Event Governance At Scale

A Practical Guide To Effective Event Message Design

Author: Timothy S. Hilgenberg <timhilco@gmail.com>

Table of Contents

Colophon . . . 1
Preface . . . 2
- CloudEvents.io Compatibility . . . 2
- What's in the Book? . . . 2

Chapter 1 . . . 4
- Why is Event Driven Architecture critical to tomorrow's applications? . . . 4
- Why are event design standards critical for organizational success? . . . 6

Chapter 2 - Message Definitions . . . 7
- Overview . . . 7
- Message Types . . . 7
- Event . . . 9
- Command . . . 11
- Audit (Proposed) . . . 12

Chapter 3 - Event Message Specifications . . . 13
- Overview . . . 13
- Event Types . . . 14
- Event Message Overview . . . 15
- Event JSON Structure . . . 18

Chapter 4 - Domain Event Examples - Consumer Business Events . . . 27
- Business Process State Change Event . . . 27
- Business Object Data Change Event . . . 30
- User Experience Action Event . . . 34
- Consumer Goal Event . . . 38

Chapter 5 - System Event Example - Runtime Operations Events . . . 40
- Platform Processing Event . . . 40
- System Resource Manager Event . . . 40
- Potential Future Operational Events . . . 41
- Operations Platform Process Event . . . 41

Chapter 6 Command Message Specifications . . . 44
- Overview . . . 44
- Command Types . . . 44
- Command Message Overview . . . 44
- Command JSON Structure . . . 45

Chapter 7 - Domain Command Example . . . 56

References 57
Appendix A: CloudEvents Data Attributes Comparison 58

Colophon

This book is a very early draft of this manuscript

This book is being written in AsciiDoc using Visual Studio Code.
I'm just learning this tool, so I apologize in advance for any bad formatting issues with early versions of the book.

Hilco is a mythical company

All comments welcome

Early Working Draft Version 0.9.5 — June 2022

Preface

This book defines a standard for asynchronous messaging for any company's Event Driven Architecture.
It is a language agnostic interface description for asynchronous message processing.
The specification is also intended to be message platform agnostic and does not specify how events can be delivered through various industry standard protocols (e.g. HTTP, AMQP, MQTT, SMTP), open-source protocols (e.g. Kafka, NATS), or platform/vendor specific protocols.
CloudEvents.io discusses these aspects of event processing in much better detail.
Although there are other serialization formats (binary, XML), the specification focuses on using JSON as it's primary format.
The goal within an organization is to have all messages published by any internal producer conform to these standards.
The specification also provides consumer of the messages with the detailed information they need to properly understand, consume and process these messages.

CloudEvents.io Compatibility

In general, this specification has the same philosophy as the CloudEvents specification.
It can be considered a supplement to the CloudEvents specification, extending it to support large enterprise organization Event Driven Architectures.
As stated above, it does not address the various protocols to deliver and process the message, but extending the semantics to support scale. (i.e number of event definitions within an organization)
Most of these attributes would be considered *extensions* in the CloudEvents specification and placed in that category.
(Ed. This statement needs to be validated by the group)

Also, the CloudEvents specification focuses on Events, where as this specification also includes Commands and Audit type messages.

See Appendix for more details on the mapping of the CloudEvents core attributes to this specification.

What's in the Book?

Chapter 1 - Why is Event Driven Architecture critical to tomorrow's applications?

Chapter 2 - What are the types of messages in Event Driven Architecture?

Chapter 3 - What are the Event Message Specification?

Chapter 4 - Domain Event Type Examples - Consumer Business Events

Note: The following chapters are still in an outline state.

Chapter 5 - System Event Type Examples - Runtime Operations Events

Chapter 6 - What are the Command Message Specification?

Chapter 7 - Domain Command Type Examples

Chapter 1

```
Events are everywhere, yet event publishers tend to describe events differently
  - CloudEvents.io Project <<cloudEvents>>
```

Why is Event Driven Architecture critical to tomorrow's applications?

Today, the primary interaction model for web based architectures to communicate to other systems is REST API via HTTP.
This interaction paradigm is called **Synchronous Request-Reply**, where the client is *waiting* for a response from the service.
This is driving the overall user experience, where the user presses a button or link and then waits for a response.
As more mobile applications are calling multiple backend services in a single request, this is causing longer wait times for the user as it takes time for all the requests to respond.
Ux Designers feel strongly that an synchronous response is the best user interaction model.

However, there are many application interaction models in practice today that do not fit this model.
People may not think of it in this way, but email and text are really disconnected (asynchronous) interaction models.
The person submits a communication message and then can perform other actions, while they wait for a notification that a response is available for review (e.g. waiting for an email response).
If the asynchronous request is fast enough, the user does not know the difference.
So there is precedence for user interface design using an asynchronous model.

So, why are interface designers not using a disconnected paradigm like email or text?
These types of interactions are very hard to design for.
There are some situations, where the user expects an immediate response.
It is hard in an asynchronous event driven model to simulate a synchronous response to the user.
It requires a very low latency interaction approach, which was hard to do with older technologies.
It was just easier to design a synchronous model when all the interactions were within the company computing ecosystem and there wasn't available technologies other than HTTP to provide an appropriate user response.

Event Driven Architectures are key to providing a rich satisfying user experience as applications become more complicated and more and mores services are being provided by

outside providers

A synchronous model also has a major impact on the computing resources required to support the application.
Instead of having to provision servers for maximum requests, that may sit idle, one can have fewer workers in the background continuously managing the work. An event driven architecture following an asynchronous approach is always a more efficient use of computing resources than a asynchronous approach.

Event Driven Architectures are key to a more efficient use of computer resources.
This is even more important is a cloud based environment where the pricing is based on actual detail usage.

Asynchronous technologies have been around for a very log time.
The IBM mainframe environment had MQ middleware that allowed inter-machine communication and intra-platform integration (primarily distributed platform).
There are even more modern technologies, like Kafka and RabbitMQ.
In prior application designs and even today, these technologies were used for inter-platform communication and more balanced resource allocation and management.
There were mostly used for underlying application interaction, but not used in an interface design.

The key to designing these type of applications is establishment of a set of event design standards and governance processes within the organization, along using asynchronous messaging technologies (IBM MQ, Kafka, RabbitMQ)
The purpose of this document is to provide a set of event specifications that can work at the scale of a large organization.

What is different in today's environment?

- Sophisticated application are leveraging more domain services that are being provided by parties who are outside of the control of the organization and the central application.
  This leads to more interactions and increases user experience wait time as the services are executed in a sequential manner.
- There are more modern messaging technologies are available to application architects to lower time time latency.
  There is also more practical real life experience in Event Driven Architecture for architects and designers to leverage
- User are becoming more comfortable with a disconnected interaction model, where they submit a request, work on a different task and then act on the response they get after the fact. Techniques, like deep linking from the response, to the specific page within the

application are becoming more prevalent.

- Real time analytics being processed by AI and Machine Learning are becoming value tools for decision making within an organization. Events are an important technique in making it successful.

As with any other scaled integration strategy, the key to success is standards and event design needs to follow this strategy.

Why are event design standards critical for organizational success?

As an organization's application portfolio continues to grow, so does it's integration requirements between these application.
In some cases, it is very difficult to create tightly integrated solutions.
In addition, users are expecting their outcomes to move faster and having their entire process completed in a single session.
Multiple applications are interested in the same events leading to increased complexity when using point to point approaches. Publish-subscribe approaches with events helps reduce this complexity.
Requirements for more integrations, faster deployment, reduced complexity and low latency integration is leading to more and more use of events and event driven architecture.

The design of **events** can come in all shapes and sizes.
They can be as low level as a single data element changing(changing one's email address), to an entire business process completing (completing a transfer in a bank account).
With the move to more domain oriented organizational structures, independent autonomous groups will be designing events.
This will cause a prolific number of events being created by an organization.
Without organizational design standards and guidelines, the lack of consistency will lead to chaos losing all the benefits of an Event Driven Architecture.

Event design standards are critical to the success of an Event Driven Architecture as the strategy is scaled up within the organization.
It isn't enough to commit to using events as a key integration strategy, the organization needs standards and governance in the design of events.
To support friction-less and over-reaching governance, the organization needs comprehensive design guidelines to support the event designers and give rubrics to the governance groups on how to judge the quality of the design.

Having common event standards are key to creating programming language libraries and tooling. The creation of these artifacts will lead to faster development times, increased quality and improve interoperability across application platforms.

Chapter 2 - Message Definitions

Overview

This chapter provides the key definitions of messages that could be supported within the organization's messaging ecosystem.
It will :

- define the types of messages and some of the guiding principles used to identify and name them.
- describe the potential fields definitions and formats for each of the message types.

Message Types

A message is a general-purpose data structure with no special intent.
In the integration world, these are typically just streamed between the systems for logging, searching, and for other operational and regulatory reasons.
As a component of the event architecture, there are 3 types of asynchronous messages:

Events

An **Event** is a message which informs various listeners about something which has happened in the past.
It is sent by a producer which doesn't know and doesn't care about the consumers of the event.
There can be multiple consumers of the event, each having their own interest in the event.
This type of messages promotes highly decoupled systems using pub/sub architectures.

An Event is an immutable record of a single event at a moment in time.

Commands

Commands trigger some action which should happen in the near future.
It's typically a one-to-one connection between a producer (who sends the command) and a consumer (who takes and executes the command) and, in a few cases, the order of commands is also of utmost importance.
Commands are usually performed by actors outside the current system.
However, commands can also be rejected, requiring new error handling patterns.

The difference in thinking between an event and a command is an event-X has occurred, rather than command-Y which should be executed as part of a conversation or interaction.

A Command is a request to retrieve some data or perform some action

Audit (Proposed)

An Audit message is an adhoc publishing of a domain business objects state.
There is no true triggering action, either from a business process or straight data change.
It would typically be triggered in batch fashion with a query predicate.
As part of normal processing, there will be situations where there are failures in the pipeline, which might lead to data inconsistencies between a systems of record and systems relying on this data.
In situations where the business objects are very stable and don't change often, the audit message is used to get to eventual data consistency between systems.
A full business object for a bounded context can be published on a periodic basis and then any consumer caches can be updated.
Thus, can also be used to seed new consumers with domain data.

Event

Something of interest that has happened in the past

An **Event** represents something that has happened, along the data context at a defined point in time.

Definitions:

- Result of some outcome
- Collins: a happening or occurrence, esp. when important
- CloudEvent Concepts
  - An event includes context and data about an occurrence. Each occurrence is uniquely identified by the data of the event.
  - Events represent facts and therefore do not include a destination, whereas messages convey intent, transporting data from a source to a given destination.

Event names should indicate a past tense action - a past-participle verb and should be action oriented.
Events are both a historical fact and a notification to other interested parties:

- Notification - a call for action, this is considered a stateless event
- A state transfer - pushing data wherever it is needed, known as an Event-Carried State Transfer

Events never produce a response object when published.
They could be the result of a business process (i.e. completed enrollment) or command (i.e. Change medical plan election).
They can't be rejected, but can be ignored.
There is no expectation of any future action by the publisher.

Events are immutable

Applications are like islands.
Inter application processing is becoming more and more important.
As applications become more specialized, business process will involve multiple services,requiring more integration to provide some specific aspect of the overall process experience.
Although it is not common practice, every application should consider publishing events as

part of it’s overall design and implementation.
Today, it is a bit of an afterthought.
Events and Commands should be first class elements of the application.
Even if there is no requirement for this in the present for an application, it will be at some point in the future.
Every quality application domain engine should provide API for request-reply processing and publish events for any downstream application.
A good event design will anticipate what events might be of interest and publish them.

Command

Represents a request to perform an important action task or retrieve data

Definitions

- Webster Definition: *to direct authoritatively*
- Represents an intent to perform some sort of action

The requested action of the command has not yet happened
(e.g. Change medical plan election). As opposed to events, where some action has already occurred.
The commands should be named with an imperative verb.
It has an explicit expectation that something (a state change or side effect) will happen in the future.
Commands can be rejected, but it these cases it is important for any consumer to response with some form of message.
Typically, this is part of a Ux conversation or program-to-program interaction.
However, there can one-way requests or fire-forgot messages with no response.

Commands can be used in situations where an async request/reply is desired.
However, this is not to be confused with a synchronous Request/reply like REST API.
The message can provide a call back information, which can be used in an asynchronous conversation with another consumer.

Audit (Proposed)

Represents the current state of a business object - Published on specific schedule

Audit messages can be used to synchronous data between systems of records and any consumer who is dependent on that data

More details to come.

Chapter 3 - Event Message Specifications

Overview

The purpose of this chapter is to define the overall design specifications for Events.
The event definitions format is JSON based and the chapter will provide an inventory of suggested fields, their attributes and their definitions.
It forms the foundations for the definitions of organization's event design standards.
The chapter will provide suggestions for various event types, header and body definitions and detailed field specifications.

The key features of the Event Message Specification are:

Provides a Unique and Global Message Identifier

Each message needs to have a unique and global identifier.
There should be no duplicate messages within the event ecosystem.
A tenant of the organization's overall event architecture is there are no duplicate messages.
In addition, it contains important auditing data: date, time, as to when the message was created.
This is important for any storage of the message in a data base and for logging and auditing purposes.

Provides provenance for the message, which includes a history of where the logical message has been processed

It answers the questions:

- Where and When did the event happen
- Who the source/publisher is, both system of record and publishing platform
- Who or What was the caused of the event

Strives to be as independent, stand alone and self-contained as much as possible.

The message should try to contain all the information a consumer might need to deal with the event.
The goal being to avoid complex and time consuming look ups for any consumer requiring additional context to process the message.

Provides simple headers and metadata to facilitate routing and filtering within the event processing network

In any complex environment, there will be a need to simply route the message to one or more message brokers and consumers.
The goal of the main header is to provide a set of standards to facilitate the creation of libraries and tooling.

This will lead to easier creation of message brokers leading to more efficient use of the networking infrastructure.

Supports schema version control and message validation

Event are not stagnant.
They will continually evolve, which means version control should be a first class element within the message.
Events in general are immutable, so changing them is basically impossible.
This makes version identification extremely important in publishing and consumer processing.
The use of dataSchemas for event type headers and body allow for the validation of a message.
This allows for the easy storage of the specifications in a dictionary or directory.

Provides the ability to store and find the key of the key business object or subject in a system of record

In general, every event should be associated with a key independent domain business entity (For HR/Payroll, this would be a person).
The message should store the primary key, name and type of the business object.
This would facilitate the retrieval of the domain object if needed.
In addition to the primary business object, there can also be related business objects that were part of the context when the event was published.
The specifications allows for this information to be part of the event.
It too, can be used to retrieve the data about this business entity.
If helpful, it can provided correlation for cross event processing.

Provides the ability to submit test or synthetic event for testing

Sometimes during testing, it is helpful not to perform the action the consumer wants to take based on the event it receives.
This field allows the consumer to identify this and not take any additional action, like calling a update API or updating a data base.

Contains metadata about the event which helps in stream processing

This information helps to route the request to the proper stream node and to facilitate proper processing of the event by a component within the stream topology.

Event Types

The follow sections provide the specification for the types of event support by the architecture.
The event types are:

Business Consumer/Application User

- Business Process (a.k.a Workflow) State Change Event
- Business Object Data Change Event
- User Experience Action Event
- Generic Goal Event

Runtime

- Platform Processing Event

Event Message Overview

The standard event message is a JSON document containing one main JSON object named **message**.
The **message** JSON object contains 3 child JSON objects: +

header	Common Message Header - Common across all messages, independent of event type and event
Event Type Header	Common header for specific event types. The name based on event type.
body	Event message body or event context data. This is a free form structure or JSON object (i.e. schema would vary by event) for each event defined with the event type

The two headers contain the metadata about the event: A standard global message header and a event type standard header.
The body contains the actual data related to the event at the time of the occupance.
The data schema for the body is determined by the event designer.

Common Message Header

The Common Message Header provides the following key features:

Global Message Metadata

The Global Message Metadata contains key information about the message, a unique, global message identifier, the type of message, creation timestamp, original publisher and history of consumer processors.
Any message defined within the ecosystem MUST contain these fields.
This information is common to all events and commands.

Event Type MetaData

The Event Type Metadata contains key information about the event type.

The event type is a attempt to create more standardization by observing that events can fall into certain categories or types.
Adding this level only increases the ability to take advantage of standard and has the same impact as the Global Metadata.
This includes the type of event and the DataSchema of the type to support automated access to the schema definition of the event type and body of the event.

Event Context

The event context are the key fields in the context when the event was published.
It includes a context label or tag, along with the action (past tense) that occurred at the time of the event.
This supports any routing of the event to other consumers and is a key analytics processing.
In general, events are processed against business domain objects.
The event context provides the fields for the retrieval of the main subject business entities and any additional related resources involved at the time of publication.

Audit History/Chain of Custody

To support debugging and auditing, the message contains information around who was the original publisher of the message, a history of processors that have touched the message.
In addition, it documents the System of Record for the key subject of the message.

Common Header Details

Every message type - event, command or audit - will have a common standard message header.
There will only be one format or schema for the common message header and the object is required.

The name of the JSON object is **header**.

It contains fields that describe the message at the highest levels and it identifies the source and type of the message. These fields determine the format and names of the fields that follow in the message object.
Since this is JSON, routing or filtering (e.g message brokers) can use only the header to determine routing of message or if the consumer is interested in processing the message. This provides a high degree of standardization, which leads to excellent tooling.

Event Type Header

The Event Type Header is a second level header that contains the common elements for all messages of a given event type.

The name of the JSON object is based on the name of the event type.

Each event type will have its own header name and structure. Examples:

- uxEventHeader - for Ux action events
- bpEventHeader - for business process state change events
- boEventHeader - for business object state change events

The messageDataSchema field in the header will indicate which event type header is in the message.
There will be a structured format/schema for each event type.
For an organization, the goal is a small bounded list of event types.
There can be an unlimited number of event definitions within a type.
The goal is to have as much standardization in the headers as possible.
The variations are meant for the **body** JSON object.

Event Message Body

The Event Message Body contains the actual data about the event.
This is the context at the time of creation.

The name of the JSON object is **body**.

These are fields that are specific to a given event definition within an event type.
The goal is to make the event as self-describing as possible.
Trying to avoid additional data retrievals to process the message.
Since most applications have a large unbounded set of events, the body represents the specific fields for a given event.
The above headers are intended to be standard, but the body is where the specific fields for that event are stored.
Each body should have it’s own schema that can be placed in a schema repository and retrieved by the bodyDataSchema field.
The schema can then be used for validation and code generation.
The eventBodyDataSchema in the Event Type Header will describe the schema for the fields in the body.
There are situations where a consumer might be interested in a change within a business object.
In this case, the body can contain both a before and after image or a list of changes fields with the old and new values.

This information can only be observed at the time of the event.

Event JSON Structure

In order to keep the processing of a message simple and easy to produce and consume, the event message has a very flexible structure and is basically an unstructured document.
The goal is to have a schema for the header, each event type header and every event data (i.e body) itself.
The desire is to have a schema dictionary which has a JSON or AVRO schema as it values and it's keyed by some name. The hierarchy is as follows:

- There is only one header schema (key name: header)
- To determine the <eventTypeheader> schema definition name, the messageDataSchema field contains the name of the event type
- To determine the body schema definition, eventBodyDataSchema field determine the name for the body schema

The event structure looks as follows:

{"message" :
"header" : { ... },
"*eventTypeHeader*" : { ... },
"body" : { ... }
}

Samples

{"message" :
"header" : {
"messageDataSchema": "com.hilco.messages/uxEvent",
"eventName" : "PageABC:clicked",
... },
"uxEventHeader" : {
... },
"body" : { ... }
}

{"message" :
"header" : {
"messageDataSchema": "com.hilco.messages/bpEvent",
"eventName" : "ContributionRateChange:Completed"
... },
"bpEventHeader" : {

```
... },
"body" : { ... }
}
```

Common Message Header Field Specification

Ed: Need to align these names with the CloudEvent name. Need to consider shorting some of the names (messageId → id) or using some of their names

Table 1. Schema Fields Table

Field Name	Attributes
messageId	String; Required
messageType	String; Required
messageDataSchema	URI (String); Required
messageVersion	String; Required
messageTopic	String
eventName	String
eventBodyDataSchema	URI (String)
contextTag	String; Required
action	String; Required
messageTimestamp	String; Required
businessDomain	String; Required
correlationId	String; Required
correlationIdType	String; Required
subject	String
publisherId	String; Required
publisherApplicationName	String; Required
publisherApplicationInstanceId	String
publishingPlatformsHistory	Object; Array; Required
- publisherId	String; Required
- publisherApplicationName	String; Required
- publisherApplicationInstanceId	String
- messageId	String; Required; Required

Field Name	Attributes
- messageTopic	String; Required
- eventName	String; Required
- messageTimestamp	String; Required
- sequenceNumber	String
subjectSystemOfRecord	Object; Array; Optional
- systemOfRecordSystemId	String; Required
- systemOfRecordApplicationName	String; Required
- systemOfRecordApplicationInstanceId	String
- systemOfRecordDatabaseSchema	String
- platformInternalId	String; Required
- platformExternalId	String
correlatedResources	Object; Array; Optional
- correlatedResourceType	String
- correlatedResourceIdentifier	String
- correlatedResourceState	String
- correlatedResourceDescription	String
isSyntheticEvent	String

Schema Field Definitions

messageId — Globally Unique Identifier of message.
The messageId is expected to be unique from a global perspective, so it is recommended to use some form of a GUID or UUID for this value.
It is not recommended that this value have any additional sematic value or meaning beyond uniqueness.

messageType — Describes the type of message.
Valid Values:

- Event

messageDataSchema

messageDataSchema is used to distinguish between the different types of messages (events or commands), source (internal vs external), and schema versions to avoid collision and help in processing the messages.
They also identify the type of Event Header contained in the full message.
The dataSchema can be used as an external endpoint to provide the schema and other machine-readable information for the event type and the latest major version.
Used to provide message definition and validation.
Example Values:

- com.hilco.messages/events/uxEvent
- com.hilco.messages/events/businessProcessEvent
- com.hilco.messages/events/dataChangeEvent
- com.hilco.messages/events/goalEvent
- com.hilco.messages/events/platformProcessingEvent

messageVersion

Conveys the version number (major.minor) of the message, and describes the structure of the overall message at hand.
Recommendation is to use semantic versions based on breaking changes.
Valid values managed by governance

- Example: 1.1

messageTopic

Logical name to describe the type of event. Note: this is not the physical topic name (i.e kafka topic) of the messaging system.
Sample Valid Values:

- BusinessProcess
- DomainDataChange
- UserExperience
- Goal
- PlatformProcess

eventName	Provides a standard name of the actual event that occurred in the publishing system. It will be treated as a label/code and used for filtering, routing, general analytics and simple processing of events in the ecosystem. It should be a combination of the business object or process name and action taken on that entity. There are specific naming conventions used to determine the value of the field. It is a field that will require governance approval.
eventBodyDataSchema	Describes the specific schema and version of the **body** field in the message. The body structure and metadata details are understood based on this name. This field is optional and only be set if there is a structure or schema for the body. If there is no body, then this field should not be sent.
contextTag	Machine readable generic label for the event type. The purpose of the contextTag is to provide a label that encoded some additional context for the event. It is highly structured, follows a specific format and provides valid values to allow programs and applications, like analytics, to easily consume the values. See event type for more details on the values. To reduce the complexity in trying to capture all the levels and details of components that produced the event, the recommendation is to encode all contextual or hierarchical information into a single label or tag. This tag along with the **action** field should reduce the complexity of the event structure and make it easier for the consuming tools to do their work without having to get into the details of the body structure. To make it more human readable, there will be an encoding standard in place to mke it easier to read and make it easier to parse the tag if necessary.

action	Represents the actual logical action or happening based on the event type. See event type for more details on the valid values. For events,the action should be described in the past tense and the name should be initial caps. For commands, the action should be present tense with initial cap. The organization should have a bounded set of actions and try to minimize the number.
messageTimestamp	Describes the date and time at which the actual event was generated by publishing systems. To be provided by producer component and should not be derived by message publishing framework(s) or component(s). The timestamp must be in the RFC 3339/ISO 8601 date format standard. See Appendix for details.
businessDomain	Describes the business domain under which the event/command was generated. Sample Valid Values in HR/Benefits: • Person • Worker • PersonWorker • Health • DefinedContribution • DefinedBenefit • Operations • N/A (for domains that do not match up to an organization service domains.
correlationId	Provide a globally unique identifier (UUID) to tie multiple events to the occurrence. Typically generated within the publishing application. This is used to correlate multiple messages across a logical process. The messageId is unique for the individual message, but the correlationId can be repeated across multiple messages

correlationIdType Describes the type of correlation identifier.
Suggested Values:

- SessionId - for participant Ux actions and sessions
- BatchId - for batch processing jobs. This is the actual instance id of a job type.
- PublisherCorrelationId - for publisher specific correction type (Typically used if the above two does not apply)

subjectIdentifier

Describes the global identity of the business subject being acted upon. The 'subject' is typically a key business domain object.
In the HR/Benefits domain, an example would be the person.

publisherId

Identifies the name or id of the publishing company who created the message.

publisherApplicationName

Describes the name of the publisher application platform or service.

publisherApplicationInstanceId

Describes the specific instance of the publisher application or service.

publishingPlatformsHistory

This is the historic details and providence of the message- *the audit trail for the message*.
It is an array, describing the internal platforms that have been processing a given logical message from the edge platforms to any internal consumer applications.
If the consumed message is being augmented (i.e new information is being added) is is important that the consumer/publisher or program add its own auditing information to the history. It has similar fields to the overall message (see above).

publisherId

Identifies the publishing company entity of the message.

publisherApplicationName

Describes the name of the publisher application platform or service

publisherApplicationInstanceId

Describes the specific instance of the publisher application or service.

messageId

Describes the messageId for the given prior message instance. See above for field details

messageTopic

Describes the messageTopic for the given prior message instance. See above for field details

eventName

Describes the eventName for the given prior message instance. See above for field details

messageTimestamp

Describes the messageTimestamp for the given prior message instance. See above for field details

sequenceNumber

The sequence should be from earliest to latest in chronological order.
The publisher should only append to the array If the array is provided as input from a message, then the new publisher should increase the sequence number and append the consumed/input header data to the array.
If this is the originating or edge processor, then the sequence number should be set to one (1), not zero

subjectSystemOfRecord

System of Record containing details related to finding the related subject or domain business object.

systemOfRecordSystemId

Identifies the system of record company entity of the message. Sometimes referred to as the partner ID.

systemOfRecordApplicationName

Describes the name of the publisher application platform or service.

systemOfRecordApplicationInstanceId

Describes the specific instance of the system of record containing the person

systemOfRecordDatabaseSchema

Describes the database schema instance of the system of record containing the business object

platformInternalId

Describes the internal identity of the business object within the platform. Only provided if the publishing platform is a source system of record and not a pure publisher application

platformExternalId

Describes the external identity of the business object within the platform. Only provided if the publishing platform is a source system of record and not a pure publisher application

correlatedResources

Describes a list of the related resources also being being accessed during the processing creating the event.
These are key *bounded contexts* associated with the primary business entity during processing.

correlatedResourceType

Describes the type of the related resource.

correlatedResourceIdentifier

Identifies the primary key of related resource. This can be the external or internal unique identifier of the resource.

correlatedResourceState

Identifies the state or status of related resource at the time the event occurred.

correlatedResourceDescription

Description of related resource at the time the event occurred.

isSyntheticEvent

Is this a synthetic or fake event? If true, assumes this is an event that should be processed under special circumstance, meaning don't change state or issue commands. Used for testing/monitoring in production by sending in fake events

Potential Extensions

dataContentType	This will be helpful if the body is not JSON. The current best practice is that all body payloads, should be JSON. The values would follow HTTP mime types

Chapter 4 - Domain Event Examples - Consumer Business Events

This chapter discuss the following consumer related events:

- Business Process State Changes
- Business Domain Object Data changes
- Consumer Ux Interactions
- Consumer Activities

All of these event would use the Event Header described in the prior chapter.

Business Process State Change Event

The purpose of this event type is capture events related to the processing of business process events.
These events are sometimes known as workflow or business transaction and are used to manage the changes in business domain objects.
Upon the completion of a business process, the process will update a business object. which will typically also create a Business Object Data Change event.

Event can be generated from two types of business processes.

1) A *long running state machine based* process that has many wait states and has time gaps in between the actions of the business process.

2) A *straight through* process or single task, where there are no waits in the process and the action is completed in a single unit of work. Typically, it is all or nothing from a commit standpoint.

Business Process examples:

- Contribution Rate Change
- Annual Enrollment
- Defined Benefit Retirement

State changes includes:

- Start
- Complete
- Valid

- In Error

Business Process State Change Event Type Header Specifications

JSON Name for *Event Type Header*: bpEventHeader

businessProcessReferenceId	Describes the primary key or Business Process Reference Identifier of the business process person instance as described in platform(s).
businessProcessId	Describes the internal identifier of the business process. It is the template used to create the person specific instances of the business process.
rawBusinessProcessName	Describes the pre-normalized Business Process Name as described in platform(s). This could be the client specific business process name or a normalized name across clients. It would be used primarily for reporting, filtering and aggregating
businessProcessDescription	Describes the long more formal description of the business process.
businessProcessStatus	Business Process status.Valid Values: • Created • Valid • Invalid • Completed • Canceled
businessProcessEffectiveDate	Effective date of the business process
businessProcessChangeTimestamp	Timestamp of when the business process was changed. The timestamp must be in the ISO 8601 date format standard.

EventName Standards

For the eventName, the standard will be the following fields separated by a colon (":") in camel case.

- *tag* which represents the business process name, hopefully normalized and
- *action* which represents the business process action

Tag Definition

The tag represents the business process name. It should be the normalized version of the business process name.

Format:

- Free format single alpha numeric value
- Free format double alphaNo formal specification is defined

Action Definition

The action represents the types of actions that result from the change in status of the business process during the processing of the consumer's business process.

Action Component Valid Values:

- Started
- Updated
- Completed
- Canceled

Body Definition Considerations

The body section is named **body**. The **body** can be any valid JSON schema

Business Object Data Change Event

The purpose of this event type is to capture the changes to key domain business objects.

Business Objects include:

- Person
- Person 401k Benefits
- Person Medical Benefits
- Person Document

Data actions include:

- Creation
- Updates
- Deletion
- Master Data Management Document Merge/Split

Business Objects Data Change Event Type Header Specifications

JSON Name for *Event Type Header*: boEventHeader

businessObjectResourceType Describes the primary domain data object type that was changed. Valid Values:

- person
- personDefinedContribution
- personHealthManagement
- personDefinedBenefit
- personDefinedBenefitCalculation
- personDocument

Editor:Think about moving this to 'tag'. Need to determine in the Identifier is included in the tag

businessObjectIdentifier

Provides the primary domain data object key of the business object that was changed.

additionalBusinessObjectResource

Provides any additional resource type and key to help further identify the component that changed.
This is similar to the path (../resource/{id}) in a REST URL

additionalBusinessObjectResourceType

Additional resource type

additionalBusinessObjectResourceId

Additional resource identifier or primary key

dataChangeTimestamp

Timestamp of the data change in the source platform.
The timestamp must be in the RFC 3339/ISO 8601 date format standard.
See Appendix for details.

EventName Standards

For the eventName, the standard will be the following fields separated by a colon (":") in camel case.

- *tag* which represents the business object name and
- *action* which represents the CRUD operation taken against the business object

Tag Definition

The tag represents the business object name.
Editor Note: Should tag replace 'businessObjectResourceType' .

Format:
* Free format single alpha numeric value
* No formal specification is defined

Action Definition

The action defines the type of data maintenance (CRUD) action taken on the business object.
Editor Note: action is replacing the dataAction field in prior versions.

- Action Component Valid Values

  dataAction

  Describes the data change or CRUD action performed on business object.- Create, Update, Delete.
  Also includes an primary key changes and Master Data Management (MDM) document merging.

- Create

- Update
- Delete
- MdmDocumentMerge
- MdmDocumentSplit

Body Definition Considerations

- The body section is named 'body'
  - **body** can be any valid JSON schema
  - Contains one predefined element 'extension'
  - Extension is a private area that can contain its own schema
  - The field is an map/array with:
    - Namespace as a key and,
    - Any valid JSON schema as its value

Data Fields Best Practices by Data Action

Update The recommendation for data fields to report is to provide only the fields that changed providing both old and new Best practice recommendations:

- PII
  - Fields: Bank/Credit Account Numbers,
  - Provide old/new unchanged from CustomerMaster; no masking required
- Arrays
  - Provides Lowest Level Detail field, include all cascading keys
  - Example: Contact → streetAddress → { AddrID → OldZipcode, newZipcode }
  - Include all the fields at the same level as the changed field in entire array data object
  - For fields in a high level/hierarchy, include all keys and simple primitive types (strings, numbers,etc) at the same hierarchy
- Do not include objects or arrays in the higher levels Do not include non-changing arrays at the same level

Create Provide the entire New document.
The alternate is too only provide foreign keys, which can be used to retrieve data from a data base.

Delete Only provide a delete event if the entire document is being deleted, not if one of the source systems deleted a person.
In the body, provide the primary document key (UniversalId or Mongo _id) and any IdMapping table
If the object/person is being delete in a given platform, but the person still exists in another platform, treat as an Update.
Only delete when no more IdMappings exist in the document

Master Data Management Platforms/CustomerMaster

Merge

- Treat as an MDM Merge Update event with two sections of data, one for survivor and one for deleted
- Both sections
- Survivor _id & Deleted _id
- Id Mapping for both survivor and deleted
- Survivor document section contains the update record for the survivor document (see Update section)
- Deleted document section
- Reason for merge
- The Platform that caused the change to occur
  - System Instance
  - Merge Field Change (old, new)

Split No new events, just two new event being generated
Web service call to deletePersonId service, which cleans up IdMapping and domain sections.
Generates a Normal Update event.
Web Service call refreshPersonForInternalId service, which causes a refresh through .

Ingest Generates a Normal Update event

User Experience Action Event

Events related to the behavioral actions taken by the participant in our user experience channels.
Channel include web, mobile, IVA/chat and other future user devices like Voice Assistants.

The purpose of this event type is to capture the pure behavioral events related to the interactions of the users in the channels - displaying pages, clicking button or links.
These events are not the result of any business process or data change events.

They are used for:

- Behavior actions for data reporting and analytics
- Provide notifications to non-domain processes (document management, campaigns) to drive their underlying processes

Actions may include, but not limited to:

- Button clicks
- Link or action selections
- Page or screen displays
- Hover
- IVA or chat intents

User Experience Action Event Type Header Specifications

JSON Name for *Event Type Header*: uxEventHeader

channel	Describes the channel (or UI application) where the event generated.
userDevice	Identifies the device used by end-user.
deviceTimestamp	Represents the timestamp on the device (May be different from the publisher timestamp). The timestamp must be in the RFC 3339/ISO 8601 date format standard. See Appendix for details.

sessionId	Represents the unique session of end user on our channels.
sessionCreateTimestamp	Session created time. The timestamp must be in the RFC 3339/ISO 8601 date format standard. See Appendix for details.
applicationName	User Experience application name
applicationVersion	Version of the application

EventName Standards

For the eventName, the standard will be the following fields separated by a colon (":") in camel case.

- UxControlName
- UserAction

Tag Definition

In the Ux channels, there are an unbounded set of device actions a user can take: pressing buttons, displaying pages, starting process flows.
In addition, they are an unbounded set of specific controls (buttons, etc) throughout the interface.
For reporting and other activities, there is a need to capture that a specific control has been acted upon: pressing a specific button within a specific group of controls within a page within a business process flow.

To reduce the complexity in trying to capture all the level and types of components, we are going to encode all hierarchical information into a single label or tag.
This tag along with the user action on this tag should reduce the complexity of the event structure and make it easier for the consuming tools to do their work.

To make it more human readable, there will be an encoding standard to make it more human readable and make it easier to parse the tag if necessary. The tag values need to take into account all types of user interfaces and devices.
We need to support new and emerging interfaces beyond web and mobile channels. The following sections discuss the naming approach.

Tag Component Valid Values

Web Channel	• Flow - A user's perceived outcome process or unit of work; Denotes flow of interaction (pages) or conversation between user and system ◦ Page ◦ Widget or Multiple Control Component • Elemental Ux Control ◦ Button, includes clickable icons - Clickable ◦ Link - Clickable ◦ CheckBox - Selectable ◦ Text - Display, Hover, Table Element ◦ TextBox - Keyboard Actions → Tabbing ,Enter pressed ◦ Bounded Lists → Radio Buttons or checkboxes or DropDown Lists or Dials - Selectable
Mobile	TBD
Smart Assistant/AlexaIVA/Chat	TBD
Other on Non-Channel	Treatment or Theme Example xxxA/xxxB

Format

- Ordered sets of tuples separated by underscore '_'
- The tuple is the following fields separated by dash '-'
  - LogicalName determined by Ux Designer and Data Analyst
  - UxControl Valid Value in all caps
- The order is from highest level (aFlow) to specific UX Control, (Button)

Example: <Flow_Name>-FLOW_<Page_Name>-PAGE or Retirement-FLOW_HubPage-PAGE

Action Definition

The action defines the type of user actions taken by the user when interacting with the channel/device.
Valid Values for userAction:

- Displayed
- Clicked

- Entered

Body Definition Considerations

- The body section is named **body**
  - **body** can be any valid JSON schema
  - Contains one predefined element **extension**
    - Extension is a private area that can contain its own schema
    - The field is an map/array with:
    - Namespace as a key and,
  - Any valid JSON schema as its value
- This can be any significant data or data of interest for reporting at the time of the UX Event

Consumer Goal Event

Events related to the action taken by the consumer in the context of reaching a personal goal.

A goal is non-transactional outcome the consumer is trying to attain.
For example, the person wants to lose 20lbs as a health goal

Actions may include:
* Started
* Completed

Consumer Goal Event Type Header Specification

The Personal goal only requires the main header
JSON Name for *Event Type Header*: pgEventHeader

Tag Definition

The tag represents the name of the personal goal in a machine readable format.

Format:
* Free format single alpha numeric value
* No formal specification is defined

Action Definition

The action defines the type of task actions taken against a personal goal.

Action Component Valid Values :
* Started
* Completed

Body Definition Considerations

- The body section is named **body**
- body can be any valid JSON schema
  - Contains one predefined element **extension**
  - Extension is a private area that can contain its own schema
    - The field is an map/array with:
    - Namespace as a key and,
    - Any valid JSON schema as its value
  - This can be any significant data or data of interest for reporting at the time of the UX

Event

Chapter 5 - System Event Example - Runtime Operations Events

Events occurring during the running of the application on a specific platform or system resource (server, data base, etc).
Focused mostly on system resource issues. Debug events would also fall into this category

Platform Processing Event

Events related to the action of completing a discreet process or unit of work and providing the resource computation of that unit of work.

These events reflect the fact that:

- the application process occurred, which is used for counting instances of the process
- the time stamps of then it occurred, which is used for elapse time and windows of time,
- the time stamp of the additional resource usage, which is used for more detailed resource consumption analytics and
- any additional status and metadata related to the process of the unit of work. They event can be used for both operational and application event where counting, and resource utilization reporting is of interest to the business

Platform processing units of work being metered include:

- Docker Containers → Service API calls
- Enterprise CI/CD Build pipeline → API Service builds

System Resource Manager Event

- Under consideration ** +

Events related to the actions of managing a system resource manager. This includes the overall operations of the resource manager and the system administrator actions to administer the resource manager.

A resource manager has the following characteristics. Example of a resource manager is a server, JVM, Web Server, Docker container, data base manager and queue manager.
* The platform contains important application data
* The resource is shared by multiple applications
* The resource is managed centrally by an system operations staff, who is responsible for the installation, upgrade and overall operations of the resource

* Privileged access authority is required to perform system administration functions

Resource manager operational event actions include:

- Start
- Stop
- Abort
- Restart

System Administrator event actions include:

- Logon
- Password change
- Commands (?)
- Group Functions (?)

Potential Future Operational Events

- Runtime System Error Events - Runtime Events because of hardware or software issues
- Code Deployment Events - DevOps Events because of program/business logic development. Include both code and configuration
- Client Deployment Events - Events related to the deployment of client assets, in particular provision migrations.
  Might also include client level processing runtime errors

Operations Platform Process Event

Event Header

See the above header section for definition of the event message header

Platform Process Event Header

JSON Name: oppEventHeader

platformProcessStartTimestamp	Timestamp of when the resource consumption started. The timestamp must be in the RFC 3339/ISO 8601 date format standard. See Appendix for details.

platformProcessEndTimestamp	Timestamp of when the resource consumption ended. The timestamp must be in the RFC 3339/ISO 8601 date format standard. See Appendix for details.
platformProcessElapsedTime	Elapse time in milliseconds
platformProcessEffectiveDate	Effective Date of process
platformProcessStatus	Process status. Valid Values: • Aborted • Completed • Canceled

Potential Platform Process Header Fields

- API/Program call Request and Response size
- Memory size at event publication
  - Heap sizes , etc
- CPU utilization for process
- Thread count at event publication

EventName Standards

For the eventName, the standard will be the following fields separated by a colon (':') in camel case.

- PlatformProcess
- Action

Tag Definition

- Format
  - Ordered sets of tuples separated by underscore '_'

Action Definition

The action defines the type of action or state changes of the process.

Action Component Valid Values

Process Action Valid Values:

- Started
- Completed

- StateChanged

Body Definition Considerations

- The body section is named 'body'
- 'body' can be any valid JSON schema
- Contains one predefined element 'extension'
  - Extension is a private area that can contain its own schema
  - The field is an map/array with:
  - Namespace as a key and,
  - Any valid JSON schema as its value
    This can be any significant data or data of interest for reporting at the time of the process state change.

Other Operational Events

Chapter 6 Command Message Specifications

Overview

Key features of Command Message Specification are:

- Provides Unique Message Identifier
- Provides auditing data, date, time, as to when the message was created
- Is Independent and self-contained
- Provides simple headers and metadata to facilitate routing and filtering within the message processing network
- Provides information on the requestor and callback action to the requestor for the message
- Supports schema version control and message validation
- Provide the ability to find a person in a system of record
    - Relates business object can be attached to the message
    - Message can provide a provider's client Id and any customer service agent data
- Provides the ability to submit test or synthetic command for testing

Command Types

The follow sections provide the specification for the types of commands support by the architecture. (Note: Some command types are in the prototype stage)
The command types are:

- Business Process Data Change Request
- Business Object Data Change Request
- Communication Composition/Delivery

Command Message Overview

The command message is a JSON document containing one JSON object named "message". The message object contains 2 JSON documents: a header and a free form body for each command.

- Message Header

    - Every message type, events or commands, will have a common standard header
    - There will only be one format or schema for the header and the object is required
    - The name of the JSON object is “header”
    - It contains fields that describe the message at the highest levels and it identifies the source and type of the message
- These fields determine the format and names of the fields that follow in the message object
    - Since this is JSON, routing or filtering consumers can use only the header to determine routing of message or if the consumer is interested in processing the message
- Message Body
    - Contains the fields that are specific to a given instance of an command type
        - The system will have a large unbounded set of commands. The body represents the specific fields for a given command
    - The commandBodyNamespace in the Command Type Header will describe the schema for the fields in the body
    - The name of the JSON object is “body”

Command JSON Structure

To keep it simple and easy to produce and consume, the command message has a very flexible structure and is basically an unstructured document. The goal is to have a schema for the header and every command data (i.e body) itself. We would like to have a schema dictionary which has a JSON or AVRO schema as it values and it’s keyed by some name. The hierarchy is as follows:

- There is only one header schema (key name: header)
- To determine the body schema name, the header.commandBodyNamespace field determine the name for the body schema

The internal command structure looks as follows:

{“message” :
“header” : { … },
“body” : { … }
}

The internal event structure looks as follows:

```
{"message" :
"header" : { ... },
"eventTypeHeader" : { ... },
"body" : { ... }
}
```

Command Message Header Field Specification

Table 2. Schema Fields Table

Field Name	Attributes
messageId	String; Required
messageType	String; Required
messageNamespace	String; Required
messageVersion	String; Required
messageTopic	String
eventName	String
eventBodyNamespace	String
contextTag	String; Required
action	String; Required
messageTimestamp	String; Required
businessDomain	String; Required
correlationId	String; Required
correlationIdType	String; Required
globalBusinessObjectIdentifier	String
publisherId	String; Required
publisherApplicationName	String; Required
publisherApplicationInstanceId	String
publishingPlatformsHistory	Object; Array; Required
- publisherId	String; Required
- publisherApplicationName	String; Required
- publisherApplicationInstanceId	String
- messageId	String; Required; Required
- messageTopic	String; Required

Field Name	Attributes
- eventName	String; Required
- messageTimestamp	String; Required
- sequenceNumber	String
businessObjectSystemOfRecord	Object; Array; Optional
- systemOfRecordSystemId	String; Required
- systemOfRecordApplicationName	String; Required
- systemOfRecordApplicationInstanceId	String
- systemOfRecordDatabaseSchema	String
- platformInternalId	String; Required
- platformExternalId	String
correlatedResources	Object; Array; Optional
- correlatedResourceType	String
- correlatedResourceIdentifier	String
- correlatedResourceState	String
- correlatedResourceDescription	String
isSyntheticEvent	String

Schema Field Definitions

messageId Global and Unique (UUID) Identifier of message.

messageType Describes the type of message.
Valid Values:

- Command

messageNamespace Namespace is used to distinguish between different types of messages (events vs commands), source (internal vs external), and schema versions to avoid collision and help in processing the messages.
The namespace can be used as an external endpoint to provide the schema and other machine-readable information for the event type and the latest major version. Used to provide message definition and validation
Valid Values:

- com.hilco.messages/commands/aCommand

messageVersion DescribesConveys the version number (major.minor) of the message, and describes the structure of the overall message at hand. Valid values managed by governance

- Example: 1.1

messageTopic String Logical name to describe the type of event. Note: this is not the physical topic name (i.e kafka topic) of the messaging system.
Sample Valid Values:

- BusinessProcess
- DomainDataChange
- UserExperience
- Goal
- PlatformProcess

For commands, this is an optional field. For events, it is required

messageSubTopic

Logical name to describe a second level categorization of event

commandName

Provides a standard name of the actual command that happened based on a user's behavior action.
It will be treated as a label/code and used for filtering, routing, general analytics and simple processing of commands in the ecosystem.
It should be a combination of the business process name and action taken on that process.
There are specific naming conventions used to determine the value of the field.
It is a field that will require governance approval.

commandBodyNamespace

Describes the specific schema and version of the body field structure of the command.
The body structure and metadata details are understood based on this combination.
This field is optional and only be set if there is a structure or schema for the body.
If there is not body, then this field should not be sent.

tag

Machine readable generic label for the command type. Its purpose is to provide a label that encoded some additional context for the command.
It is highly structured, follows a specific format and provides valid values to allow program and applications, like analytics, to easily consume the values. See command type for more details on the values.

To reduce the complexity in trying to capture all the level and types of components, we are going to encode all contextual or hierarchical information into a single label or tag.
This tag along with the user action on this tag should reduce the complexity of the command structure and make it easier for the consuming tools to do their work without having to get into the details of the body structure

To make it more human readable, there will be an encoding standard to make it more human readable and make it easier to parse the tag if necessary.

action

Represents the action being requested by the consumer on. See command type for more details on the valid values. For commands, the action should be described in the present tense and the name should be initial caps.

tagObjectId

Used to provide a separate identifier for the object of the tag. If the tag represents a general category and there are instances of that category that contain a key /identifier, this field can be used to provide the identifier.
The recommended best practice is to put the identifier in the tag itself.
This field, along with the generic tag value, provides an alternate to that approach

messageTimestamp

Describes the date and time at which the actual command was generated by publishing systems. To be provided by producer component and should not be derived by message publishing framework(s) or component(s).
The timestamp must be in the RFC 3339/ISO 8601 date format standard.
See Appendix for details.

messageCriticality

Valid Values:

- High
- Medium
- Low

messageExpiry

Number in seconds
Used to determine if the message is still valid to process.
The determination of whether this message should still be processed is set against the messageTimestamp.
If the current time is past the messageTimestamp plus this value, then the message should be ignored

businessDomain

Describes the business domain under which the event/command was generated.
Valid Values:

- Person
- Worker
- PersonWorker
- Health
- DefinedContribution
- DefinedBenefit
- Operations
- N/A (for domains that do not match up to our organization service domains.

correlationId

Describes the globally unique identifier (UUID) typically generated within the publishing application.
This is used to correlate multiple messages across a logical process.
The messageId is unique for the individual message, but the correlationId can be repeated across multiple messages.

correlationIdType

Describes the type of correlation identifier.
Valid Values:

- SessionId - for participant actions and sessions
- BatchId - for batch processing jobs. This is the actual instance id of a job type.
- PublisherCorrelationId - for publisher specific correction type (Typically used if the above two does not apply)

agentId

Identifies logged-in agent acting on the participants behalf

globalPersonIdentifier

Describes the global identity of the participant within hilco, in particular the UDP platform.
Required if source platform of record Ids are not present and the command is related to a participant.
Note: sometimes this is referred to as the universalId.

requestorId

Identifies the publishing company entity of the message.

requestorApplicationName

Describes the name of the requestor application platform or service. See Appendix for list of publishing applications.
See Appendix for list of recordkeeping systems

requestorApplicationInstanceId

Describes the specific instance of the requestor application or service.

messageHistory

Publishing Applications history and details. This is the history and providence of the message.
It is the array, describing the platforms that have been processing a given message from the edge platforms to any internal consumer applications.
This includes command processing or transformation applications and systems of record.
It provides an audit trail of the message thought it's lifecycle

publisherId

Identifies the publishing company entity of the message. Sometimes referred to as the partner ID. For internal requestors, it will be 'hilco'. For partners in the Partner Network, it will be a partner identifier.

publisherApplicationName

Describes the name of the requestor application platform or service. See Appendix for list of publishing applications.
See Appendix for list of recordkeeping systems

publisherApplicationInstanceId

Describes the specific instance of the requestor application or service.

messageId

Describes the messageId for the given prior message instance. See above for field details

messageType

Describes the type of message.
Valid Values:

- Event
- Command

messageTopic

Describes the messageTopic for the given prior message instance.
See above for field details

messageSubTopic

Describes the messageSubTopic for the given prior message instance.
See above for field details

commandName

Describes the commandName for the given prior message instance.
See above for field details

messageTimestamp

Describes the messageTimestamp for the given prior message instance.
See above for field details

sequenceNumber

The sequence should be from earliest to latest in chronological order.
The publisher should only append to the array if the array is provided as input from a message, then the new publisher should increase the sequence number and append the consumed/input header data to the array.
If this is the originating or edge processor, then the sequence number should be set to one (1), not zero

personIdentificationSystemOfRecord

System of Record containing details related to finding a person.
Required if globalPersonIdentifier is not present and the command is participant related.

systemOfRecordSystemId

Identifies the system of record company entity of the message.
Sometimes referred to as the partner ID. For internal publishers, it will be ‘hilco’.
For partners in the Partner Network, it will be a partner identifier.

systemOfRecordApplicationName

Describes the name of the publisher application platform or service. This section should contain the best system for person related data.
If that system is not available, then the publishing application should provide the best platform available.

systemOfRecordApplicationInstanceId

Describes the specific instance of the system of record containing the person

systemOfRecordDatabaseSchema

Describes the database schema instance of the system of record containing the person

platformInternalId

Describes the internal identity of the participant within the platform. Only provided if the publishing platform is a source system of record and not a pure publisher application

platformExternalId

Describes the external identity of the participant within the platform. Only provided if the publishing platform is a source system of record and not a pure publisher application

platformRoleType

TBA use only. If TBA is the source platform, a valid role type can be provided.

platformClientId

Describes the client Id in the publishing platform. This is a platform specific ClientID. The normalized ClientId is above

relatedResources

Describes a list of the related resources. These are key “bounded contexts’ associated with the primary business entity. This can be 'campaign' or 'business process' or some other resource related to the action performed by the end user.

relatedResourceType

Describes the type of the related resource.
Valid Values:

- PersonActivity
- Document
- Plan
- TbaTransaction

- Fund
- Account
- Address
- PersonDefinedBenefitCalculation
- Campaign & PersonCampaign

relatedResourceIdentifier

Identifies the primary key of related resource. This can be the external or internal unique identifier of the resource.

relatedResourceState

Identifies the state or status of related resource at the time the command occurred.

relatedResourceDescription

Description of related resource at the time the command occurred.

isSyntheticCommand

Is this a synthetic or fake command?
If true, assumes this is an command that should be processed under special circumstance, meaning don't change state or issue commands. Used for testing/monitoring in production by sending in fake commands

Potential Future Command Fields

consumerCallbackInstructions

HEADER <how to execute the callback>. This could be:

- An Id of a function or policy to execute
- Actual source code that can be interpreted and executed (DSL, Lambda

consumerCallbackInputs

<inputs unique to this callback logic> Array of name value pairs

consumerCallbackScript

<Actual scripting code/logic to execute which may update a database or call a rest service, etc…>

consumerCallbackCredentials

This could be:

- Token based → Short lived token and Expiration Date
- Functional UserID/Password → for internal use only
- SAML like approach

consumerCallbackErrorInstructions

HEADER <how to execute the callback>. This could be:

- An Id of a function or policy to execute

Actual source code that can be interpreted and executed (DSL, Lambda

consumerCallbackErrorInputs

<inputs unique to this callback logic> Array of name value pairs

consumerCallbackErrorScript

<Actual scripting code/logic to execute which may update a database or call a rest service, etc...>

queryParameters

BODY GET parameters command input

requestBody

BODY PUT/POST parameters command input

Chapter 7 - Domain Command Example

This chapter ....

References

- [cloudEvents] CloudEvents; CNCF Serverless Working Group *https://cloudevents.io*

Appendix A: CloudEvents Data Attributes Comparison

id → messageId
source → eventName and Namespace
specVersion → messageVersion
dataContentType → JSON only
dataSchema → messageNamespace, eventNamespace (body)
subject → Subject/Business Object
time → messageTimestamp +