2

Implementing CIP event types

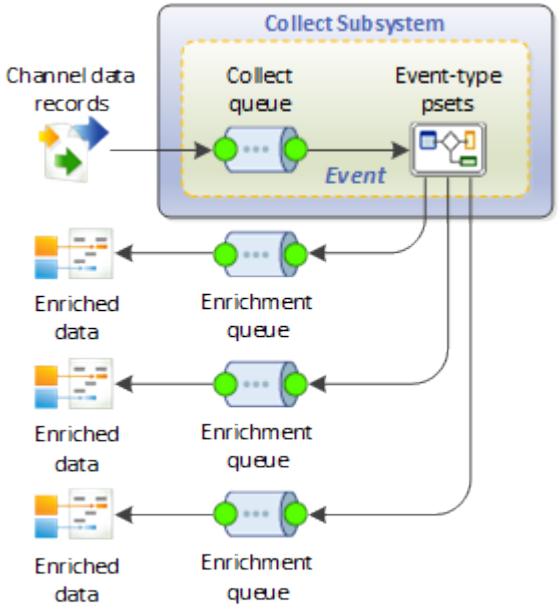
This chapter describes how to define and implement CIP event types and corresponding event processing triggers.

This chapter covers the following topics:

Overview of CIP events

In order for CIP to process data records received through a CIP channel, those data records must first be written to a CIP *collect queue* in the CIP *collect subsystem*. For each defined *event type*, the CIP collect subsystem launches a corresponding CIP *event-type pset*, which reads the collect queue, performs required preprocessing, and then routes the data records to one or more *enrichment queues* for further transformation, enrichment, and other processing.

The following figure illustrates this process:

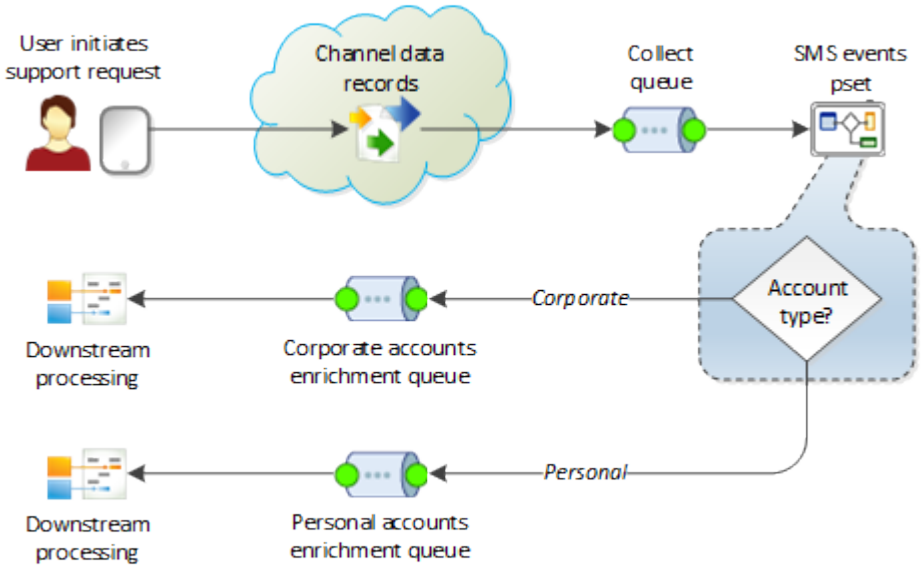


A CIP *event* consists of the instantiation of a CIP collect queue and an associated event-type pset that reads from that queue. CIP event-type psets implement the logic by which channel data records are subsequently routed to one or more downstream enrichment queues.

In the CIP UI, event types and event-type fields (derived from the event-type record formats) provide the building blocks that users can use as inputs when the they create eligibility or fulfillment rules. For example, the following figure shows an Activation event in the Eligibility Rule dialog for a journey stage.

For example, you may have an inbound channel through which customers can send text messages with technical support questions. Depending on the type of support contract the customer has, you may want to route their support questions to one or another support team. Similarly, you may want to route questions separately from corporate accounts and personal accounts, or separate hardware issues from website issues, or triage issues by reported severity.

The following figure illustrates this kind of workflow:



Structure of CIP events and event types

The software elements that make up CIP events and event types are summarized in the following table:

Element	Description
CIP collect subsystem	Provides CIP event collection and processing capabilities. The CIP collect subsystem runs as a service, and must be running in order for any CIP event processing to occur. The collect subsystem is driven by two primary plans: • **$COLLECT_INTEGRATE_PLAN/collect_event_subsystem.plan** — Looping plan that starts common CIP event collection and routing services, and that initiates customer-specific event service plans. • **$***public-project-prefix***_PLAN/collect_integrate.plan** — Plan that initiates customer-specific event collection and routing.
CIP channels	CIP inbound, outbound, and fulfillment channels through which channel data records are transmitted.
Collect and enrichment queues	Queues to which channel data records are received and to which preprocessed data records are routed as part of CIP event handling: • **Collect queues** — Queues to which CIP channel data records are written and then consumed by CIP event handler graphs. • **Enrichment queues** — Queues to which CIP event handler graphs route and write preprocessed data records for subsequent enrichment and processing.
Collect and enrichment queue DML files	Record formats for the collect and enrichment queues associated with your defined event types. By default, all customer-specific queue DML files should be stored in **$***public-project-prefix***_DML/source_event_types** (**$CIP_COMMON_EVENT_DML**).
Event handler graphs	Event handler service graphs that implement event-specific psets: • **$COLLECT_INTEGRATE_MP/collect_event.mp** — Generic CIP graph. • **$***public-project-prefix***_MP/collect_event.mp** — Customer-specific graph created for your implementation environment. Note that this graph is named **collect_event.mp** by default, but you can use a different name if you prefer.

Element	Description
Event-specific psets	Event-specific psets, implemented by **collect_event.mp**, that contain the logic for preprocessing event data records before handing off to one or more enrichment queues for subsequent processing. All event-specific psets must be stored in **$CIP_COMMON_PRIVATE_PSET/collect_integrate/event**. The collect subsystem service iterates over this **event** directory, and all psets in the directory are automatically run by the subsystem plan. CAUTION! Your event-specific psets *must* be stored in the **event** directory; if they are not, they will not be processed by the CIP collect subsystem plan.
Event-type DML files	DML files that define the record formats for the event types that you want make available in CIP. Each event type must have a corresponding DML file. Event-type DML files are stored in the following locations: • **$CIP_COMMON_DML/event_types** — Common event-type DML files that are included with CIP. • **$***public-project-prefix*_**DML/event_types** (**$CIP_COMMON_EVENT_DML**) — Customer-specific event-type DML files.
Event-type enumerations	DML files containing numeric IDs and name strings that correspond to the event types that you have defined. These numeric IDs and name strings are used to reference event-type DML during event processing. The name strings are also presented to users in the CIP UI. Event-type enumerations are defined in the following DML files: • **$CIP_COMMON_DML/event_type_ids.dml** — Enumerations for the common event types defined in **$CIP_COMMON_DML/event_types**. • **$***public-project-prefix*_**DML/event_type_ids.dml** — Enumerations for the customer-specific event types defined in **$***public-project-prefix*_**DML/event_types** (**$CIP_COMMON_EVENT_DML**).
Event-type DML file references	References to the record formats defined in **event_types** DML files, made by means of **include** declarations and conditional statements in the **$***public-project-prefix*_**DML/event.dml** (**$CIP_COMMON_EVENT_DML**) file.
Event hash transforms	Transforms used by CIP to generate a unique key — an *event hash* — based on a defined set of fields for each event. These hashes map to event IDs defined in **event_type_ids.dml**. CIP ships with a default **cip_event_hash()** function in **$CIP_COMMON_XFR/common_functions.xfr**, but you must define your event type-specific hash functions in one or both of the following files: • **$AI_XFR/event_hash.xfr** — Hash functions for customer-specific event types. • **$CIP_COMMON_XFR/private_stub/event_hash.xfr** — Stub file for common event hash functions.

Element	Description
Dimensioning transform functions	Transform functions for keys, or *dimensions*. The default event types that are shipped with CIP do not support dimensioning. However, if you want to define dimensioning functions for your project-specific event types, CIP includes a prototype transform template that you can use. The prototype transform template is named **$CIP_COMMON_XFR/prototype/getEventParamKeys.xfr**. After defining any dimensioning functions for your project-specific events, you must save your transform file as **$AI_XFR/getEventParamKeys.xfr**. In the context of CIP event handling, dimensions are primarily used to enable aggregations. For example, you may want to perform reloads based on several dimensions.
CIP database entries	Entries for event-type definitions in the CIP database. These database entries are required in order to present events and event fields in the CIP UI. The affected tables are the following: • dml_entity • dml_entity_field • dml_entity_measures • dml_entity_field_value You update the CIP database by adding a SQL file to **$***public-project-prefix*_**SQL/cip_db_data** and then running **$AI_PSET/utility/create_environment/cip_db_data/upgrade_database.pset**.
CIP user interface	In the CIP UI, event types and event-type fields (derived from the event-type record formats). These fields provide the building blocks that CIP users can use to create rules based on events.

Files and directories used in CIP events

CIP collect and event processing features rely on a number of plan, graph, pset, transform, and SQL files that are spread across various locations in your project sandbox trees. Listed below are the most important of these files and locations.

TIP: This is not meant to be a comprehensive list of every file and directory involved in CIP collect and event processing, but rather only a summary of the files and locations that you will most likely need to know about when you implement CIP events.

Filename/directory	Sandbox path	Description
cip_db_data directory	**$***public-project-prefix***_SQL**	Directory containing SQL scripts for adding event definitions to the CIP database.
collect_event.mp	**$***public-project-prefix***_MP** (**$COLLECT_INTEGRATE_MP**)	Event handler service graphs that implement event-specific psets.
collect_event_subsystem.plan	**$COLLECT_INTEGRATE_PLAN**	Looping plans that start common CIP event collection and routing services.
collect_integrate.plan	**$***public-project-prefix***_PLAN** (**$CIP_COMMON_PRIVATE_PLAN**)	Plan that initiates customer-specific event collection and routing services.
common_functions.xfr	**$CIP_COMMON_XFR**	Default CIP **cip_event_hash()** function.
event.dml	**$***public-project-prefix***_DML** (**$CIP_COMMON_EVENT_DML**)	Event-type DML reference file; contains references to the event-type DML files defined in the **event_types** directory.
event_hash.xfr	**$AI_XFR** **$CIP_COMMON_XFR/private_stub**	Hash functions for customer-specific event types, and stub file for common hash functions, respectively.
event_type_ids.dml	**$CIP_COMMON_DML** **$***public-project-prefix***_DML** (**$CIP_COMMON_EVENT_DML**)	Enumerations for common and customer-specific event types.
event_types directory	**$CIP_COMMON_DML** **$***public-project-prefix***_DML** (**$CIP_COMMON_EVENT_DML**)	Directories containing DML record format files for common and customer-specific event types, respectively.

Filename/directory	Sandbox path	Description
event directory	**$CIP_COMMON_PRIVATE_PSET/ collect_integrate**	Directory containing all event-specific psets. All event-specific psets must be stored in the **event** directory; if they are not, they will not be processed by the CIP collect subsystem plan.
source_event_types directory	**$***public-project-prefix***_DML** (**$CIP_COMMON_EVENT_DML**)	Directory containing all DML record format files for the queues associated with your defined event types.
upgrade_database.pset	**$AI_PSET/utility/ create_environment/cip_db_data**	Database upgrade script for adding your event-type entries to the CIP database.

Tasks for defining CIP event types

For each event type that you want to enable in CIP, you must create or modify a number DML files, psets, parameters, and CIP database entries. The following is a summary of the tasks involved in defining and implementing CIP event types. The rest of this section describes these tasks in more detail.

NOTE: Be sure to check in your implementation code to the technical repository. Ensure that other CIP developers then check out your code as needed.

Task 1: Add event-type entries to the event_type_ids.dml file

The **event_type_ids.dml** file contains numeric IDs and descriptive name strings that correspond to the CIP event types that you will define in "Task 4: Create event-type DML files". Each CIP event type that you define must have a corresponding entry in **event_type_ids.dml**.

File locations

event_type_ids.dml files are located in the following two directories:

- **$CIP_COMMON_DML** — Enumerations for the common event types defined in the **$CIP_COMMON_DML/event_types** directory. These are the default event types that are shipped with CIP.
- **$***public-project-prefix*_**DML** (**$CIP_COMMON_EVENT_DML**) — Enumerations for the customer-specific event types defined in the **$***public-project-prefix*_**DML/event_types** directory. All entries for customer-specific event types should be created in this file.

NOTE: Be sure to put *all* customer-specific event-type enumerations in **$***public-project-prefix*_**DML/event_type_ids.dml**. In most cases, you should not modify **$CIP_COMMON_DML/event_type_ids.dml**.

TIP: Customer-specific event-type files are maintained by the customer and are not promoted with CIP.

File syntax

For each CIP event type, you must create a pair of corresponding entries in **event_type_ids.dml**. The first entry in the pair is a unique *name=integer* constant for the event type. The second entry in the pair is corresponding *name=description* constant.

The syntax for each entry pair is as follows:

```
constant int_t ET_event-type = integer;
constant string_t ETN_event-type = description;
```

where:

- **ET**_*event-type* is the **ET**_ prefix followed by unique name for the event type.
- *integer* is a unique integer ID for the event type. For customer-specific event types, this integer value must be greater than **1000**. Integer values **1000** and below are reserved for the common default event types that are shipped with CIP.
- **ETN**_*event-type* is the **ETN**_ prefix followed by the same *event-type* name that you used for **ET**_*event-type*.
- *description* is a descriptive name for the event type. This name will be displayed in the CIP UI.

For example, to define an event type named **HANDSET_CHANGES** with a numeric ID of **1008**, you would add the following pair of entries to **$***public-project-prefix*_**DML/event_type_ids.dml**:

```
constant int_t ET_HANDSET_CHANGES = 1008;
constant string_t ETN_HANDSET_CHANGES = "HANDSET_CHANGES";
```

Required include statement

If the **$***public-project-prefix*_**DML/event_type_ids.dml** file does not already contain it, be sure to add the following line to the top of the file:

```
include "~$CIP_COMMON_DML/common_types.dml";
```

Example event_type_ids.dml files

This section lists the default **event_type_ids.dml** files that are included in various directories in a default CIP installation.

$CIP_COMMON_DML/prototype directory

The following shows the contents of the default **event_type_ids.dml** file that is included in the **$CIP_COMMON_DML/prototype** directory:

```
include "~$CIP_COMMON_DML/common_types.dml";

constant int_t ET_EVENT1 = 1001;
constant int_t ET_EVENT2 = 1002;

constant string_t ETN_EVENT1 = "EVENT1";
constant string_t ETN_EVENT2 = "EVENT2";
```

$CIP_COMMON_DML directory

The following example shows the contents of the default **event_type_ids.dml** file that is included in the **$CIP_COMMON_DML** directory:

```
include "~$CIP_COMMON_DML/common_types.dml";

constant int_t ET_LOG = 50;
constant int_t ET_ERROR = 51;
constant int_t ET_ERROR_ENRICHED = 52;
constant int_t ET_TRACING = 53;
constant int_t ET_SYS = 100;
constant int_t ET_TIMER = 101;
constant int_t ET_CAMPAIGN_EVENT = 102;
constant int_t ET_CAMPAIGN_EVENT_CAMPAIGN_START = 102; /* this is essentially an
alias of ET_CAMPAIGN_EVENT. See transpile_rules.xfr for more details */
constant int_t ET_BATCH_TRIGGER = 103;
constant int_t ET_CAMPAIGN_EVENT_CAMPAIGN_ENDED = 104;
constant int_t ET_AGGREGATION_QUERY_SPEC = 6;
//constant int_t ET_FULFILLMENT_RESPONSE = 9;
constant int_t ET_NBOS_FULFILLMENT_REQUEST = 19;
constant int_t ET_NBOS_DELIVERY_REPORT = 20;
constant int_t ET_OUTBOUND_CHANNEL_FULFILLMENT_REQUEST = 24;
constant int_t ET_ACTIVATION = 0;
constant int_t ET_SMS_REPLY = 2;
constant int_t ET_NBOS_OUTBOUND_EVENT = 3;

constant string_t ETN_SYS = "SYS";
constant string_t ETN_TIMER = "TIMER";
constant string_t ETN_CAMPAIGN_EVENT = "CAMPAIGN";
constant string_t ETN_CAMPAIGN_EVENT_CAMPAIGN_START = "CAMPAIGN_START";
constant string_t ETN_CAMPAIGN_EVENT_CAMPAIGN_ENDED = "CAMPAIGN_ENDED";
constant string_t ETN_BATCH_TRIGGER = "BATCH_TRIGGER";
constant string_t ETN_AGGREGATION_QUERY_SPEC = "AGGREGATION_QUERY_SPEC";
//constant string_t ETN_FULFILLMENT_RESPONSE = "FULFILLMENT RESPONSE";
constant string_t ETN_NBOS_FULFILLMENT_REQUEST = "ET_NBOS_FULFILLMENT_REQUEST";
constant string_t ETN_OUTBOUND_CHANNEL_FULFILLMENT_REQUEST  =
 "ET_OUTBOUND_CHANNEL_FULFILLMENT_REQUEST";
constant string_t ETN_ACTIVATION = "ACTIVATION";
constant string_t ETN_SMS_REPLY = "SMS_REPLY";
constant string_t ETN_NBOS_OUTBOUND_EVENT = "NBOS_OUTBOUND_EVENT";

// Special signalling events within graphs, intended for use when parts of graphs
process
// their own log event streams (e.g. state_machine.mp)

constant string_t INTRNL_USER_EVT_AICIP_3181_RELOAD_PROFILE =
 'AICIP_3181_SYS_RELOAD_PROFILE';
```

$PUBLIC_PROTOTYPE_DML directory

The following shows the contents of the default **event_type_ids.dml** file that is included in the **$PUBLIC_PROTOTYPE_DML** directory.

NOTE: The integer ID for each customer-specific event type that you define in **$***public-project-prefix***_DML/event_type_ids.dml** must have a unique value greater than **1000**.

```
include "~$CIP_COMMON_DML/common_types.dml";

constant int_t ET_EVENT1 = 1001;
constant int_t ET_EVENT2 = 1002;
constant int_t ET_EVENT3 = 1003;
constant int_t ET_UTF8_EVENT = 1008;

constant int_t ET_USAGE_VOICE = 1004;
constant int_t ET_USAGE_SMS = 1005;
constant int_t ET_RELOAD = 1006;
constant int_t ET_CAMPAIGN_START = 1007;

constant string_t ETN_EVENT1 = "EVENT1";
constant string_t ETN_EVENT2 = "EVENT2";
constant string_t ETN_EVENT3 = "EVENT3";

// Example Event definitions for CIP2 Real-time aggregate integration tests


constant string_t ETN_USAGE_VOICE = "voice_usage";
constant string_t ETN_USAGE_SMS = "sms_usage";
constant string_t ETN_RELOAD = "reload";
constant string_t ETN_CAMPAIGN_START = "campaign_start";
```

Task 2: Define or obtain queue details

You must define or obtain the following details about the collect and enrichment queues that you want to use for reading and writing CIP channel data sets. For information about creating queues, see "Managing Ab Initio Queues" in the *Continuous>Flows Guide*.

- **Collect queues** — For each event type:
  - CIP channel
  - Queue name
  - Event-type name
  - Queue filesystem type
  - Queue landing directory
  - DML requirements
- **Enrichment queues** — For each enrichment type:
  - Queue name
  - Queue filesystem type
  - Queue landing directory
  - DML requirements
- **All queue types** — For all queue types:
  - Landing filesystem username(s)/authorization(s)
  - Any environment-specific guidelines or requirements; for example, for DEV, PROD, SIT and so forth

Task 3: Create queue DML files

Create a DML file for each of the collect and enrichment queues that are associated with your defined event types. These DML files must use record formats that meet the requirements determined in "Task 2: Define or obtain queue details".

Queue DML file location

By default, all queue DML files should be stored in **$***public-project-prefix*_**DML/source_event_types** (**$CIP_COMMON_EVENT_DML**). However, you can store queue DML files in a different directory of your choice — for example, **sources** — under **$***public-project-prefix*_**DML**.

Queue DML file naming

We recommend that you name your queue DML files using the following naming pattern:

***event-type*_queue.dml**

where *event-type* is the name of an event type listed in the **$***public-project-prefix*_**DML/event_type_ids.dml** file.

For example, for an event type named **handset_change**, you would name its corresponding queue DML file **handset_change_queue.dml**.

Queue DML file syntax

Queue DML files use common DML syntax for defining record formats. For detailed information about writing and working with DML, see the *DML Guide and Reference*.

Example queue DML files

The following shows the contents of the default placeholder **event1.dml** file that is included in the **$PUBLIC_PROTOTYPE_DML/source_event_types** directory in the CIP **prototype** project:

```
include "~$CIP_COMMON_DML/common_types.dml";

type source_event1_t =
record
        long_t cust_prof_id;
        datetime("YYYY-MM-DD HH24:MI:SS") event_timestamp;
 string_t f1;
 string_t f2;
end;

metadata type = source_event1_t;
```

The following is an example of a complete queue DML file:

```
// Server: example-server-name
// Queue directory : $example-dir_QUEUE/example-event-type_queue

type output_type =
record
  datetime("YYYY-MM-DD HH24:MI:SS.NNNNNN")("\x01") sys_creation_date = NULL("");
```

```
  date("YYYY-MM-DD")("\x01") process_date = NULL("");
  decimal("\x01") customer_id = NULL("");
  decimal("\x01") subscriber_id = NULL("");
  utf8 string("\x01", maximum_length=2) event_type = NULL("");
  utf8 string("\x01", maximum_length=40) prim_resource_val = NULL("");
  decimal("\x01") pps_new_balance = NULL("");
  decimal("\x01") pps_amt = NULL("");
  utf8 string("\x01", maximum_length=200) cd_main_id = NULL("");
  utf8 string("\x01", maximum_length=200) 19_dealer_cgi = NULL("");
  utf8 string("\x01", maximum_length=200) soc_cd = NULL("");
  decimal("\x01") file_id = NULL("");
  utf8 string("\x01", maximum_length=200) session_counter = NULL("");
  datetime("YYYY-MM-DD HH24:MI:SS")("\x01") charging_date = NULL("");
  utf8 string("\x01", maximum_length=200) session_id = NULL("");
  decimal("\x01") source_id = NULL("");
  utf8 string("\x01", maximum_length=20) pym_cat = NULL("");
  string(1) newline = "\n";
end;

metadata type = output_type;
```

Task 4: Create event-type DML files

Create a DML file for each defined event type that you want to make available in the CIP UI. Each event type must have a corresponding DML file. These DML files must use record formats that correspond with the queue DML for the event type, as defined in "Task 3: Create queue DML files".

Event-type DML file locations

CIP event-type DML files are located in the following two directories:

- **$CIP_COMMON_DML/event_types** — Common event-type DML files that are included with CIP.
- **$***public-project-prefix***_DML/event_types** (**$CIP_COMMON_EVENT_DML/event_types**) — Customer-specific event-type DML files.

TIP: Customer-specific event-type DML files are maintained by the customer and are not promoted with CIP.

Event-type DML file naming

We recommend that you name your event-type DML files using the following naming pattern:

***event-type*.dml**

where *event-type* is the name of an event type listed in the **$***public-project-prefix***_DML/event_type_ids.dml** file.

For example, for an event type named **handset_change**, you would name its corresponding event-type DML file **handset_change.dml**.

Event-type DML file syntax

Event-type DML files use common DML syntax for defining record formats. For detailed information about writing and working with DML, see the *DML Guide and Reference*.

The field formats defined in any given event-type DML file must correspond to field formats that are defined in a related queue DML file. For example, the following table shows some hypothetical event-type field formats and their corresponding queue formats.

Event-type fields	Queue fields
`string_t brand = NULL("");`	`string("\x01") brand = NULL("");`
`string_t model = NULL("");`	`string("\x01") model = NULL("");`
`string_t imei = NULL("");`	`string("\x01") imei = NULL("");`
`string_t imsi = NULL("");`	`string("\x01") imsi = NULL("");`
`string_t target_segment = NULL("");`	`string("\x01") target_segment = NULL("");`

The string types that you should use, such as **string_t**, **string(little_int_t)**, or **string(tiny_int_t)**, will depend on the original data specifications — for example, contexts such as **varchar max length=200**.

Required include statement

If an event-type DML file does not already contain it, be sure to add the following line to the top of the file:

```
include "~$CIP_COMMON_DML/common_types.dml";
```

Example event-type DML files

The following shows the contents of a default placeholder DML file, named **event1.dml**, that is included in the **$PUBLIC_PROTOTYPE_DML/event_types** directory in the CIP **prototype** project:

```
include "~$CIP_COMMON_DML/common_types.dml";

type event1_t =
record
 string_t f1;
 string_t f2;
end;

metadata type = event1_t;
```

The following is an example of a complete event-type DML file:

```
include "~$CIP_COMMON_DML/common_types.dml";

type nbos_delivery_report_t =
record
  long_t subscriber_id = NULL;
  long_t msisdn;
  string_t campaign_id;
  dec_t cycle_number = 0;
  string_t offer_id;
end;

metadata type = nbos_delivery_report_t;
```

Task 5: Update the event.dml file

For each customer-specific CIP event type that you want to enable, you must create a corresponding set of references in the public project **event.dml** file (**$CIP_COMMON_EVENT_DML/event.dml**).

event.dml file locations

event.dml files are located in the following directories:

- **$CIP_COMMON_DML** — For common event types that are included with CIP. You should not need to modify the **event.dml** file in this directory.
- **$CIP_COMMON_DML/prototype** — Template file that you can use to create customer-specific **event.dml** files. You should not modify any of the files4 in this directory.
- **$***public-project-prefix*_**DML** (**$CIP_COMMON_EVENT_DML**) — For customer-specific event types. The **event.dml** file in this directory is the only one you should modify.

NOTE: Be sure to put *all* customer-specific event types in **$***public-project-prefix*_**DML/event.dml** only. In most cases, you should not modify **$CIP_COMMON_DML/event.dml**.

TIP: Customer-specific event-type files are maintained by the customer and are not promoted with CIP.

Structure and syntax of the event.dml file

The **event.dml** file contains the following sections:

Section	Description

References to event-type DML files	A set of **include** statements that point to event-type DML files. These **include** statements reference both **cip_common** and public project DML; for example:
	`include "~$CIP_COMMON_DML/event_types/batch_trigger.dml";`
	`include "~$CIP_COMMON_DML/event_types/campaign_event.dml";`
	`include "~$CIP_COMMON_PRIVATE_DML/event_types/activation.dml";`
	`include "~$CIP_COMMON_PRIVATE_DML/event_types/reload.dml";`
	NOTE: Be sure to include in this block all of the entries listed in "Required include statements for common CIP event types".

References to event_type_ids.dml files	**include** statements that point to the **event_type_ids.dml** files that are associated with the CIP event types that you want to enable. These **include** statements must reference the **event_type_ids.dml** files in both the **cip_common** project and your public project; for example:
	`include "~$CIP_COMMON_DML/event_type_ids.dml";`
	`include "~$CIP_COMMON_PRIVATE_DML/event_type_ids.dml";`

Primary event_details record format	Specific event types (and optionally methods) that are triggered by means of conditional statements that output a record named **event_details**. The record format for **event_details** is the same in all **event.dml** files, and must take the following form:
	`type event_details_t = record`
	`  int_t event_type_id;`
	`  string_t event_type;`
	`  ts_t event_time = NULL("");`
	`  ts_t received_time = NULL("");`
	`  event-type-conditionals`
	`  event-type-methods`
	`end;`

Section	Description

Event-type conditional statements	Conditional **if** statements that walk through the list of event types in order to output the record to the correct event handler. These conditional statements use the following syntax:
	`if(event_type_id == ET_`*`event-type`***`)`** *`format field`*`;`
	where:
	• **ET_***event-type-name* is the event-type name, including the **ET_** prefix, as defined in **event_type_ids.dml**.
	• *format* is the event-type record format, as defined in the DML file for the event type.
	• *field* is the target database field name for the record.
	For example:
	`if(event_type_id == ET_CAMPAIGN_EVENT) campaign_event_t campaign_event;`
	`if(event_type_id == ET_ACTIVATION) activation_t activation;`

Common subrecord formats	Format definitions for common subrecords. Every **event.dml** file must include several common subrecord format definitions, which are required for CIP event-handling features. For a detailed listing of these record formats, see "Required subrecord formats for CIP event handling".
Metadata type declaration	The following **metadata** declarations, which every **event.dml** file must end with: `metadata type = event_t;`

Required include statements for common CIP event types

If the **$***public-project-prefix***_DML/event.dml** file does not already contain the following lines, in addition to customer-specific event types, be sure to add them to the top of the file. These **include** statements are required to enable common CIP event types:

```
include "~$CIP_COMMON_DML/aggregates/aggregates.dml";
include "~$CIP_COMMON_DML/campaign_key_types.dml";
include "~$CIP_COMMON_DML/common_types.dml";
include "~$CIP_COMMON_DML/event_engine/engine_types.dml";
include "~$CIP_COMMON_DML/event_type_ids.dml";
include "~$CIP_COMMON_DML/event_types/aggregation_query_spec.dml";
include "~$CIP_COMMON_DML/event_types/batch_trigger.dml";
include "~$CIP_COMMON_DML/event_types/campaign_ended.dml";
include "~$CIP_COMMON_DML/event_types/campaign_event.dml";
include "~$CIP_COMMON_DML/event_types/nbos_fulfillment_request.dml";
include "~$CIP_COMMON_DML/event_types/outbound_channel_fulfillment_request.dml";
include "~$CIP_COMMON_DML/event_types/SYS.dml";
include "~$CIP_COMMON_DML/event_types/timer_event.dml";
include "~$CIP_COMMON_DML/fulfillment/fulfillment_response.dml";
include "~$CIP_COMMON_DML/subscriber_profile/subscriber_profile_payloaded.dml";
include "~$CIP_COMMON_PRIVATE_DML/event_type_ids.dml";
include "~$CIP_COMMON_PRIVATE_DML/event_types/activation.dml";
```

NOTE: Despite the ***_PRIVATE_*** part of the name, the **$CIP_COMMON_PRIVATE_DML** variables resolve to the customer's public project.

Required subrecord formats for CIP event handling

Every **event.dml** file must include the following subrecord format definitions. These record formats are required for CIP event handling features:

```
/* used for the state engine types */
type event_details_subset_t =
record
        int_t event_type_id;
        string_t event_type;
        ts_t event_time = NULL("");
        ts_t received_time = NULL("");
        system_event_t SYS = NULL;
        campaign_event_t campaign_event = NULL;
        timer_event_t timer_event = NULL;
end;

type event_t =
record
        key_types_t  keys;
        event_details_t event;
end;

type partial_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
Query Results" @*/;
        long_t partial_enrich_time /*@ BizName:"Partial Enrich Time" @*/;
end;

type enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
       subscriber_profile_payloaded_t subscriber_profile /*@ BizName:"Subscriber Profile"
@*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
Query Results" @*/;
        name_value_pair_t[int] personal_params = NULL;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
end;

type partial_eligible_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
       subscriber_profile_payloaded_t subscriber_profile /*@ BizName:"Subscriber Profile"
@*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
Query Results" @*/;
        name_value_pair_t[int] personal_params = NULL;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
        long_t eligible_time /*@ BizName:"Eligible Time" @*/;
        string_t[int_t] eligible_campaigns;
        string_t[int_t] fulfillment_campaigns;
        string_t[int_t] taker_campaigns;
        string_t subscriber_reply;
        vvoid_t event_context_payload = NULL("");
end;
```

```
type eligible_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
        long_t msisdn = NULL(0) /*Field: MSISDN_SUBS*/ /*@BizName:msisdn@*/;
        name_value_pair_t[int] personal_params = NULL;
        event_details_subset_t event;
        string_t[int_t] eligible_campaigns;
        string_t[int_t] fulfillment_campaigns;
        string_t[int_t] taker_campaigns;
        string_t subscriber_reply;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
        long_t eligible_time /*@ BizName:"Eligible Time" @*/;
        vvoid_t event_context_payload = NULL("");
end;
```

NOTE: Depending on when your CIP deployment was created, you may need to manually modify the **partial_eligible_enriched_event_t** and **eligible_enriched_event_t** formats so they include a **string_t subscriber_reply** field. Ensure that the **partial_eligible_enriched_event_t** and **eligible_enriched_event_t** record formats are exactly as shown in the preceding example.

Example event.dml file

The following shows the contents of the default **event.dml** file that is included in the **$PUBLIC_PROTOTYPE_DML** directory in the CIP **prototype** project:

```
include "~$CIP_COMMON_DML/common_types.dml";
include "~$CIP_COMMON_DML/aggregates/aggregates.dml";
include "~$CIP_COMMON_DML/event_engine/engine_types.dml";
include "~$CIP_COMMON_DML/fulfillment/fulfillment_response.dml";
include "~$CIP_COMMON_DML/event_types/timer_event.dml";
include "~$CIP_COMMON_DML/event_types/SYS.dml";
include "~$CIP_COMMON_DML/event_types/batch_trigger.dml";
include "~$CIP_COMMON_DML/event_types/campaign_event.dml";
include "~$CIP_COMMON_DML/event_types/campaign_ended.dml";
include "~$CIP_COMMON_DML/campaign_key_types.dml";
include "~$CIP_COMMON_DML/event_type_ids.dml";
include "~$CIP_COMMON_DML/event_types/aggregation_query_spec.dml";
include "~$CIP_COMMON_DML/event_types/nbos_fulfillment_request.dml";
include "~$CIP_COMMON_DML/event_types/outbound_channel_fulfillment_request.dml";

include "~$CIP_COMMON_PRIVATE_DML/event_type_ids.dml";
include "~$CIP_COMMON_DML/subscriber_profile/subscriber_profile_payloaded.dml";

include "~$CIP_COMMON_PRIVATE_DML/event_types/activation.dml";
include "~$CIP_COMMON_PRIVATE_DML/event_types/event1.dml";
include "~$CIP_COMMON_PRIVATE_DML/event_types/event2.dml";

type event_details_t = record
        int_t event_type_id;
        string_t event_type;
        ts_t event_time = NULL("");
        ts_t received_time = NULL("");

 // Standard events
        //if(event_type_id == ET_FULFILLMENT_RESPONSE) fulfillment_response_t
fulfillment_response;
        if(event_type_id == ET_SYS) system_event_t SYS;
        if(event_type_id == ET_TIMER) timer_event_t timer_event;
        if(event_type_id == ET_AGGREGATION_QUERY_SPEC) aggregation_query_spec_t
```

```
aggregation_query_spec;
        if(event_type_id == ET_CAMPAIGN_EVENT) campaign_event_t campaign_event;
       if(event_type_id == ET_CAMPAIGN_EVENT_CAMPAIGN_ENDED) campaign_ended_t campaign_ended;
        if(event_type_id == ET_BATCH_TRIGGER) batch_trigger_t batch_trigger;
        if(event_type_id == ET_NBOS_FULFILLMENT_REQUEST) nbos_fulfillment_request_t
nbos_fulfillment_request;
        if(event_type_id == ET_OUTBOUND_CHANNEL_FULFILLMENT_REQUEST)
outbound_channel_fulfillment_request_t outbound_channel_fulfillment_request;
        if(event_type_id == ET_ACTIVATION) activation_t activation;

 // Customer Events
        if(event_type_id == ET_EVENT1) event1_t event1;
        if(event_type_id == ET_EVENT2) event2_t event2;

        int_t accumulator_event() =  event_type_id == ET_ACTIVATION || event_type_id ==
ET_EVENT1 || event_type_id == ET_EVENT2 || event_type_id == ET_CAMPAIGN_EVENT;

        long_t bytes() = 0;

        long_t amount() = 0;

        long_t remaining_bal() = 0;

        long_t usage_units() = 0;

        int_t enrich_event() = event_type_id == ET_SYS || event_type_id ==
ET_AGGREGATION_QUERY_SPEC  ? 0 : 1;
end;

/* used for the state engine types */
type event_details_subset_t =
record
        int_t event_type_id;
        string_t event_type;
        ts_t event_time = NULL("");
        ts_t received_time = NULL("");
        system_event_t SYS = NULL;
        campaign_event_t campaign_event = NULL;
        timer_event_t timer_event = NULL;
end;

type event_t =
record
        key_types_t  keys;
        event_details_t event;
end;

type partial_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
Query Results" @*/;
        long_t partial_enrich_time /*@ BizName:"Partial Enrich Time" @*/;
end;

type enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
       subscriber_profile_payloaded_t subscriber_profile /*@ BizName:"Subscriber Profile"
@*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
```

```
Query Results" @*/;
        name_value_pair_t[int] personal_params = NULL;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
end;

type partial_eligible_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
       subscriber_profile_payloaded_t subscriber_profile /*@ BizName:"Subscriber Profile"
@*/;
        event_details_t event /*@ BizName:"Triggering Event" @*/;
       query_aggregate_result_instance_t[int_t] aggregate_results /*@ BizName: "Aggregate
Query Results" @*/;
        name_value_pair_t[int] personal_params = NULL;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
        long_t eligible_time /*@ BizName:"Eligible Time" @*/;
        string_t[int_t] eligible_campaigns;
        string_t[int_t] fulfillment_campaigns;
        string_t[int_t] taker_campaigns;
        string_t subscriber_reply;
        vvoid_t event_context_payload = NULL("");
end;

type eligible_enriched_event_t =
record
        key_types_t  keys /*@ BizName:"Triggering Event Keys" @*/;
        long_t msisdn = NULL(0) /*Field: MSISDN_SUBS*/ /*@BizName:msisdn@*/;
        name_value_pair_t[int] personal_params = NULL;
        event_details_subset_t event;
        string_t[int_t] eligible_campaigns;
        string_t[int_t] fulfillment_campaigns;
        string_t[int_t] taker_campaigns;
        string_t subscriber_reply;
        long_t aggregates_enrich_time /*@ BizName:"Aggregate Time" @*/;
        long_t enrich_time /*@ BizName:"Enrich Time" @*/;
        long_t eligible_time /*@ BizName:"Eligible Time" @*/;
        vvoid_t event_context_payload = NULL("");
end;

metadata type = event_t;
```

Task 6: Create event-specific psets

Create your event-specific psets and put them in the **$CIP_COMMON_PRIVATE_PSET/collect_integrate/event** directory. All event-specific psets *must* be stored in this directory. The CIP collect subsystem service iterates over the **event** directory, and all psets in the directory are automatically run by the subsystem plan.

When you create your psets, use the following naming pattern:

collect_event.*queue-name***.pset**

where *queue-name* is the name of the collect queue for the event type.

TIP: For example psets that you can use as starting points for your event-specific psets, look in the **$AI_PSET/collect_integrate/event** directory.

Task 7: Update CIP event hash transform files

For each event, CIP generates a unique key — an *event hash* — based on a defined set of fields for each event. These hashes map to event IDs defined in the **event_type_ids.dml** file.

CIP ships with a default **cip_event_hash()** function in **$CIP_COMMON_XFR/common_functions.xfr**, but you must define your event-type-specific hash functions in one or both of the following files:

- **$AI_XFR/event_hash.xfr** — Hash functions for customer-specific event types.
- **$CIP_COMMON_XFR/private_stub/event_hash.xfr** — Stub file for common event hash functions.

Task 8: Define dimensioning transforms

The default event types that are shipped with CIP do not support dimensioning. However, if you want to define dimensioning functions for your project-specific event types, CIP includes a prototype transform template that you can use.

The prototype transform template is named **$CIP_COMMON_XFR/prototype/getEventParamKeys.xfr**. After defining any dimensioning functions for your project-specific events, you must save your transform file as **$AI_XFR/getEventParamKeys.xfr**.

In the context of CIP event handling, dimensions (keys) are primarily used to enable aggregations. For example, you may want to perform reloads based on several dimensions.

Task 9: Update the public project .sandbox.pset file

For each event type, add corresponding event queue and queue subscriber parameters to the public project **.sandbox.pset** file; for example:

```
CUST3_COLLECT_INTEGRATE_SOURCE_QUEUE_DIR_CREDITCARD_EVENT = $CIP_COMMON_LOCAL_MFS_QUEUE
CUST3_COLLECT_INTEGRATE_SOURCE_SUBSCRIBE_ID_CREDITCARD_EVENT = cip
CUST3_COLLECT_INTEGRATE_SOURCE_QUEUE_DIR_CHKSAV_EVENT = $CIP_COMMON_LOCAL_MFS_QUEUE
CUST3_COLLECT_INTEGRATE_SOURCE_SUBSCRIBE_ID_CHKSAV_EVENT = cip
CUST3_COLLECT_INTEGRATE_SOURCE_QUEUE_DIR_CUSTOMER_REPLY_EVENT = $CIP_COMMON_LOCAL_SERIAL_QUEUE
CUST3_COLLECT_INTEGRATE_SOURCE_SUBSCRIBE_ID_CUSTOMER_REPLY_EVENT = cip
```

Task 10: Add a list of event-specific psets to project pset files

Add a list of your event-specific psets as an **@cip_common** override to the CIP_COMMON_COLLECT_INTEGRATE_EVENT_PSETS parameter in the public project **.project.pset** file. If there are environment-specific public project psets, add the override to the project psets that are relevant in the given environment. For example, you may need to add the list of psets to a **dev.pset**, **sit.pset**, and **prod.pset**.

For example, the following is an override containing references to five event-specific psets:

```
CIP_COMMON_COLLECT_INTEGRATE_EVENT_PSETS@cip_common =
cust3_collect_event.creditcard.pset
cust3_collect_event.chksav.pset
cust3_collect_event.customer_reply.pset
cust3_collect_event.mobile_app.pset
cust3_collect_event_dsg.geofence.pset
```

Task 11: Update the CIP database

You must add your event-type definitions to the CIP database. You do this by creating a SQL file with relevant event-specific entries, and then running **$AI_PSET/utility/create_environment/upgrade_all_databases.pset**.

► **To update the CIP database:**

1. Create a SQL file in with relevant event-specific entries in **$***public-project-prefix***_SQL/cip_db_data**.

   For each event type, your SQL file must insert entries into the following tables in the CIP database:

   - dml_entity
   - dml_entity_field
   - dml_entity_measures
   - dml_entity_field_value

   For example, the following is an SQL file containing entries for an event named **Event2**:

```
insert into dml_entity_field(dml_entity_id, dml_type, field_name, business_name, is_nullable)
VALUES ((select id from dml_entity where entity_name = 'Event2'), 'datetime_t', 'timestamp',
 'Event Time', true);

insert into dml_entity_field(dml_entity_id, dml_type, field_name, business_name, is_nullable)
VALUES ((select id from dml_entity where entity_name = 'Event2'), 'int_t', 'int_flag',
 'International Flag', true);

insert into dml_entity_field(dml_entity_id, dml_type, field_name, business_name, is_nullable)
VALUES ((select id from dml_entity where entity_name = 'Event2'), 'string_t', 'trans_type',
 'Transaction Type', true);

insert into dml_entity_field(dml_entity_id, dml_type, field_name, business_name, is_nullable)
VALUES ((select id from dml_entity where entity_name = 'Event2'), 'dec_t', 'trans_amount',
 'Transaction Amount', true);

insert into dml_entity_field(dml_entity_id, dml_type, field_name, business_name, is_nullable)
VALUES ((select id from dml_entity where entity_name = 'Event2'), 'dec_t', 'trans_duration',
 'Transaction Duration', true);

INSERT INTO "public"."dml_entity_measures" (entity_id,entity_field_name,measure_name,
measure_description,measure_type) VALUES ((select id from dml_entity where entity_name =
 'Event2'),'trans_amount','monetary_value','Monetary Value','long_t');
INSERT INTO "public"."dml_entity_measures" (entity_id,entity_field_name,measure_name,
measure_description,measure_type) VALUES ((select id from dml_entity where entity_name =
 'Event2'),'trans_duration','duration','Duration','long_t');

INSERT INTO "public"."dml_entity_dimensions" (entity_id,entity_field_name,dimension_name,
dimension_description,key_code) VALUES ((select id from dml_entity where entity_name =
 'Event2'),'int_flag','International Flag','International Flag (true/false)','I');
```

```
INSERT INTO "public"."dml_entity_dimensions" (entity_id,entity_field_name,dimension_name,
dimension_description,key_code) VALUES ((select id from dml_entity where entity_name =
 'Event2'),'trans_type','Transaction Type','Transaction Type','T');
```

2. Open a Korn shell on the Co>Operating System run host for the private project sandbox, and then change to the **$AI_RUN** directory.

3. Verify that there are no leftover recovery files from previous runs of the pset:

   ```
   ls -altr * upgrade_all_databases*.rec
   ```

   If there are any leftover recovery files, delete them before proceeding.

4. Run the pset:

   ```
   air sandbox run $AI_PSET/utility/create_environment/upgrade_all_databases.pset
   ```