( pi_name           in varchar2
, pi_run_nr         in number
, pi_date_detection in date
);
```

4.3.4. Purge Intervals

Removal of old event records is a batch function. It is possible to set up the intervals in the Event Definition window. It is possible to have different values for failed events since investigation may take longer than successful events.

The batch will remove records from the `ALG#EVENTS` tables at the end of the run.

4.4. Near Real Time Events

Events stored to a queue are processed by a continuous Background Process. Since each event is processed individually no begin handler or end handler is available for these events. Only the handler is applicable.

4.4.1. Handler

The OHI Back Office event package will take an event from the queue and call the handler defined in Event Definition window. The (package) procedure for this handler receives an object as parameter. This object contains the following information.

- The ID of the event definition
- The ID of the table the signaled record is stored in
- The record ID
- The DML type that caused the event

The handler can be defined as:

```
my_event_pck.queued_event_handler
( pi_load  in alg_ede_payload_tp
);
```

4.5. Custom Plug-ins

The event tables and event framework are pre-installed in the OHI Back Office database. The custom plug-ins for the detector and handlers of the events must be implemented in a separate database schema.

The following is assumed for the purpose of this installation procedure:

- The event definition is called *my_event*
- The custom components are combined in a single package called *my_event_pck*
- The *my_event_pck.install* procedure creates and configures an event definition for *my_event*
- The OHI components are owned by database schema *ozg_owner*
- The database schema for bespoke software is called *my_schema*
- The business event framework is started by the *ozg_batch* schema.

The installation consists of the following steps:

- Ensure that public synonyms and access privileges are created for the *ozg_owner* components that are accessed by the *my_event_pck* package (you may only use the objects granted by $OZG_ADMIN/OZG_DIRECT.grt.<sid> sqlplus script for this)
- Ensure that *my_schema* has execute privileges for *ozg_owner.alg_event_interface_pck* (should be taken care of in the previous step but in previous releases the grant was missing)
- Compile the package specification and package body for *my_event_pck* under the database schema *my_schema*
- Create a public synonym *my_event_interface_pck* for *my_schema.my_event_pck*
- Grant execute privileges for *my_schema.my_event_pck* to the *ozg_owner* schema.
- Run *my_event_interface_pck.install* under *my_schema* to install the definition for *my_event* or set up the event definition using the Event Definition window.

4.6. Starting Business Events

Starting business events is dependent of the business event definition. Detected events are registered by the Process Business Events batch. Triggered events are started by database triggers.

4.6.1. Process Business Events Batch

The Process Business Events (SYS5001S) batch has been developed to start up a business event processing run. The batch can be scheduled using the Submit Batch Request (SYSS003F) window.

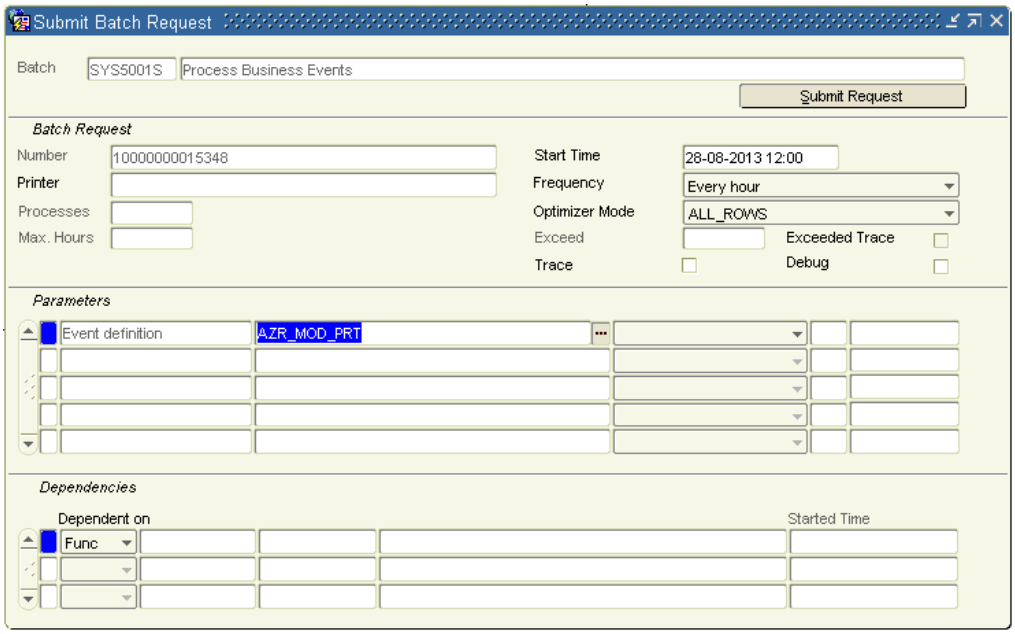


Figure 1 - Sample of scheduling Process Business Events (SYS5001S) batch

In the screenshot business event 'AZR_MOD_PRT' will start every hour.

The batch serves two purposes and is only needed for these types of events:

1. Signal Detected events.
   For detected events the program code defined by the Detector is executed once. See Detector for a more detailed description of the detector.

   This step is skipped for triggered events.

2. Process events stored in a table.
   Events stored in the table are processed. First the specified Begin Handler is called once. Per event the Handler is called to process the event. After processing all events the End Handler is called once.

   After the end handler the batch purges old events.

   This step is skipped for events stored in the queue.

Note: Detection and Processing in one run

For Detected events storing the events to a table and registering and processing the events happen in the same processing run.

4.6.2. Queued Events

Events stored to the queue are processed by a dedicated process monitoring the business event queue. Events are taken from the queue and the handler is called to process the event.

The dedicated process is started and stopped together with the OHI Back Office batch scheduler.

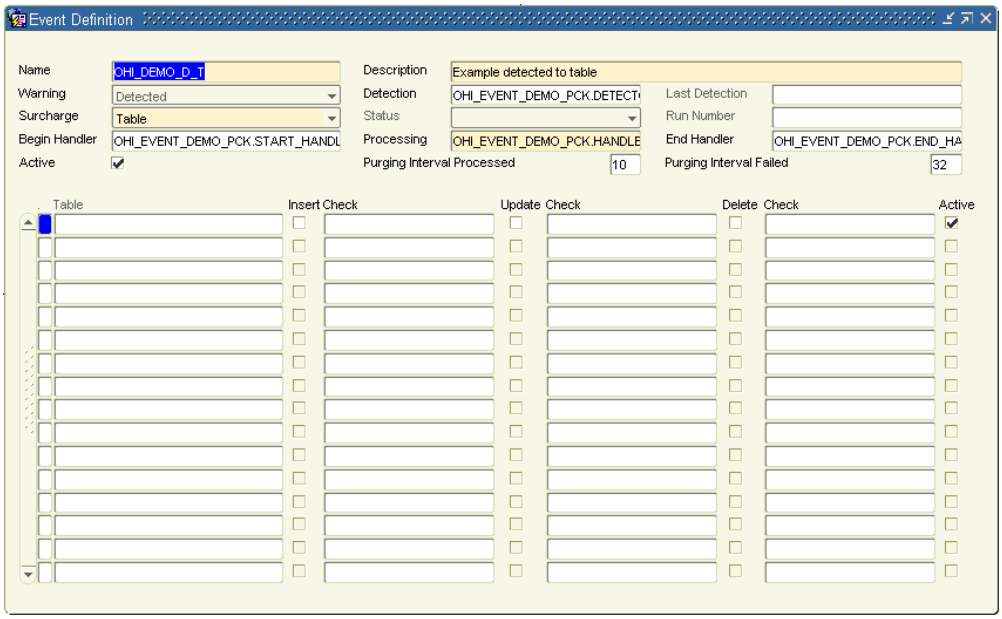
5. Examples

This chapter contains examples of how to set up events to be handled by the Business Event Framework.

5.1. Detected Event, Store to a Table

This example shows an event to signal all relations that have been updated since the last time the event was run. It writes the identification of the relations to a file. Since there is no need to act on individual updates and no immediate action is required upon the change, a detected event storing to a table will suffice.

5.1.1. Event Definition



The screenshot shows an example event definition named OHI_DEMO_D_T. It is a detected event (Type is set to Detected) and it will store events to the `ALG#EVENTS` table (Storage set to Table).

5.1.2. Detector

The detector of the event in this example is `OHI_EVENT_DEMO_PCK.DETECTOR_T`. Typically the code of a detector consists of a query to select the records to be registered and a call the `ALG_EVENT_INTERFACE_PCK` package to save the identification of the selected records:

```
procedure detector_t
( pi_name      in varchar2
, pi_date_from in date
)
is
begin
  for l_rec in ( select rel.id
                 ,      rel.laatste_mutatie_moment
                 from   rbh_relaties rel
                 where  rel.laatste_mutatie_moment >= pi_date_from
               )
  loop
    alg_event_interface_pck.add_event
    (pi_name => pi_name
    ,pi_code => to_char(l_rec.id)
    ,pi_date => l_rec.laatste_mutatie_moment
    );
  end loop;
end detector_t;
```

5.1.3. Begin Handler

The begin handler `OHI_EVENT_DEMO_PCK.START_HANDLER_T` in this example is used to open the file for writing the identifications of the relations.

```
procedure start_handler_t
( pi_name          in varchar2
, pi_run_nr        in number
, pi_date_detection in date
)
is
  l_filename     varchar2(100);
  l_location     varchar2(100) := 'OZG_TMP';
  l_max_linesize constant binary_integer := 32767;
begin
  l_filename := pi_name||'_'||to_char(pi_run_nr)||'_'||to_char(pi_date_detection,'YYYYMMDDHH24MISS')||'.txt';
  a_file_handle := utl_file.fopen(l_location,l_filename,'w',l_max_linesize);
end start_handler_t;
```

5.1.4. Handler

The handler OHI_EVENT_DEMO_PCK.HANDLER_T of the example writes the data to the opened file. The code parameter contains the identification of the relation record. It could be used to select more detailed information from the relation record. For instance who and when the last modification was made. Since this is a detected event the data would however only reflect the **last** modification.

```
procedure handler_t
( pi_code           in varchar2
, pi_date_source    in date
, pi_date_detection in date
)
is
begin
  utl_file.put_line( a_file_handle
                   , pi_code||' -> '||
                     to_char(pi_date_detection,'YYYY-MM-DD HH24:MI:SS')||' -> '||
                     to_char(pi_date_source,'YYYY-MM-DD HH24:MI:SS')
                   );
end handler_t;
```

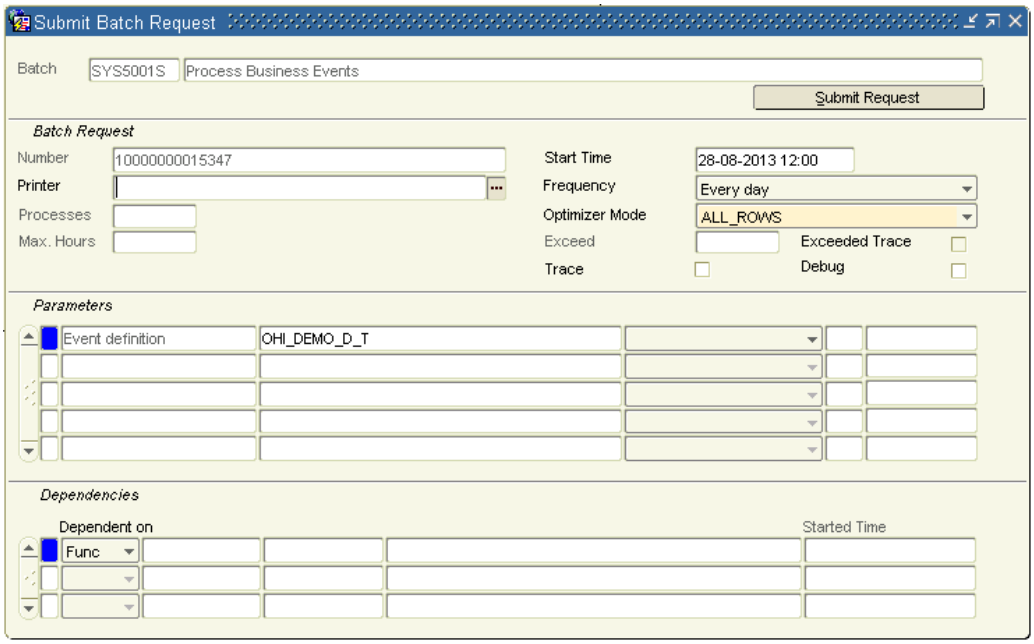
5.1.5. End Handler

The end handler `OHI_EVENT_DEMO_PCK.END_HANDLER_T` in this example is used to close the file.

```
procedure end_handler_t
( pi_name           in varchar2
, pi_run_nr         in number
, pi_date_detection in date
)
is
begin
  if utl_file.is_open(a_file_handle)
  then
    utl_file.fclose(a_file_handle);
  end if;
end end_handler_t;
```

5.1.6. Scheduling the Event

All steps for the detected event have finished. The event can be scheduled to run using Submit Batch Request (SYSS003F) window.

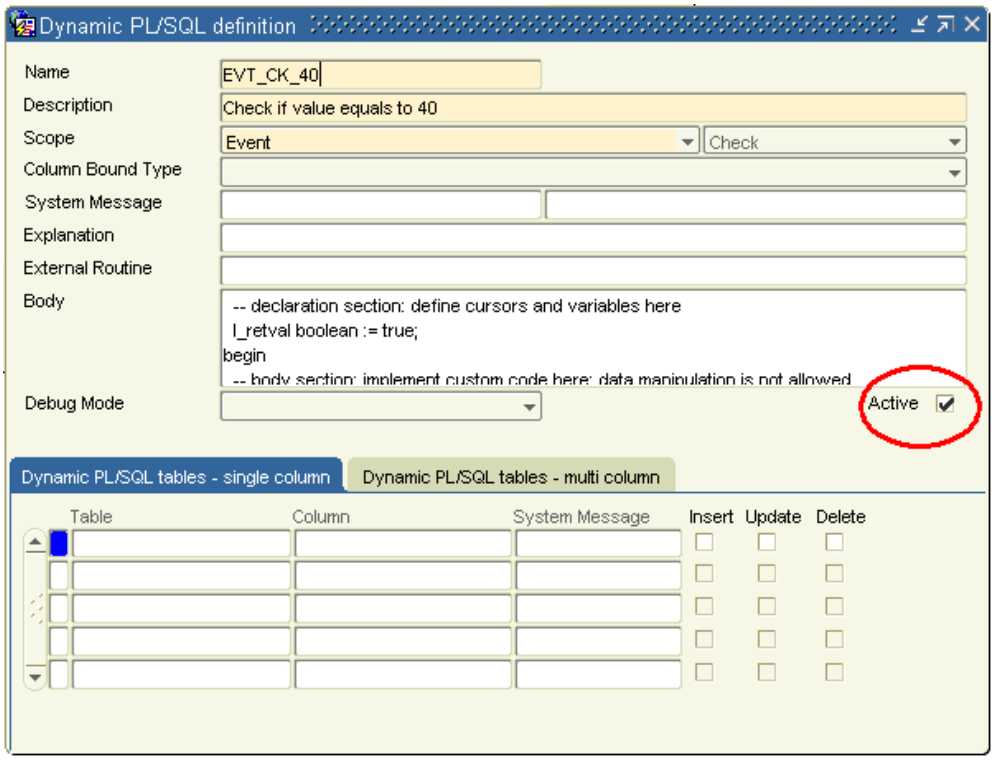


5.2. Triggered Event, Store to the Queue

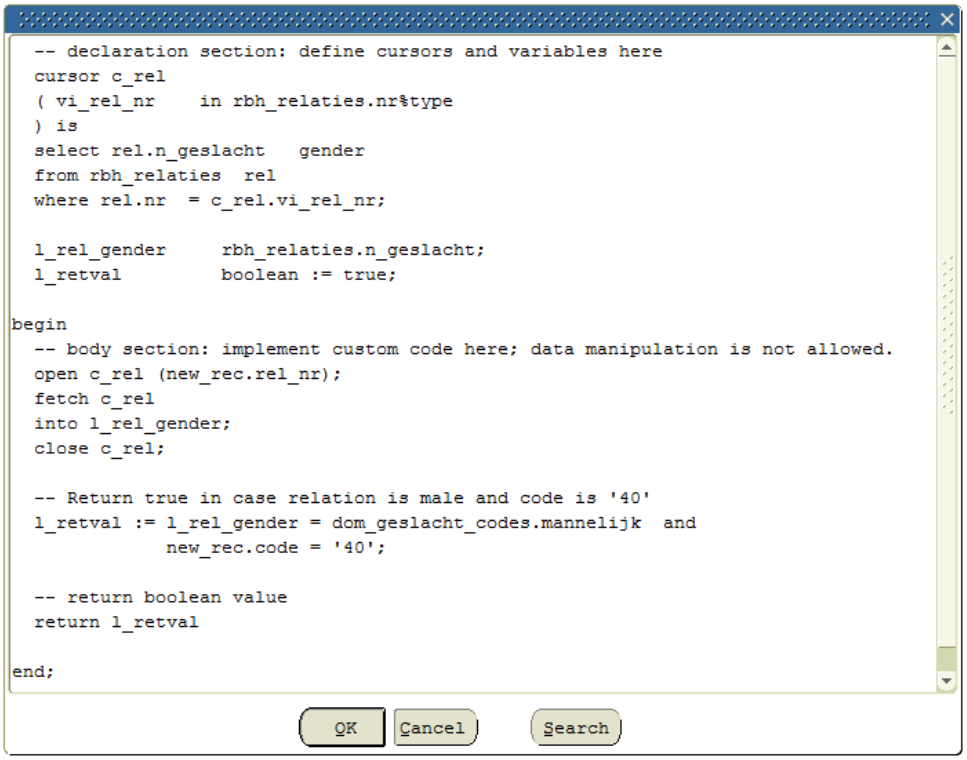
This example shows how to define an event that will be registered when a new record is created.

5.2.1. Evaluation Function

First step is creating the evaluation function which will be used in the event definition. OHI Back Office will validate the entered PL/SQL code when it is saved. Since the PL/SQL Definition has not yet been linked to a table this validation will fail. Therefore the indicator 'Active' should not be checked. In that case the PL/SQL code is disabled and will not be validated.



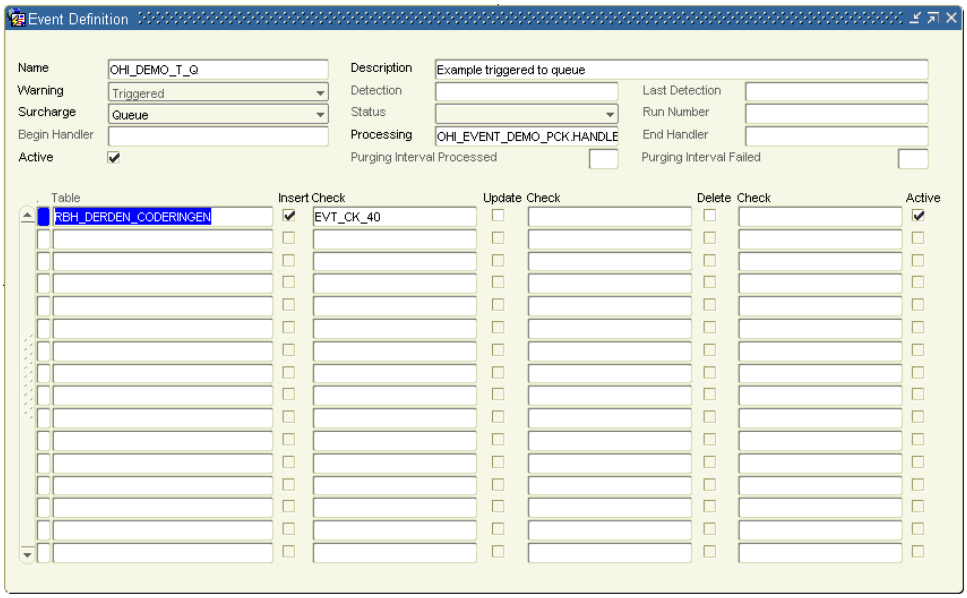
The complete Body of the PL/SQL definition is not visible, it contains:



The example shows that when a record is created for a male relation and the code is equal to '40' the function will return true and an event will be registered.

Since it is linked to the `RBH_DERDEN_CODERINGEN` table in the event definition, the `new_rec` will hold all the new values of the record. It can be used for more sophisticated evaluation than this example.

5.2.2. Event Definition



The screenshot shows example event definition named OHI_DEMO_T_Q. It is a triggered event (Type is set to Triggered) and it will store events to Business Event Framework queue (Storage set to Queue). This type of event will be signaled by

database triggers so a separate Detector is not needed. A begin handler and end handler are not required when the storage clause is set to Queue as the events will be handled.

The event will be signaled by the creation of a record in the `RBH_DERDEN_CODERINGEN` table when the dynamic PL/SQL definition EVT_CK_40 returns true.

'Active' indication

After the PL/SQL Definition has been linked to a table the 'Active' indication in the Dynamic PL/SQL Definition window must be checked to validate and enable the code.

5.2.3. Handler

The events are handled by `OHI_EVENT_DEMO_PCK.HANDLER_Q`. It takes the identification of the record from the object type it receives as parameter and sends an email to notify another department for instance.

```
procedure handler_q
( pi_load  in alg_ede_payload_tp
)

is

  c       utl_smtp.connection;
  l_code  varchar2(100);

  procedure send_header
  ( name   in varchar2
  , header in varchar2
  ) is
  begin
    utl_smtp.write_data(c, name || ': ' || header || UTL_TCP.CRLF);
  end;

begin

  l_code := alg_event_interface_pck.type_payload_to_code (pi_ede_payload => pi_load );

  c      := utl_smtp.open_connection('smtp-server.oracle.com');

  utl_smtp.helo(c, 'oracle.com');
  utl_smtp.mail(c, 'sender@oracle.com');
  utl_smtp.rcpt(c, 'recipient@oracle.com');

  utl_smtp.open_data(c);

  send_header('From', '"Sender" <sender@oracle.com>');
  send_header('To', '"Recipient" <recipient@oracle.com>');
  send_header('Subject', 'Relation created');

  utl_smtp.write_data(c, UTL_TCP.CRLF || 'Relation with code '||l_code||' has been created.');
  utl_smtp.close_data(c);
  utl_smtp.quit(c);

end;
```

Appendix 1 – ALG_EVENT_INTERFACE_PCK

Procedures		Parameters		
Name	**Description**	**Name**	**Type**	**Description**
install	Creates an event definition in the database. Also available as function returning alg_event_definities.id%type.	pi_name	alg_event_definities.naam%type	The name of the event definition.
		pi_description	alg_event_definities.oms%type	The description of the event definition.
		pi_event_type	alg_event_definities.type_signalering% type	Indicates how the event is signaled. Allowable values: D for Detected events T for Triggered events
		pi_storage	alg_event_definities.type_opslag%type	Indicates how events are stored. Allowable values: T for Table Q for Queued
		pi_handler	alg_event_definities.handler%type	The (package) procedure for the handler of the event.
		pi_detector	alg_event_definities.detector%type	The (package) procedure for the detector of the event.
		pi_begin_handler	alg_event_definities.begin_handler% type	The (package) procedure for the begin handler of the event.
		pi_end_handler	alg_event_definities.end_handler%type	The (package) procedure for the end handler of the event.
		pi_purge_processed	alg_event_definities.schoningsinterval_ verwerkt%type	Determines after how many days successfully processed events can be deleted from the event table.
		pi_purge_failed	alg_event_definities.schoningsinvterval_ mislukt%type	Determines after how many days failed events can be deleted from the event table.
deinstall	Removes an event definition form the database. When events still exist for this definition an error is given.	pi_name	alg_event_definities.naam%type	The name of the event definition to be removed from the database.
deinstall	Removes an event definition form the database. When events still exist for this definition an error is given.	pi_ede_id	alg_event_definities.id%type	The ID of the event definition to be removed from the database.
purge_all_events	Removes all events for the given event definition from the database. Can be used prior to the deinstall procedure to remove all events.	pi_name	alg_event_definities.naam%type	The name of the event for which all the event occurrences will be removed.
purge_all_events	Removes all events for the given event definition from the database. Can be used prior to the deinstall procedure to remove all events.	pi_ede_id	alg_event_definities.id%type	The ID of the event definition for which all the event occurrences will be removed.
reapply_failed_ event	Reset events with the status 'Failed' from a previous run.	pi_name	alg_event_definities.naam%type	the unique name of the event definition
		pi_code	alg#events.code%type	the identifying code of the event
reapply_failed_ event	Reset events with the status 'Failed' from a previous run.	pi_ede_id	alg_event_definities.id%type	the unique identifier of the event definition
		pi_code	alg#events.code%type	the identifying code of the event
add_event	This procedure must be called by the detector of an event to add an occurrence of the event to the event table or queue.	pi_name	alg_event_definities.naam%type	The name of the event definition.
		pi_code	alg#events.code%type	The identifying code of the event. Must be in format: Table_id##record_id##dml_type. E.g. 1234##876##U.

Procedures		Parameters		
Name	**Description**	**Name**	**Type**	**Description**
	When the storage type of the event is set to Table, the event is only created in case there is no existing event with the given code for the event definition with a status N(ew). When the storage type of the event is set to Queue, the event is always placed on the queue.	pi_date	alg#events.master_date%type	Optional timestamp for ordering event handling.
add_event	This procedure must be called by the detector of an event to add an occurrence of the event to the event table of queue. When the storage type of the event is set to Table, the event is only created in case there is no existing event with the given code for the event definition with a status N(ew). When the storage type of the event is set to Queue, the event is always placed on the queue.	pi_ede_id	alg_event_definities.id%type	The ID of the event definition.
		pi_code	alg#events.code%type	The identifying code of the event. Must be in format: Table_id##record_id##dml_type. E.g. 1234##876##U.
		pi_date	alg#events.master_date%type	Optional timestamp for ordering event handling.
add_event	This procedure must be called by the detector of an event to add an occurrence of the event to the event table or queue. In case the storage type of the event is set to Table, the event is only created when there is no existing event with the given code for the event definition with a status N(ew). In case the storage type of the event is set to Queue, the event is always placed on the queue.	pi_ede_payload	alg_ede_payload_tp	The object type containing the data of the event.
type_payload_to_code	Function which converts a storage type Queue payload type to a storage type Table format. Returns alg#events.code in the format table_id##record_id##dml_type. E.g. 1234##876##U	pi_ede_payload	alg_ede_payload_tp	The object type containing the data of the event.
code_payload_to_type	Function which converts a storage type Queue payload type to a storage type Table format. Returns alg_ede_payload_tp.	pi_name	alg_event_definities.naam%type	The name of the event definition.
		pi_code	alg#events.code%type	The identifying code of the event. Must be in format: Table_id##record_id##dml_type. E.g. 1234##876##U.
code_payload_to_type	Function which converts a storage type Queue payload type to a storage type Table format. Returns alg_ede_payload_tp.	pi_ede_id	alg_event_definities.id%type	The id of the event definition.
		pi_code	alg#events.code%type	The identifying code of the event. Must be in format: Table_id##record_id##dml_type. E.g. 1234##876##U.

Appendix 2 – Framework Tables

ALG_EVENT_DEFINITIES

This table holds the event definitions and contains the following

- NAAM (Name)
  This is a logical name for the event type, for example: AZR_MODIFY_PARTY.
- OMS (Description)
  For example: Export updates to parties to the XYZ system.
- DETECTOR
  A custom plug-in procedure which is called to detect events for this event definition. Example: azr_mod_prt_pck.detector.
- BEGIN_HANDLER
  A plug-in procedure that must be called once before processing events for this type, for example to create a .csv file to which all event data will be written.
- HANDLER
  A plug-in procedure which is called for every event that must be processed. Example: my_event_pck.handler.
- END_HANDLER
  A plug-in procedure which is called once after all events have been processed, for example to close a .csv file to which all event data were written.
- LAATSTE_DETECTIE_DATUM (last_detection_date)
  Used by the framework to record the last date when the detection mechanism was used.
- STATUS
  Updated by the framework to avoid multiple starts of the framework for this event definition.
  The status can be: 'K' (ready) or 'L' (running).
- RUN_NR
  Managed by the framework. All events that were detected in a single run are given the same run number for later processing and reporting.
- SCHONINGSINTERVAL_VERWERKT (Purge interval processed)
  Defines when (successfully) processed events for this definition may be purged. The default interval is 7 days.
- SCHONINGSINTERVAL_MISLUKT (Purge interval failed)
  Defines when failed events for this definition may be purged. The default interval is 27 days.
- IND_ACTIEF (Active indicator)
  Indicates whether the event is currently active.

ALG_EVENT_INIT_WIJZIGINGEN

This table holds the tables which are monitored by the event for triggered events. It contains:

- EDE_ID
  Foreign key to ALG_EVENT_DEFINITIES.
- TAB_ID
  Foreign key to ALG_TABELLEN.
- IND_INSERT
  Indicates whether insert actions on the table should be signaled.
- DPS_ID_INSERT
  Foreign key to ALG_DYN_PLSQL_DEFINITIES to fine-tune the insert trigger.
- IND_UPDATE
  Indicates whether update actions on the table should be signaled.
- DPS_ID_UPDATE
  Foreign key to ALG_DYN_PLSQL_DEFINITIES to fine-tune the update trigger.
- IND_DELETE
  Indicates whether delete actions on the table should be signaled.
- DPS_ID_DELETE
  Foreign key to ALG_DYN_PLSQL_DEFINITIES to fine-tune the delete trigger.
- IND_ACTIEF (Active indicator)
  Indicates whether the event is currently active.

ALG#EVENTS

This table stores events with storage clause set to Table.

- EDE_ID
  Foreign key to the ALG_EVENT_DEFINITIES table for the event definition that signaled this event.
- EDE_RUN_NR
  Set by the framework to group event occurrences. The highest run number is stored in the event definition.
- CODE
  Code retrieved by the detection plug-in for use as a key to process the event. In most cases this will be the primary key that can be used to find the data with which the event is to be processed. To be compatible with storing the event to the queue this should be in format table_id##record_id##DML-type.
- STATUS
  Records the processing status of an event occurrence. Possible values: 'N' (new), 'O' (pending), 'V' (processed), 'M' (failed).
- DATUM_ORIGINEEL (original date)
  An optional column that can be used to determine the processing order.

- CREATIE_MOMENT (creation date)
  This standard column is used for processing in the correct order if the DATUM_ORIGINEEL has not been set.

ALG#EVENT_ERRORS

This table holds event errors.

- EDE_ID (event definition ID)
  Refers to the event definition that detected this event.
- TAB_ID (table ID)
  Refers to the table on which the DML was executed.
- RECORD_ID
  Refers to the record of the changed record. Together with TAB_ID this uniquely identifies the record in OHI Back Office.
- DML_TYPE
  What DML action caused the event
- EET_ID (event ID)
  Refers to the event in ALG#EVENTS table.
- CODE
  Code for event processing.
- CREATIE_MOMENT (creation date)
  Timestamp when the error occurred
- FOUTCODE (Error code)
  The code of the error occurred.
- FOUTMELDING (Error message)
  The error message for the error that occurred.

ALG_EDE_PAYLOAD_TP

The object for storing events to the queue contains the following:

- EDE_ID (event definition ID)
  Refers to the event definition that detected this event.
- TAB_ID (table ID)
  Refers to the table on which the DML was executed.
- RECORD_ID
  Refers to the record of the changed record. Together with TAB_ID this uniquely identifies the record in OHI Back Office.
- DML_TYPE
  What DML action caused the event