Brightware™

Report Developer's Guide

Version 8.1.4

edocs

Trademark, Copyright, and Patent Acknowledgements

Version Date: November 24, 2004

Table of Contents

Chapter 1

About this Manual

Audience

If you are involved in using the Analytics report templates or modifying/creating templates, read this manual. It contains information about the Datamart (including specifics for e-mail and Web data), and a data model diagram.

Related Documentation

For more information about Analytics, please refer to these documents included in PDF format on the CD:

- *Analytics Overview Guide*
- *DB Administrator Guide*

Conventions

The following typographic conventions are used in this document:

- Items that you are instructed to click or select, such as button names and hyperlinks, are bold:
  - Select **Add Response**.
  - Click the **OK** button.
- Documents, headings, and chapter titles are italicized:
  - "Refer to the *Reference Manual* for more information."
- Notes are flagged along the left margin:

This icon indicates noteworthy information.

- Cautions are flagged along the left margin:

This icon indicates critical information.

- Programming code and system messages appear in a fixed-width font:

```
Set-request-condition (<condition>)
```

- Hyperlinks and Cross References - If viewing a document online, you can navigate through it using hyperlinks, which appear in blue text, and cross references. Although not displayed in blue, the Table of Contents and Index entries are also hyperlinks. Cross references are specific page number references. Click the page number to navigate to that page:
  - Refer to "Technical Support", on page 3.
- The term Type usually refers to typing information on your keyboard:
  - Type the number of decimal places you want displayed.
- The term Enter typically refers to the "Enter" key on your keyboard:
  - Type the number of decimal places you want displayed and press the **Enter** key.
- When a directory path is given, the hard drive letter is omitted since it is unknown what hard drive the system is installed on. Only the default install path is supported:
  - Documents are available under edocs\Brightware\docs\.

Technical Support

Assistance is available from edocs Technical Support.

	North America	Europe
Telephone	(508) 652-8400	+44.20.8956.2673
Hours of operation	8:30 AM – 8:00 PM Eastern Time, Monday - Friday	09:00 – 17:00 GMT Monday -Friday
E-mail address	support@edocs.com	support@edocs.com
Please consult your Warranty and Maintenance Attachment for the terms of your technical support.		

Before you call Technical Support, please have the following information available for the representative:

- Your company name.
- Version of software currently being used.
- Exact error message.
- Where the error occurred.
- Exact path for recreation of the error.

Crystal Reports

If you need assistance with Crystal Reports, navigate to http://support.crystaldecisions.com/. Be sure to have your Crystal Reports registration number available if you contact Crystal Decisions.

Chapter 2

General Datamart Information

Introduction

This chapter describes the Datamart data model. Information in this chapter, as well as the information in Chapter 3, “Email Datamart Specifics”, on page 13 and Chapter 4, “Web Datamart Specifics”, on page 55, prepares you to use and customize the reports of your choice. We assume you are familiar with database concepts and the Brightware application.

Datamart

The Datamart is a relational database which uses a *star schema* to define relationships between tables. The Datamart contains data necessary for analytic and reporting purposes. It does not contain data needed solely for email or Web operations.

The Datamart database is populated by running the ETL. Figure 2-1 shows the overall architecture.

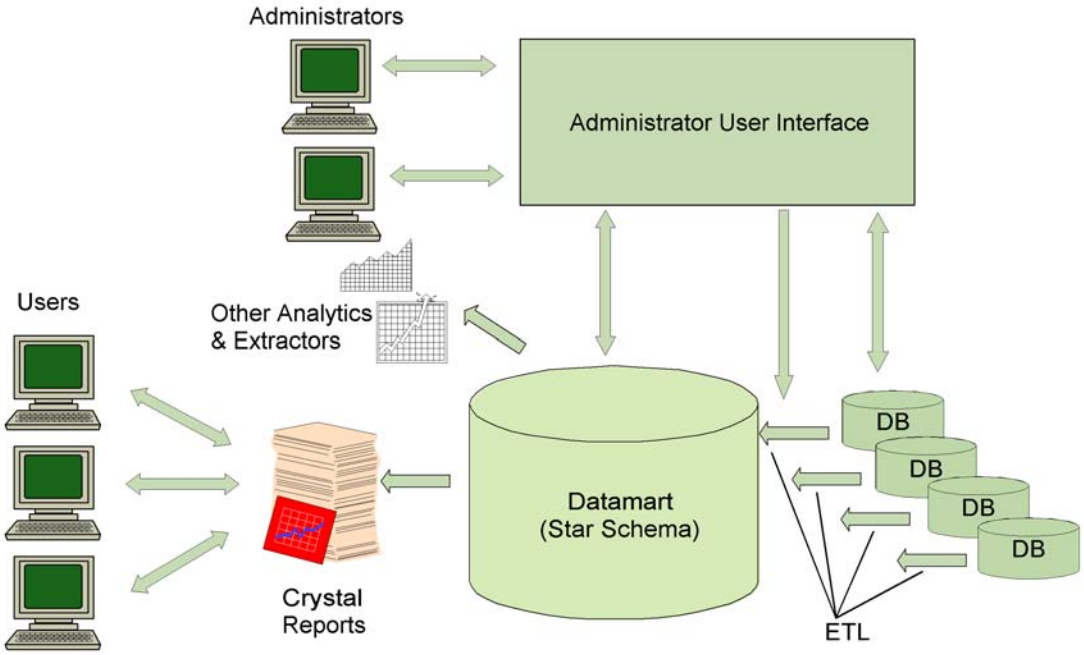

Figure 2-1. Analytics Architecture

The sections that follow describe the general information about the Datamart. Other components are discussed in separate documents.

ETL Tools

ETL, which stands for *extract*, *transform*, and *load*, is the process used to populate the Datamart. The ETL Tools are a Java-based application that extract relevant information from the Brightware operational databases. They transform the data into a representable form in a star schema, and load it into the Datamart. Other features of the ETL Tools include:

- **Incremental data transfer –** The ETL tools extract only data added to the transactional database since the last time it ran. This reduces the transfer time since the Datamart is not rebuilt each time.
- **Interruptible and restartable –** ETL events can be stopped and restarted any time. The ETL Tools track data transfers using a timestamp. When an ETL event is restarted it ignores previously transferred data and begins at the point at which it finished.
- **Scrubbing –** The ETL removes or transforms bad or ambiguous data that may reside in the transactional database from previous versions of the product.

Functional Overview

The Datamart contains four main types of tables:

Fact Tables

These tables track events. Most information is extracted from history tables maintained in the transactional database. There are six fact tables used by the Datamart for email and three fact tables used for Web. See "Email Fact Tables", on page 30 and "Web Fact Tables", on page 62 for additional information. Analytics uses two kinds of Fact tables:

- **Event Fact Tables** – These tables track events that occurred at a given point in time. Analytics has named the Event Fact tables as *somethingEvent* to indicate their content and to distinguish them from the Dimension tables.
- **Summary Fact Tables** – Each Summary Fact table is the center of another star in the schema used for the Web. However, it does not define an event directly. The Summary Fact table specifies a start and end time to define the period of time each summarization spans, and it points to other tables which describe the facts related to the event. Analytics has named Summary Fact tables as *somethingSummary.*

Dimension Tables

These tables, located at the points of the star in the schema, describe or quantify the events listed in the fact tables. Most Dimension tables are used in more that one Fact table. There are 13 dimension tables and 11 Group dimension tables (also known as Bridge tables) used for email in the Datamart and 15 Dimension tables and 9 Group Dimension tables used for Web. A dimension table may have more than one of the following attributes:

- **Group (Bridge) Dimension Tables** – The records in these tables identify the individual members of a group and point to other Dimension tables which define the individual member. Analytics named all Group tables as *somethingGroup*.

- **Fixed Dimension Tables** – These tables contain a fixed set of records predetermined by settings in the email or Web programs. The requestStatus table is a typical Fixed Dimension table. It identifies if a request is Open, Pended, or Closed in all email installations. The set of records cannot vary from installation to installation.
- **Varying Dimension Tables** – These tables contain a varying set of records. Because the records are defined by the configuration of your installation, they are not the same for all installations of email or Web. For example, the intent Dimension table is a typical Varying Dimension table. Its contents depend upon the definition of an intent in your Knowledge Base and will vary from Knowledge Base to Knowledge Base.
- **Entity Dimension Tables (Shared or Unshared) –** These tables identify an entity (such as a person) and typically include a number of attributes in the table which describe the entity (such as first name, last name, and birth date). Any fact record which uses the entity points to the applicable record in the entity dimension table where the information about the entity is recorded.

  A shared dimension is one that appears in more than one source database and is considered to be the same dimension. For example, an intent named "Buy a PC" in one source is assumed to be the same intent as the one named "Buy a PC" in another source and so are shared across sources. Only the following dimensions are shared in the Web tables:

  - intent
  - initialInputField
  - additionalInputField
  - customer

  The first three dimensions define names of things, an intent in a Knowledge Base and forms which allow the customer to make inquiries. The customer dimension is a bit different. In Analytics, a given email address refers to one and only one customer regardless of which source the email address came from, and in this way it is a shared dimension.

  All of the other entities used for Web are not shared. The fact that two entities in different sources share the same identifier (such as name) does not mean they refer to the same entity. A response named "Response 1" in source "A" may cause an escalation to email. In another source, a response named "Response 1" may route to the company's career page on its Website. They are not considered the same entity because response is not a shared entity and the items are from different database sources.
- **Attribute Dimension Tables** – Attribute dimension tables define attributes about a fact record. For example, an attribute dimension might be "firstName" and include records such as "Chip", "Ganesh", or "Sue". Any fact record which uses "firstName" would point to the applicable record in this Attribute Dimension table. It is important to understand that the name such as "Chip", "Ganesh", or "Sue" is recorded in the Attribute table. If there are two different people named "Chip", the fact records or dimensions using "firstName" will point to the same Attribute dimension record.

Note that it is also quite acceptable to point to an attribute dimension table from another dimension table. For example, a given entity dimension may use a "firstName" attribute table (as well as a "lastName" attribute table). Such configurations, where the fact table points to an entity dimension which, in turn, points to an attribute dimension table is considered a snowflake—a variation of a star.

See "Email Dimension Tables", on page 14 and "Web Dimension Tables", on page 66 for additional information.

- **Set dimension table** – Set dimension tables define sets of related entities. For example, an agent group (not to be confused with a Group table) can contain many agents and an agent may be contained in many agent groups. The agentGroupSet table defines those relationships. Such tables provide the ability to relate an entity from one fact record to another entity. In this example, a fact record referring to a particular agent can use the set table to determine the agent groups related to that fact record through the agent. Analytics has named set dimension tables as somethingSet.

Metadata Tables

These tables are used by the ETL Tools. They contain data that the tools use to detect changes to the transactional database. For example, when a new Agent or queue is added to the email, the ETL Tools need to recognize those changes and update the `agent` and `queue` dimension tables accordingly. Some metadata tables serve as staging areas for loading facts and dimensions. Metadata table names are prefixed by the word *meta*.

All these tables are considered owned by the ETL Tools and the DB ADMINISTRATOR user interface. Therefore, they must not be changed by other applications. Modifying them may jeopardize the integrity of your Datamart.

Administration Tables

These tables are used by the ETL Tools to maintain state information (for example, the status of the last ETL) and to log information for later analysis. Administration tables are prefixed by the word ETL.

All these tables are considered owned by the ETL Tools and the DB ADMINISTRATOR user interface. Therefore, they must not be changed by other applications. Modifying them may jeopardize the integrity of your Datamart.

Primary, Foreign, and Run Keys

Analytics uses Primary, Foreign, and Run keys to identify information or point to records in a table. A Primary key uniquely identifies a record in the current table. A Foreign key points to a primary key located in another Dimension table. The Run key identifies which ETL loaded the data.

The primary and foreign keys act as a link from one table to another. A schematic of the link from a foreign key in a fact table to a primary key in a dimension table resembles a star. Figure 2-2, “Star Schema”, on page 9 shows a simple star using the `requestEvent` fact table as the center. The four dimension tables (date, queue, customer, and agent) are the points of the star and describe or quantify the date, queue, customer, and agent respectively for the request event listed in the Fact table. Remember that the rows of the requestEvent Fact table define individual events that occurred during the life of a request.

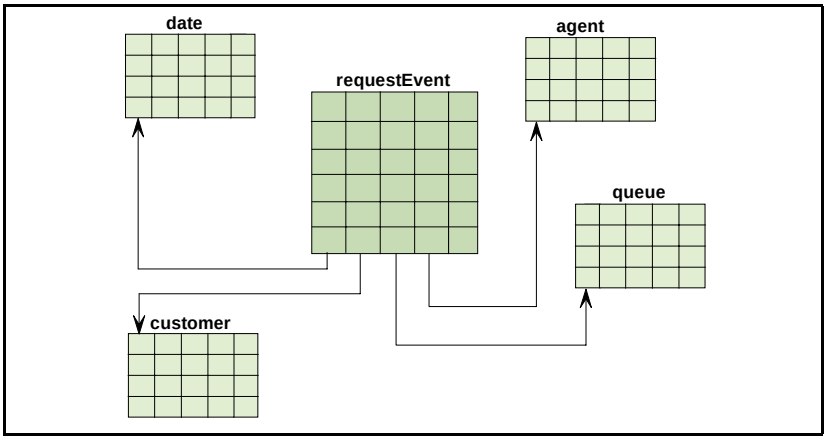

Figure 2-2. Star Schema

Figure 2-3 shows the relationship between the foreign keys of the `requestEvent` Fact table (the left four columns) to the primary keys of the referenced Dimension tables.

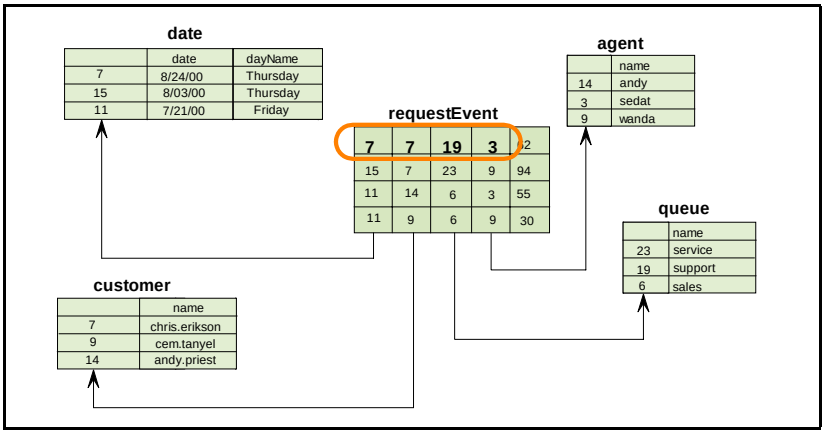

Figure 2-3. Star Schema Example

The circled foreign keys (shown in Figure 2-3) in the requestEvent Fact table point to the keys in the *date*, *customer*, *agent*, and *queue* Dimension tables. As you can see in this example, on date *8/24/00*, a message was received from customer *chris.erickson* which was assigned to the queue *support* and handled by the agent *sedat*.

There are other important facts to remember regarding primary and foreign keys:

- Most Fact table data are foreign keys that point to data in other tables. By itself, the value of the foreign key is not meaningful.
- The key values are arbitrary and subject to change from time to time. You cannot expect a key value of 7 to be assigned to the same date in another ETL.
- Furthermore, you cannot compare attributes by their key values. For example, if the key value for the date 8/24/00 is 7, you cannot assume that the key value of 8 for date refers to the date 8/25/00. Any apparent pattern or relationship between key values is purely incidental.
- An application or analytic tool that uses the key values directly cannot provide meaningful analysis of data in this Datamart.

The ETL Run Key acts in a different way than the primary and foreign keys. It identifies the ETL run that created the fact record. The ETL run information describes the source database that supplied the data for the ETL as well as facts about the run (such as time and date).

Being able to identify the source database is especially important to the report designer who wants to limit reports to data from a particular source, or who wants to differentiate between unshared entities with the same identifying information. Remember that if two entities in different sources share the same identifier (such as name), it does not mean they refer to the same entity. So, being able to identify the source database allows the report designer to differentiate the name Response1 in one source from the name Response 1 in another source.

Data Model Extensions

The data model is extensible. edocs strongly discourages any modifications to the data model at the risk of destroying the Datamart integrity or causing ETL failure. The following suggestions provide reasonable approaches to customization.

Extending dateDim with Holidays

You can extend the `dateDim` by augmenting it with a holiday flag. For example: for each date, add a flag (`Y/N`) that indicates whether the date is a holiday. This information can then be used by some applications to analyze the amount of email received near a major holiday.

The simplest approach is to create another dimension, say `holidayDateDim`, that is an exact copy of the `dateDim` with an additional column, such as `isHoliday`, with a value of `Y/N` appropriately for each date. Then analytic applications can use the new dimension exclusively, instead of `dateDim` in queries.

Another possibility is to create a snowflake dimension that has two columns: `dateKey` as a foreign key to `dateDim` and a holiday flag, as shown in Table 2-1.

Table 2-1. holidayKey

dateKey	isHoliday
73	Y
194	N
249	Y

Assuming that `dateKey` values 73, 194, and 249 represent the dates July 4th, August 25th, and December 25th, the application queries need to join this table with `dateDim` and then to fact tables to discover events falling on United States holidays.

For more generality, you can add a third column to handle holidays in different countries.

The `customer` dimension is another dimension that is a good candidate for extension and the above techniques apply as well.

Since the ETL Tools only manage known dimensions and facts, the applications become responsible for ETL Tools maintaining the extended tables. This may require mechanisms for detecting when the ETL maintained tables change and applying corresponding changes to the extended tables.

Chapter 3

Email Datamart Specifics

This chapter describes the email Datamart data model. The intent is to provide sufficient information to enable you to use the analytic tools and reports of your choice. Familiarity with database concepts is assumed.

Email Fact/Dimension Matrix

The Datamart contains six Event Fact tables, 13 Dimension tables, and 12 Group (Bridge) tables. This section provides an overview of the fact and dimension tables. It shows the relationships between dimensions and fact tables.

Table 3-1 shows the dimensions associated with each fact table. Fact tables are listed across the top row of the matrix, and dimensions appear in the left–most column.

Table 3-1. Fact/Dimension Matrix

Facts ⇒ **Dimensions ⇓**	requestEvent	queueEvent	agentEvent	SLAEvent	messageEvent	KBEvent
action	•	•			•	
agent	•	•	•			
assignment		•				
businessUnit	•	•		•	•	•
channel	•					
comments	•	•				
customer	•	•			•	•
dateDim	•	•	•	•		
feature						•
featureGroup						•
header	•	•			•	
headerAddress	•	•			•	
inbox	•	•			•	

Table 3-1. Fact/Dimension Matrix

Facts ⇒ **Dimensions ⇓**	requestEvent	queueEvent	agentEvent	SLAEvent	messageEvent	KBEvent
intent						•
KBInformation	•					•
phrase						•
queue	•	•		•		
requestStatus	•	•				
requestType	•					
responseAction						•
responseClassification						•
responseClassificationGroup						•
rules						•
SLATag		•				
SLAValue		•		•		
textPattern						•
textPatternGroup						•
timeDim	•	•	•	•		

Email Dimension Tables

This section describes each dimension. As necessary, it includes the database column types, the possible values of the columns, and sample column values.

In addition to the information provided in this chapter, the data model diagram is provided in Appendix A, “Data Model Diagram”.

action

Related fact tables: `requestEvent`, `queueEvent`, `messageEvent`

Every message received by the Contact Center, or acted upon by an Agent or Supervisor, can be categorized with the type of message and type of action taken on it.

This dimension lists such combinations of the two columns. Fact tables that deal with messages have a foreign key that identifies the category of the message in the event.

This table includes:

- `actionKey` – Primary key
- `messageType`: string – one of: `original message`, `manual review message`, `suggested response`, `reply`, `blank response`, `auto acknowledgement`, or `other`
- `actionType` – string: one of: `auto response`, `response`, `manual review`, `auto acknowledgement`, `auto close`, `forward`, or `comment`.

agent

Related fact tables: `requestEvent`, `queueEvent`, `agentEvent`

This table lists all Agents known to the Contact Center. It is updated automatically when Agents change names or email addresses. New Agents are inserted as necessary.

It is important to understand that all people (or components) who can perform the operations of an agent are recorded in this table. These operations include responding to messages and assigning messages to queues. Besides real agents, the automated answer agent and supervisors are recorded here, among others.

This table includes:

- `agentKey` – Primary key.
- `firstName`, `middleName`, `lastName` – First, middle, last names of the Agent.
- `emailAddress` – Email inbox name without the domain name.
- `emailDomain` – Email domain name without the inbox name.
- `type` – Identifies the type of agent, that is, whether it is an Agent (human, desktop Agent) or the Intelligence Engine (Administrator, Supervisor, or Knowledge Manager).

agent group

Related fact tables: `none`

This table lists the agent groups that are in the transactional database. If an existing group is renamed it is inserted as a new group into this table by the ETL. ETL populates this table using the current group information at the time of the ETL run. StartDate and EndDate do not necessarily reflect the actual date a group was renamed or deleted.

This table includes:

- `agentGroupKey` – Primary key.
- `name` – Name of the Agent Group.
- `comment` – Comment entered when the group was created.
- `previousName` – Previous name of the group if it was renamed.
- `startDateKey` – The date that this group was first seen by ETL.
- `endDateKey` – The latest date when this group was active as seen by the ETL.

agent groupset

Related fact tables: `none`

This table lists the association between agents, groups, and sources. In other words, it lists agents that belong to a group that belongs to a source.

This table includes:

- `agentGroupKey: foreign key:` – Identifies the agent group.
- `ETLAdminKey: foreign key:` – Identifies the source to which this group belongs.
- `agentkey: foreign key:` – Identifies the agent that belong to the group.

assignment

Related fact tables: `queueEvent`

The assignment dimension is similar to the `requestStatus` dimension. While `requestStatus` identifies the three `requestEvent` records, the `assignment` dimension identifies the several `queueEvent` records. The semantics of this dimension are best understood in the context of the `queueEvent` fact record. See "queueEvent", on page 33 for additional information.

The contents of this table uses the form shown in Table 3-2 (not all possible rows are listed).

Table 3-2. assignment

assignmentKey	assignedTo	reassignedTo	reasonCode	reason
1	Queue	Queue	RSQ	Reassigned to Sales
2	Queue	Agent	QAG	Assigned to Agent
3	Queue	Exit	QXT	Resolved
4	Agent	Queue	AQU	Reassigned
5	Agent	Reply	ARE	Reply sent
6	Agent	Exit	AXT	Agent Close

In this document, `queueEvent` fact records that have the key value `1`, used in this example, are referred to as *queue→queue* records. Similarly, the remaining `queueEvent` with appropriate `assignmentKeys` is referred to as *queue→agent* records, or *queue→exit* records, and so on.

The `assignment` dimension table captures all possible reasons for assignment and reassignment combinations (and their respective reasons) that can occur in your Contact Center. For example, if your Contact Center has defined 10 reasons (Reason Codes) from which an Agent can choose when reassigning a message, then this table contains 10 records for each assignment type where an assignment type is `agent->agent`, `agent->queue`, `agent->review`, and so on. Likewise, if your Contact Center contains 20 reasons (Reason Codes), a message can be reassigned from a queue

(such as queue->agent, queue->queue, queue->review, then this table contains 20 records for each such assignment type.

businessUnit

Related fact tables: requestEvent, queueEvent, messageEvent, SLAEvent and KBEvent.

The businessUnit dimension table lists all the Business Units that have been created by the Contact Center Administrator for all of the Contact Centers. The table includes:

- businessUnitKey – Primary key; the value is an integer.
- name – The name of the Business Unit; the value is a string.
- description – A description of the Business Unit as entered by the Administrator in the Notes field when the Business Unit is created; the value is a string.
- isDeleted – Set to Y if the business unit has been deleted from the list of active business units in the Contact Center; otherwise set to N.
- isSecure – Set to Y if replies sent from the business unit are secure (meaning that they must be read through the portal); otherwise set to N.

channel

Related fact tables: requestEvent

This table includes all channels through which a request can be received by the Brightware products. The table includes:

- channelKey – primary key.
- name – string: the name of the channel.

The contents of this table are shown below:

Table 3-3. channel

channel	Keyname
1	Web
2	Email
3	Portal
4	Chat

comments

Related fact tables: `requestEvent`, `queueEvent`

This table includes all free text comments entered by Agents and Supervisors. Normally, when an Agent or Supervisor changes the state of a request (open, pend, close) or reassigns a message to another Agent or queue, they can type a free form comment. This table contains those comments.

All comments are stored in this table with an assigned primary key. The matching foreign key can be in either the `requestEvent` or `queueEvent` fact table.

This table includes:

- `commentsKey` – Primary key.
- `comments` – string: Comment typed in by the Agent or Supervisor.

`Comments` are in free-text form. Therefore it is difficult to perform analytics on this column. For example, the comments `misdirected message` and `sorry, wrong person` may mean the same thing, so it is difficult to count the number of messages that were misdirected based on the contents of this column.

customer

Related fact tables: `requestEvent`, `queueEvent`, `messageEvent`, and `KBEvent`

The `customer` dimension table lists all people who sent mail to the Contact Center. In an email processing system, the only attribute known of the customer is the email address. Other attributes may be discovered by other means and added to this table. This table includes:

- `customerKey` – Primary key.
- `emailAddress` – string: Email inbox name without the domain name, for example: `johndoe`
- `emailDomain` – string: Email domain name without the inbox name, for example: `edocs.com`
- `firstName` – string: `NULL`, reserved for future.
- `lastName` – string: `NULL`, reserved for future.
- `alias` – string: reserved for future.
- `preferred` – string: Full email address which the customer prefers.

dateDim

Related fact tables: `requestEvent`, `queueEvent`, `agentEvent`, and `SLAEvent`

Date and time are the two most common dimensions used in most Datamarts. The `dateDim` dimension contains a single row for any given day. The attributes describe the date, such as the day of week, the month of year, whether it was the last day of the month, and so on.

This table contains all possible dates. It is populated automatically the first time you run an ETL event.

This table includes:

- `dateKey` – The primary key.
- `fullDate` – The full date in the native datetime format of the database. The time part of the column is set to midnight (for example 00:00:00). Therefore, this column truly represents only the date.
- `year` – The year of the date.
- `month` – The month of year (1–12).
- `day` – The day of month (1–31).
- `quarter` – The quarter of year (1–4).
- `dayOfYear` – The day number of year (1–366).
- `dayOfWeek` – The day number within week (0–6, 0=Sunday).
- `weekOfMonth` – The week number within month (1–6).
- `daysSince1990` – Number of days since 1/1/1990.
- `monthName` – string: January, February, and so on.
- `dayName` – string: Sunday, Monday, and so on.
- `isLastDayOfMonth` – Character: Y/N.

feature

Related fact tables: `KBEvent`

The `feature` dimension is a table of key-value pairs assigned to a defined Feature in the Knowledge Base. Since each Knowledge Base is unique, Table 3-4 provides an example of the information this table may contain.

Table 3-4. feature

featureKey	name
31	location
32	cancel order
33	credit card denied
34	product feature
35	mailing list request
36	catalog request
37	order status

featureGroup

Related fact tables: `KBEvent`

The `featureGroup` dimension is a matrix of numbers that associates a set of Features found in the same request. These features are assigned a unique group number. Table 3-5 provides an example.

Table 3-5. featureGroup

featureGroupKey	featureKey	phraseKey
301	36	24
301	36	25
303	35	26
303	31	27
303	34	28

The `featureKeys` in this example match those in Table 3-4. From this association, you can determine the names of Features associated with which `featureKeys`. For example, `featureGroupKey 301` includes `featureKey 36` which is a Feature named *catalog request* and `featureKey 37` which is a Feature named *order status*.

Using `featureGroup`, you can perform queries to determine the number of requests that match groups of Features in your Knowledge Base. Then, if necessary, modify an existing Feature or create a new Feature to capture all Features in the `featureGroup`.

header

Related fact tables: `requestEvent`, `queueEvent`, and `messageEvent`

This dimension table contains the extracted headers of all messages. This table includes:

- `headerKey` – Primary key.
- `subject` – Subject of a message.
- `trackingID` – Number identifying the conversation thread. All messages that logically belong to the same thread have the same `trackingID`.
- `returnReceipt` – `"Y"/"N"`
- `receivedDateKey` – This is a foreign key to the `date` dimension indicating the date the message was received. This date is assigned by the mail server.
- `sentDateKey` – Similar to the `receivedDateKey`.
- `receivedTime` – Time the mail server received the message, recorded in seconds since midnight.
- `sentTime` – Similar to `receivedTime`.

headerAddress

This is a *snowflaked* dimension of the header dimension. It contains the email addresses attached to a message.

This table includes:

- `headerKey`: – foreign key. This is not a primary key. Instead, it identifies the header to which it belongs.
- `type` – one of `From`, `To`, `Reply_To`, `Cc`, `Bcc`
- `emailAddress` – Email inbox name without the domain.
- `emailDomain` – Email domain name without the inbox name.

inbox

Related fact tables: `requestEvent`, `queueEvent`, `messageEvent`

This table lists all inboxes known to Contact Center. An inbox is the mailbox to which incoming mail is sent. For example, info@edocs.com, help@edocs.com, and so on.

This table includes:

- `inboxKey` – Primary key.
- `emailAddress` – Email inbox name without the domain name.
- `emailDomain` – Email domain name without the inbox name.

intent

Related fact tables: requestSummary, situationSummary

A shared, varying entity dimension table which defines intents that are used in the Datamart. This table defines individual intents as defined in the Knowledge Base, regardless of their usage. Their actual usage is defined in other tables that reference these definitions.

This table includes:

- `intentKey` – Primary key; the key value which other tables referencing the intentKey point to.
- `name` – Dimension attribute; a string value; the individual name of the intent as defined in the Knowledge Base.

KBInformation

Related fact tables: requestSummary

This table is an unshared varying entity dimension table identifying Knowledge Bases that are used in the Datamart. The Datamart schema for Concierge records two types of Knowledge Bases in this table:

- Email Knowledge Bases are those whose definitions are stored outside of the source database (as described in the ExternalKB table).
- Concierge Knowledge Bases are those whose definitions are stored inside of the source database (in the InternalKB table).

This table includes:

- `KBInformationKey` – Primary key; the key value which tables referencing the KBInformationKey point to.
- `modificationDateKey` – Foreign key; the date the Knowledge Base was last modified prior to the latest ETL. This is NULL for Email Knowledge Bases.
- `modificationTimeKey` – Foreign key; the time of day that the Knowledge Base was last modified prior to the latest ETL. This is NULL for Email Knowledge Bases.
- `name` – Dimension attribute; a string value; the unique name of the Knowledge Base. For Email Knowledge Bases, the name is generated by the ETL (since none is stored in the source database). This name is the last 20 characters in the file/path name for the Knowledge Base with an integer suffix which is the ETLAdminKey. The suffix is added simply to assure uniqueness of Email Knowledge Base names in the Datamart.
- `description` – Dimension attribute; a string value; the description of the Knowledge Base as defined in the Knowledge Base.
- `languageKey` – Identifies the language for which the Knowledge Base is designed.

language

Related fact tables: requestEvent

Related dimension tables: KBInformation

This table includes all languages for which a request can be processed by the Brightware products. For requests, this is the language in which the request was determined to have been written. For Knowledge Bases, this is the language that is supported by the Knowledge Base. The table includes:

- `languageKey` – Primary key.
- `name` – string: the name of the language.

This table provides an example of the information this table may contain:

Table 3-6. channel

languageKey	Keyname
1	English - US
2	English - UK
3	Danish
4	German

phrase

Related fact tables: `KBEvent`

The `phrase` dimension is a table of key-value pairs assigned to a defined Phrase in the Knowledge Base. Since each Knowledge Base is unique, Table 3-7 provides an example of the information this table may contain.

Table 3-7. phrase examples

phraseKey	name
21	What is
22	Do you sell
23	account number
24	Please send
25	Can I get
26	Please add
27	Where are
28	how long

queue

Related fact tables: `requestEvent`, `queueEvent`, `SLAEvent`

This table lists all Contact Center queues. It is updated automatically during ETL runs if new Contact Center queues were added.

It also tracks previous names of a queue for analytic purposes. Thus, the previous queue names are tracked and time stamped.

This table includes:

- `queueKey` – Primary key.
- `name` – String: name of the queue such as `Service`, `Sales`, and so on.
- `type` – Character(254): the queue table list one of the six new types of queue: Agent; Inbound; Outbound; Review; Exception; Default;

You can change the name of a queue, but its type will remain the same.

- `priority` – Integer: such as `1`, `2`, `3`, and so on.
- `previousName` – String: If a queue name changed since the previous ETL, then this is the previous name whereas the column `name` is the current name. When a new queue is created, then the `previousName` is the same as `name`.
- `startDateKey` and `endDateKey` – `previousName` was effective between these two dates. These are date keys — foreign keys. The actual dates are in the `dateDim` dimension. Therefore, these keys must be joined with the `dateDim` to discover the actual dates.

requestStatus

Related fact tables: `requestEvent` and `queueEvent`

The `requestEvent` fact table (see "requestEvent", on page 30 for more information) contains three kinds of event records. The `requestStatus` dimension identifies the types of those records. Examples of the contents of these records are shown in Table 3-8.

Table 3-8. requestStatus examples

requestStatusKey	status	reasonCode	reason
1	Open	UNK	Unknown
2	Open	ROS	Reopened by Supervisor
3	Open	ROA	Reopened by Agent
4	Pending	RES	Researching issue
5	Pending	AFD	Away from desk
6	Closed	IRR	Issue resolved with reply
7	Closed	IRN	Issue resolved without reply
8	Incomplete Open	NULL	NULL
9	Incomplete Pending	NULL	NULL

The `status` column values of *Incomplete Open* and *Incomplete Pending* are generated by the ETL Tools to indicate that when the ETL event ran, the request was in the OPEN or PENDING state. Thus, if fact records existed whose `requestStatusKey` values were 8 (as in this example), then the associated request was in the OPEN state at the time of the ETL event.

The contents of the `reasonCode` column are derived from a pick list that an Agent uses. The `reason` column shows the description of the associated `reasonCode`. In other words, the contents of `reasonCode` column are not generated as free text by the Agent. Therefore, these values are good candidates for Analytics.

The `requestStatus` dimension table captures all possible reasons for the state of a request. For example, if your Contact Center has five Reason Codes that can be assigned to `Open` requests and three Reason Codes for `Pending` requests, then this table includes five `Open` and three `Pending` `requestStatusKeys`, or one `requestStatusKey` for each Reason Code record in your Contact Center database attached to the state of a request.

requestType

Related fact tables: requestEvent

This table includes all types of requests which can be processed by the Brightware products. The table includes:

- `requestTypeKey` – Primary key.
- `name` – string: The name of the request type.

The contents of this table are shown below:

Table 3-9. responseAction

requestTypeKey	name
1	Customer - initiated
2	Agent - initiated

responseAction

Related fact tables: `KBEvent`

The `responseAction` dimension is a table of key-value pairs assigned to a defined Action in the Knowledge Base. Since each Knowledge Base is unique, Table 3-10 provides an example of the information this table may contain.

Table 3-10. responseAction examples

responseActionKey	name
61	add to mailing list
62	how do I order
63	order shipment

responseClassification

Related fact tables: `KBEvent`

The `responseClassification` dimension is a table of key-value pairs assigned to a defined classification in the Response Library. Since each Knowledge Base is unique, Table 3-11 provides an example of the information this table may contain.

Table 3-11. responseClassification

responseClassificationKey	name
14	Television
9	Electronics
15	Camera
6	Digital

responseClassificationGroup

Related fact tables: `KBEvent`

The `responseClassificationGroup` dimension is a matrix of numbers that associates a set of response classifications found in the same request. These response classifications are assigned a unique group number. Table 3-12 provides an example.

Table 3-12. responseClassificationGroup

responseClassificationGroupKey	responseClassificationKey
21	14
21	9
13	15
13	9
13	6

Combining the contents of this table with Table 3-11 shows that `responseClassificationGroup` *21* represents the `responseClassification` *Electronics* and *Television*, while the `responseClassificationGroup` *13* represents the `responseClassification` *Electronics*, *Camera*, and *Digital*.

If a fact record has the value 13 for the `responseClassificationGroup` column, then the associated request would be classified by the Agent as *Electronics*, *Camera*, and *Digital*.

rules

Related fact tables: `KBEvent`

The `rules` dimension is a table of key-value pairs assigned to a defined Rule in the Knowledge Base. Since each Knowledge Base is unique, Table 3-13 provides an example of the information this table may contain.

Table 3-13. rules examples

rulesKey	name
51	marketing request
52	address change
53	investment inquiry

SLATag

Related fact tables: `queueEvent`

This dimension is similar to `SLAValue`, but provides a degree of qualitative measure rather than quantitative. That is, instead of containing values, it contains tags, as shown in Table 3-14.

Table 3-14. SLATag

SLATagKey	eventTag	requestTag	handlingTag
1	Normal	Normal	Normal
2	Normal	Warning	Critical
3	Warning	Critical	Exceeds SLA
4	••••	••••	••••

Similar to the `SLAValue`, this dimension is of little value by itself. However, when used as a dimension for the `queueEvent` fact table, it can qualitatively indicate whether a message assignment to a queue exceeded the assigned SLA value. See "queueEvent", on page 33 for more information.

SLAValue

Related fact tables: `queueEvent` and `SLAEvent`

The `SLAValue` dimension is simply a matrix of numbers based on the Service Level Agreement (SLA). It captures all possible Service Level Agreement values of each queue in your Contact Center. Table 3-15 provides an example of the information this table contains.

Table 3-15. SLAValue

SLAValueKey	warningDuration	criticalDuration	goalDuration
1	7200	10800	14400

Table 3-15. SLAValue

SLAValueKey	warningDuration	criticalDuration	goalDuration
2	28800	23200	86400
3	64800	75600	86400
4	64800	86400	86400

SLAValueKey is the primary key.

By itself, this dimension is of little value. All it says is that at some time, there existed some queue whose SLA values matched the values of a row in this table.

However, when used with the `SLAEvent` fact table, and its other dimensions, it is possible to identify which queue had one of these values, the date it had the values, and the duration of those values.

textPattern

Related fact tables: `KBEvent`

The `textPattern` dimension is a table of key-value pairs assigned to a defined text pattern in the Knowledge Base. Since each Knowledge Base is unique, Table 3-16 provides an example of the information this table may contain.

Table 3-16. textPattern

textPatternKey	name
1	phone number
2	zip code
3	account number
4	credit card number
5	part number
6	SKU number
7	social security number

textPatternGroup

Related fact tables: `KBEvent`

The `textPatternGroup` dimension is a matrix of numbers that associates a set of text patterns found in the same request. These text patterns are assigned a unique group number. Table 3-17 provides an example.

Table 3-17. textPatternGroup

textPatternGroupKey	textPatternKey
101	5
101	3
201	7
201	4
201	3

The `textPatternKeys` in this example match those in Table 3-16. From this association, you can determine which text pattern names are associated with which `textPatternGroupKey`.

Using `textPatternKey`, you can perform queries to determine the number of requests that match groups of Text Patterns in your Knowledge Base. Then, if necessary, modify an existing Text Pattern or create a new Text Pattern to capture all Text Patterns in a `textPatternGroup`.

timeDim

Related fact tables: `requestEvent`, `queueEvent`, `agentEvent`, and `SLAEvent`

This dimension contains a single row for each second of the day. It is populated automatically the first time you run an ETL event. The attributes describe the time, such as the minute of the hour, seconds of the minute, whether it was the first or last half of the hour, and so on.

This table includes:

- `timeKey` – integer: the primary key.
- `secondsOfDay` – integer: `0–86399` (seconds since midnight).
- `hour12` – integer: `1–12` (hour of the day/12-hour clock).
- `hour24` – integer: `0–23` (hour of the day/24-hour clock).
- `minuteOfHour` – integer: `0–59`.
- `secondOfMinute` – integer: `0–59`.
- `time12` – char (8): `00:00:00` (12-hour clock).
- `time24` – char (8): `00:00:00` (24-hour clock).
- `amPm` – char (2): `"AM"` or `"PM"`
- `halfHour` – integer: `0, 30`.
- `quarterHour` – integer: `0, 15, 30, 45`.

Email Fact Tables

Fact tables track the actual events. With the exception of `KBEvent`, most events have several properties in common, such as:

- Date of the event (`dateKey`).
- Time of the event (`timeOfDay`, seconds since midnight).
- Duration of the event (`eventDuration`, in seconds).

If the fact table contains more than one type of event, a key is used to identify the type of event (`requestStatusKey`, `assignmentKey`).

Where a fact table tracks more than one type of event (`requestEvent`, `queueEvent`), it is usually essential to use the respective type identifier dimensions (`requestStatus`, `assignment`) to analyze the event records separately.

Queries that use these fact tables should also use the type identifier dimensions, since the meaning or semantics of several facts and measures depend on the type of the event record.

Keys are always non–additive. The *additiveness* of other facts and measures are indicated for each fact table in the sections that follow.

requestEvent

This fact table tracks the state changes of requests. A given request, identified by `requestNumber`, can be in one of three states at any given time: OPEN, PENDING, or CLOSED. Therefore there can be three kinds of event records in this fact table. The type is identified by the `requestStatus` dimension using the key `requestStatusKey`.

Also, a request is always associated with some message, identified by `messageNumber`. The following four sections describe this in greater detail.

It is important to understand that all requests, whether agent-initiated or customer-initiated, are recorded in this table. This is not the case with the queueEvent table in which only customer-initiated requests are recorded. Any query which does not filter based upon requestTypeKey will return information about both types of requests.

requestEvent Independent Dimensions and Facts

This classification includes:

- `businessUnitKey` – Identifies the business unit in which the event occurred.
- `requestNumber` (non–additive) – A unique identification number assigned to a request.

This is useful in count–based aggregation. For example, to find the number of events associated with a specific customer, count the number of occurrences of `requestNumber (SQL: count(requestNumber))`. However, to find the number of different requests associated with a customer, count the distinct values of `requestNumber (SQL: count(distinct requestNumber))`.

- `messageNumber` (non–additive) – A unique identification number assigned to a message.

  This can also be used as a foreign key to the `messageEvent` fact table to obtain the actual message body text.

1. `dateKey` – Identifies the date of this event.

2. `timeOfDay` (non–additive) – The time of day, in seconds since midnight, of this event.
3. `commentsKey` – Identifies the Agent or Supervisor (or both) comments associated with this event.

requestEvent Dimensions Depending on requestNumber

This classification includes:

- `customerKey` – Identifies the customer associated with this request.
- `inboxKey` – Identifies the mailbox that received the customer's request.
- `channelKey` – Identifies the channel through which the request was received.
- `languageKey` – Identifies the language of the request as determined by the analysis done.
- `requestTypeKey` – Identifies the type of the request.

requestEvent Dimensions Depending on messageNumber

This classification includes:

- `headerKey` – Identifies the message headers associated with this message.
- `actionKey` – Identifies the message type and work action taken on this message.

requestEvent Dimensions and Facts that Depend on the Event Type

This classification includes the `requestStatusKey` which identifies the type of event (OPEN, PENDING, CLOSED).

Any query that uses the following dimension keys, or facts and measures, without also using the event identifier, `requestStatusKey`, is probably incorrect.

For OPEN events:

- `queueKey` – Identifies the first queue to which this request was assigned, excluding the inbound queue.
- `agentKey` – Identifies the first Agent to whom this request was assigned, excluding the Intelligence Engine component.
- `actingAgentKey` – Identifies the agent who Pended or Closed a message on behalf of the Agent assigned to the message, for example it might identify the Supervisor who Closed a message assigned to a Desktop Agent.
- `eventDuration` (semi–additive) – The elapsed time while this request was in the OPEN state before it went to CLOSE or PEND.
- The eventDurations may be aggregated (added, averaged, and so on) with other OPEN events for the same or different requests to get a measure of handling time.
- `requestDuration` (semi–additive) – The elapsed time from the beginning of the initial open event for the request until the end of the current event (including the duration of the current event described by this record). However, this item is reset if the message is transferred from one Business Unit to another.

- `systemDuration` – The elapsed time the message is in the system; this differs from the request Duration in that it is not reset if the message is transferred from one Business Unit to another.
- `isFirst` (non–additive) – Set to `Y` if this is the first OPEN event for this request; otherwise set to `N`.
- `isLast` (non–additive) – Set to `Y` if this is the last OPEN event for this request; otherwise set to `N`.

For PENDING events:

- `queueKey` – Identifies the queue from which this request was pended. This is the queue to which the associated message was assigned when the request was pended.
- `agentKey` – Identifies the Agent who pended this request.
- `actingAgentKey` – Identifies the agent who Pended or Closed a message on behalf of the Agent assigned to the message. For example it might identify the Supervisor who Closed a message assigned to a Desktop Agent.
- `eventDuration` (semi–additive) – The elapsed time while this request was in the PENDED state. The eventDurations may be aggregated (added, averaged, and so on) with other PENDED events for the same or different requests to get a measure of idle time.
- `requestDuration` (semi–additive) – The elapsed time from the beginning of the initial open event for the request until the end of the current event (including the duration of the current event described by this record). However, this item is reset if the message is transferred from one Business Unit to another.
- `systemDuration` – The elapsed time the message is in the system; this differs from the requestDuration in that the systemDuration is not reset if the message is transferred from one Business Unit to another.
- `isFirst` (non–additive) – Set to `Y` if this is the first PEND event for this request; otherwise set to `N`.
- `isLast` (non–additive) – Set to `Y` if this is the last PEND event for this request; otherwise set to `N`.

For CLOSED events:

- `queueKey` – Identifies the queue from which the request was closed.
- `agentKey` – Identifies the Agent who closed this request.
- `actingAgentKey` – Identifies the agent who Pended or Closed a message on behalf of the Agent assigned to the message. For example, it might identify the Supervisor who Closed a message assigned to a Desktop Agent.
- `eventDuration` (semi–additive) – The elapsed time since the most recent OPEN event.
- `requestDuration` (semi–additive) – The value is the elapsed time from the beginning of the initial open event for the request until the start of the current event. That is, it does not include any duration of the event described by this record since, in most cases, the duration of a Closed event is infinite. In other words, it is the time between an OPEN and a CLOSE event. Essentially, it is a measure of how long it took to process a request.
- `systemDuration` – The elapsed time the message is in the system. This differs from the requestDuration in that the systemDuration is not reset if the message is transferred from one Business Unit to another.

- `isFirst` (non–additive) – Set to `Y` if this is the first CLOSED event for this request; otherwise set to `N`.
- `isLast` (non–additive) – Set to `Y` if this is the last CLOSED event for this request; otherwise set to `N`.

By *averaging* the `eventDurations` of the first CLOSED event records across several requests (`requestNumbers`), it is possible to measure the average response times of the system.

`requestDuration` always includes `eventDuration`. Since requestDuration is to some degree a *running total* since its creation, it does not make sense to add this number to another `requestDuration` event for the same `requestNumber`. It may be *averaged*, however, with `requestDurations` of other `requestNumbers`.

queueEvent

This fact table tracks the flow of requests (email messages) through the Queue Manager (email queue management system).

It is important to understand that only customer-initiated requests are recorded in this table. This is not the case with the requestEvent table which includes agent-initiated requests. Any query written against this table, which summarizes request information, will necessarily include only information about customer-initiated requests. This is due to the fact that agent-initiated requests are not processed by the Queue Manager at all. They do not flow through the queue management system. Therefore, there are no records of such requests in this table.

To understand the data represented in the fact table, it helps to understand how a request is processed by the Queue Manager. The following section provides an overview.

Queue Manager Overview

When the system receives an email message, it is assigned a request identifier (request number). The Queue Manager uses this request number to track the email message as it flows through the queue management system. The sections that follow describe how requests are processed.

Request Flow through the Queue System

A request enters the system and is initially assigned to a queue. Perhaps it is then reassigned to another queue, then to another, then perhaps to a previously assigned queue, and so on. Eventually it exits the system when the request is closed. This is known as the *request flow through the queue system*.

A request is always in a queue (Sales, Service, and so on). It is only in one queue at any given time. Therefore, if a request is assigned to the Sales queue, it is only present in the Sales queue. If it is reassigned to the Service queue, it is no longer in the Sales queue.

On occasion, however, the Queue Manager makes a copy of the request and tracks the flow of the copy. A copy of a request is created when an Agent creates a reply. It is a copy of the original request that an Agent edits and sends as a reply. Since an Agent can send several separate replies, several different copies of the original request may exist. The Queue Manager tracks the copies of the request independently.

All copies, including the original request, are associated with (assigned to) the same request number. To distinguish and track the various copies of the original request, the Queue Manager

assigns each of them, including the original request, a unique identifier (message number). For the purpose of this document, there are two types of requests: the original and a reply.

Request Flow through Agent Desktops

When the Queue Manager assigns a request to a queue, it may also assign the request to an Agent. It is only after the Queue Manager assigns a request to an Agent that this Agent can receive the request on his or her desktop.

Agents may act on the request by sending a reply, closing it, or reassigning it to another Agent.

It is important to understand a request does not flow from a queue to an Agent's desktop. An Agent is simply assigned to a request that is also assigned to some queue.

However, an Agent can reassign the request to another Agent. In which case, it does flow from the former Agent's desktop to the latter Agent's desktop.

The word *flow*, as used in this document, means that a request has moved from its former place to a new place: it has *flowed* from one place to another.

To emphasize, a request flows from one desktop to another when an Agent reassigns the request to another Agent.

A flow in the Agent's desktop may or may not cause a flow in the queues managed by the Queue Manager. If an Agent reassigns the request to an Agent in the same queue, then the request flows from that Agent desktop to another.

If the Agent reassigns the request to an Agent in a different queue, then the request flows from the former queue to the new queue. The former Agent no longer controls the request. The Queue Manager proceeds to assign a new Agent to the request from the new queue as previously described.

The Queue Manager tracks the flow of requests through the queues and Agent desktops independently.

Replies and Reviews

When an Agent composes and sends a reply (using the **Send** button on the Agent desktop), the Queue Manager copies the request (assigning it a new message number) with the Agent's comments, adds the copy to the *reply* queue, and tracks its progress as any request in any queue. Note that the request does not flow from the Agent's desktop since the Agent still controls the request and is free to compose another reply or reassign the request to another Agent.

When an Agent closes a request (using the **Close** button on the Agent desktop), it simply flows from the Agent desktop and out of (or exits) the Queue Manager.

By combining these two actions, using the **Send & Close** button, an Agent achieves the same results, as if the two actions were separate: a copy of the request is placed in the reply queue and tracked independently; the request flows from the Agent desktop and exits the queue system.

For Agents under Supervisor review, all composed and sent copies of the requests are placed in the review queue instead of the reply queue. The remainder of the flow is much the same as previously described with a subtle difference: while a request never flows from a queue to the reply queue, it does flow from a queue to the review queue since it may remain in the review queue indefinitely.

Overlapping Time Series of Events

A time series is a sequence of events that occur consecutively; events do not overlap in time; that is, one event in the series does not start until the previous event finishes.

For example, a request flowing through the queues forms a time series: a request flows from one queue to another and eventually to exit.

A request also flows through Agent desktops: it flows from one Agent's desktop to another, by reassignment and eventually to exit. This forms another time series.

These two time series can overlap. For example, while flowing through the queues, a request may be assigned to an Agent and flow through the Agent's desktop. The event records (in `queueEvent`), which track the flow of requests through the queue system and the Agent desktops, form an overlapping time series. Figure 3-1 illustrates these concepts.

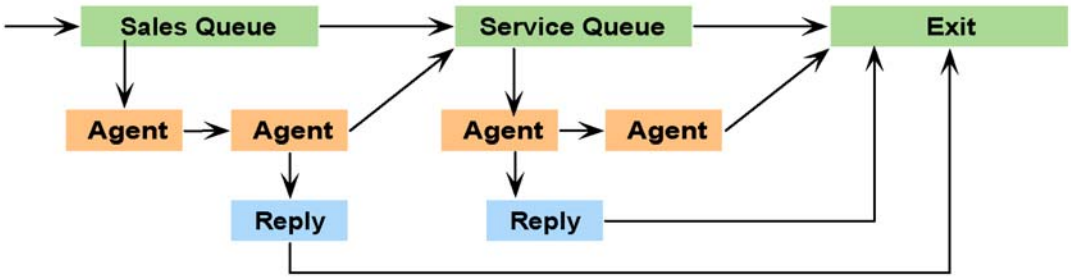

Figure 3-1. Overlapping Time Series

This figure shows three overlapping time series.

In the first time series (top row), a request enters the Sales queue, flows to the Service queue, and finally flows to Exit.

In the second time series (middle row), a request is assigned to the first Agent in the Sales queue, flows to the second Agent in the same queue, and then to the Service queue. The process is repeated for the Service queue where the request flows from one Agent's desktop to another and finally flows to Exit.

In the third time series (bottom row), two Agents compose and send replies (without closing the requests) that are introduced into the Reply queue, from which they flow to Exit.

It is important to understand the various activities that can occur, their relationship (flow) with other activities, and the time series. These elements are key to understanding the `queueEvent` fact table and the types of records it tracks.

queueEvent Data Model

The `queueEvent` fact table tracks (records) several types of fact records, corresponding to the various activities of the Queue Manager.

Each row of the fact table is of a specific type indicated by the dimension key (and its respective dimension) `assignmentKey`.

Any query that uses the queueEvent fact table, but does not also link to (join with) the event identifier dimension, assignment, is probably incorrect. See "assignment", on page 16 for more information.

The assignmentKey indicates one of several types of event records. The corresponding assignment dimension lists all possible types of records.

For example, if a row of assignment contains the values assignedTo="Queue" and reassignedTo="Queue" then any queueEvent fact record whose assignmentKey value is the same as the primary key of said record of assignment is a fact record that indicates a flow from one queue to another. For such a fact record, the key assignedQueueKey indicates the queue that held a request (requestNumber) and forwardedQueueKey indicates the queue to which the request flowed.

As another example, if a fact row contained an assignmentKey value that linked to a record of the assignment dimension containing the values assignedTo="Agent" and reassignedTo="Agent", then the fact record describes a flow from one Agent's desktop to another. In this case, the former Agent is identified by assignedAgentKey and the latter Agent is identified by forwardedAgentKey. Since this flow can only occur within the same queue, the queue is identified by assignedQueueKey and forwardedQueueKey.

In this document, the event records of the types previously discussed in the examples are referred to as queue–>queue and agent–>agent respectively.

There are several other types of event records in this fact table. The following sections provide example scenarios.

queueEvent Entry to Exit Events

In the simplest case, a request enters the queue management system and is assigned to the Intelligence Engine component (the automated Answer Agent). The Intelligence Engine closes the request, perhaps as an unsolicited advertisement message. This generates event records of the types shown in Example 5-1.

Example 5-1

1. entry—>Answer Agent
2. Answer Agent—>exit
3. entry—>exit

The numbers are for reference only and carry no other significance.

The entry–>Answer Agent and Answer Agent–>exit event types indicate a flow through the Agent desktops (even though no physical desktop is involved) and the entry–>exit indicates a flow through the queue system.

The entry–>Answer Agent record indicates that Answer Agent was assigned the request on entry (which is always the case). The Answer Agent–>exit indicates that Answer Agent closed the request. The entry–>exit indicates the request was closed on entry in the queue management system (that is, from the Entry queue).

From this scenario, the following question can be asked: "How many requests were closed as they entered the system?" The answer is to count the number of records of type entry–>exit. In SQL, this is written as shown in Query 5-1.

Query 5-1

```
SELECT COUNT(*)
FROM queueEvent QE,
     Assignment ASG
WHERE QE.assignmentKey = ASG.assignmentKey
  AND ASG.assignedTo = 'Entry'
  AND ASG.reassignedTo = 'Exit'
```

If you are familiar with SQL, you can see that this is a join between the fact table and its event identifier `assignment` dimension where the dimension is constrained to identify `entry–>exit` type records.

queueEvent Entry to Exit with Reply Events

In the next case, Answer Agent sends a reply before closing the request and generates the event records shown in Example 5-2.

Example 5-2

1. entry—>Answer Agent
2. Answer Agent—>reply
3. Answer Agent—>exit
4. entry—>exit
5. reply—>exit

There are now two new types of records in addition to ones shown in "Example 5-1", on page 36. The `Answer Agent–>reply` record indicates that Answer Agent sent a reply. The `reply–>exit` indicates that a reply was sent.

From this scenario, the following question can be asked: "How many replies did Answer Agent send out (automated replies)?" The solution is to count the number of replies, in SQL, as shown in Query 5-2.

Query 5-2

```
SELECT COUNT(*)
FROM queueEvent QE,
     Assignment ASG
WHERE QE.assignmentKey = ASG.assignmentKey
  AND ASG.assignedTo = 'Reply'
  AND ASG.reassignedTo = 'Exit'
```

This query counts the number of replies sent by the system. In addition, it also includes replies sent by human Agents. To count the number of automated replies, you must look for `Answer Agent-agent–>reply` records as shown in Query 5-3.

Query 5-3

```
SELECT COUNT(*)
FROM queueEvent QE,
     Assignment ASG
WHERE QE.assignmentKey = ASG.assignmentKey
  AND ASG.assignedTo = 'Answer Agent'
  AND ASG.reassignedTo = 'Reply'
```

In a case where auto-reply is disabled in the queue management system, or Answer Agent is unable to reply and close the request automatically, the request is eventually assigned to another queue and a human Agent.

Example 5-3 shows the event records for one such case where a human Agent receives a request, reassigns it to another Agent in the same queue who then replies and closes it.

Example 5-3

1. entry—>Answer Agent
2. Answer Agent—>queue
3. entry—>queue
4. queue—>agent
5. agent—>agent
6. agent—>reply
7. reply—>exit
8. agent—>exit
9. queue—>exit

Note that, compared to "Example 5-2", on page 37, the `entry–>exit` is replaced by `entry–>queue` to indicate that the request was not closed on entry but was assigned to a queue. Then `queue–>agent` indicates that an Agent was initially assigned to the request. The `agent–>agent` indicates that the initial Agent reassigned the request to another Agent. The `agent–>reply` and `agent–>exit` indicates that the Agent replied and closed the request respectively. Finally, `queue–>exit` indicates the request was closed from the queue.

The flow records through the queues are: `entry–>queue` and `queue–>exit`. This means that when the request entered the system, it was initially given to the entry queue (as is always the case). It then flowed to a queue where it was closed.

The flow records through the Agent desktops: `entry->Answer Agent`, `Answer Agent–>queue`, `queue–>agent`, `agent–>agent`, and `agent–>exit`. This means that Answer Agent was unable to respond to the request and assigned it to a queue, an Agent was assigned to the request, who reassigned it to another Agent, who finally closed it.

As for the `agent–>reply` and `reply–>exit` records, they are in a separate time series.

Remember that when an Agent replies to a request, that Agent does not necessarily give up control of the original request. The Agent is free to send another reply. Therefore, the request does not flow from the Agent's desktop to the reply queue.

A **Send & Close** action by an Agent is equivalent to a **Send** followed by a **Close**.

Query 5-4 asks the following question: "How many times were assignments made to Agents?" This requires counting the `agent–>agent` and `agent–>exit` records.

Query 5-4

```
SELECT COUNT(*)
FROM queueEvent QE,
     Assignment ASG
WHERE QE.assignmentKey = ASG.assignmentKey
  AND ASG.assignedTo = 'Agent'
  AND ASG.reassignedTo IN ('Agent', 'Exit')
```

Based on this example, the query should return a count of 2: Record numbers 5 and 8 (see "Example 5-3", on page 38). This is correct since the Queue Manager initially assigned the request to an Agent, and that Agent reassigned it to another Agent.

Query 5-5 asks "Who are the Agents receiving requests and how many are they getting?" To find the identity of the Agents, you must link to the `agent` dimension. To get the count per Agent, you must use the SQL `GROUP` operator.

Query 5-5

```
SELECT Agent.emailAddress, count(*)
FROM queueEvent QE,
     Assignment ASG,
     Agent
WHERE QE.assignmentKey = ASG.assignmentKey
  AND QE.assignedAgentKey = Agent.agentKey
  AND ASG.assignedTo = 'Agent'
  AND ASG.reassignedTo IN ('Agent', 'Exit')
GROUP BY Agent.emailAddress
```

Query 5-6 asks another interesting question: "Who are the Agents who are reassigning requests to other Agents instead of closing requests themselves?" This SQL query is the same as the previous one, but it excludes the `agent–>exit` records.

Query 5-6

```
SELECT Agent.emailAddress, count(*)
FROM queueEvent QE,
     Assignment ASG,
     Agent
WHERE QE.assignmentKey = ASG.assignmentKey
  AND QE.assignedAgentKey = Agent.agentKey
  AND ASG.assignedTo = 'Agent'
  AND ASG.reassignedTo IN ('Agent')
GROUP BY Agent.emailAddress
```

The Importance of Understanding Flows-in-Time Series

This section answers the question: “Why is it important to understand the records part of a flows-in-time series?”

All records that belong in flows-in-time series posses an *additive measure* property. In the simplest sense, this means the counts of such records (which is a measure) can be added to reveal a meaningful result. The measures `eventDuration` and `eventHandlingTime` can also be aggregated (added, averaged, and so on).

For example, the records of type `agent–>agent` indicate an event where an Agent received a request and reassigned it to another Agent. As previously noted, the count of these records indicate the number of times an Agent reassigns the request to another Agent. Records of type `agent–>exit` indicate an event where an Agent received a request and closed it. Records of type `agent–>queue` indicate an event where an Agent received a request and reassigned it to another queue.

Since these three records belong to the same flow, you can add the counts of these records to reveal the number of times an Agent received a request and either reassigned it to another Agent or queue or closed it. Since an Agent can only *get the request off his desktop*, this is also the measure of the number of requests handled (or received) by the Agent.

To illustrate a counter example, the record of type `queue–>exit` indicates an event where a request is closed from a queue. However, this record is in a different flow from an `agent–>exit` record. Therefore adding the counts of these records together yields the number of requests closed either from the queue or by an Agent. Such a measure is meaningless since the Agent closing the request causes the request to be closed from the queue. Therefore, if an Agent closes a single request, adding the counts of these records yields a sum of 2, indicating that 2 requests were closed, which is clearly incorrect!

The following section enumerates the various types of event records, grouped by the time series flow to which they belong, and provides a brief explanation of each. Where relevant, a few other columns from the `queueEvent` fact table whose meaning depends on the record type are also included here.

Queue Flow records

- `entry–>queue`

  This record starts the queue flow time series. Answer Agent was unable to close the request automatically. A queue assignment was made. The queue is indicated by `fowardedQueueKey`. The `eventDuration` is the amount of time, in seconds, the request was in the system before it was assigned to the queue.
- `entry–>exit`

  This record indicates a request that exits the queue management system as soon as it enters. No Agent or queue assignments were made.
- `queue–>queue`

  This record indicates that an assignment was made to a queue indicated by `assignedQueueKey` and later reassigned to the queue indicated by `forwardedQueueKey`. The `eventDuration` is the amount of time the request was in the assigned queue before reassignment.
- `queue–>exit`

  This record indicates that the request was assigned to the queue indicated by `assignedQueueKey` and was later closed from the queue.

- `queue–>Review`

  This record indicates that the request was assigned to the queue indicated by `assignedQueueKey` and later moved to the review queue for review by the supervisor. This occurs when an Agent who is under Supervisor review composes and sends a reply to a request.

- `review–>queue`

  This record indicates that a requested Agent reply was disapproved by the Supervisor and reassigned to a queue indicated by `forwardedQueueKey`. This occurs when an Agent who is under Supervisor review sends a reply to a request. The reply is automatically moved from the queue to the review queue where the Supervisor disapproved of the reply and instead of closing the request, the Supervisor reassigned the request to a queue for reconsideration by another Agent.

Agent desktop flow records

- `entry–>Answer Agent`

  This record starts the Agent desktop flow time series. This flow record always occurs. It indicates that Answer Agent assigned the request on entry.

- `Answer Agent–>exit`

  This record indicates that Answer Agent closed the request.

- `Answer Agent–>queue`

  This record indicates that Answer Agent assigned the request to a queue indicated by `forwardedQueueKey`.

- `agent–>agent`

  This record indicates that a request was assigned to the Agent indicated by `assignedAgentKey` who later reassigned it to the Agent indicated by `forwardedAgentKey`. The `eventDuration` is the amount of time, in seconds, the first Agent held the request on his desktop before reassigning it. Both `assignedQueueKey` and `forwardedQueueKey` indicate the same queue.

- `agent–>exit`

  This record indicates that a request was assigned to the Agent indicated by `assignedAgentKey` who later closed the request. `assignedQueueKey` indicates the queue from which the request was closed. `eventDuration` is the amount of time the request was on the Agent's desktop before being closed.

- `agent–>queue`

  This record indicates that a request was assigned to the Agent indicated by `assignedAgentKey` in queue `assignedQueueKey` who later reassigned the request to the queue indicated by `forwardedQueueKey`.

- `queue–>agent`

  This record indicates that the request was assigned to the queue indicated by `assignedQueueKey` and then assigned to the Agent indicated by `forwardedAgentKey`. The `eventDuration` is the time duration the request was in the queue before being assigned to an Agent.

When an Agent reassigns a request to an Agent in a different queue, the records created are `agent–>queue` and `queue–>agent`.

Non-flow record types

These records do not flow in either time series previously discussed. They may flow in a third time series, or in none at all, and usually indicate events of interest. The measures of these records (counts, sums, averages, and so on) should not be combined with the measures of the records in other time series.

- `agent–>reply`

  This record indicates that a request was assigned to the Agent indicated by `assignedAgentKey` who composed and sent a reply. `eventDuration` is the amount of time the request was on the Agent's desktop before a reply was sent.
- `agent–>review`

  This record indicates that a request was assigned to the Agent indicated by `assignedAgentKey` who composed a reply and either explicitly requested Supervisor review or the reply was automatically sent for Supervisor review. `eventDuration` is the amount of time the request was on the Agent's desktop before Supervisor review was requested.
- `Answer Agent–>reply`

  This record indicates that Answer Agent automatically sent a response to the original request. To distinguish between an auto acknowledgement and a *completed reply* use the `actionKey` to link to the `action` dimension and look at the `actionType` and `messageType` columns.
- `reply–>exit`

  This record indicates that a reply was sent. This occurs when an Agent sends a reply.
- `review–>reply`

  This record indicates that the Supervisor approved and sent the reply. This occurs when an Agent under Supervisor review sends a reply and the reply is approved.
- `review–>exit`

  This record indicates that a requested Agent reply was disapproved by the Supervisor. This occurs when an Agent who is under Supervisor review sends a reply to a request. The reply is automatically moved from the queue to the review queue where the Supervisor disapproves of the reply and instead closes the request without a reply.

To conclude the general discussion of the `queueEvent`, the following example illustrates the records generated by the queue management system.

1. A request enters the system.
2. Answer Agent sends out an auto-acknowledgement.
3. Answer Agent assigns the request to the Sales queue.
4. Agent Andy is assigned to the request.
5. Agent Andy sends a reply to the request.
6. Agent Andy is under Supervisor review. The Supervisor approves the reply.
7. Agent Andy reassigns the request to Agent Jim in the Service queue.
8. Agent Jim sends a reply to the request and closes it.
9. Agent Jim is also under Supervisor review. The Supervisor disapproves of the reply and reassigns the request to Agent Andy in the Sales queue.
10. Agent Andy, still under Supervisor review, sends a reply and finally closes the request.
11. The Supervisor approves Agent Andy's reply and sends it out.

Table 3-18 provides an overview of the event records that this scenario generates.

Table 3-18. Event Record Scenario

Record #	Event Record	Agent/Queue Assignment
1	entry—>Answer Agent	
2	Answer Agent—>reply	
3	reply—>exit	
4	Answer Agent—>queue	queue=Sales
5	entry—>queue	queue=Sales
6	queue—>agent	queue=Sales, agent=Andy
7	agent—>review	agent=Andy
8	review—>reply	
9	agent—>reply	agent=Andy
10	reply—>exit	
11	agent—>queue	agent=Andy, queue=Service
12	queue—>queue	assignedQueue=Sales, forwardedQueue=Service
13	queue—>agent	queue=Service, agent=Jim
14	agent—>review	agent=Jim
15	queue—>review	queue=Service
16	review—>queue	queue=Sales
17	agent—>queue	agent=Jim, queue=Sales
18	queue—>agent	queue=Sales, agent=Andy
19	agent—>review	agent=Andy
20	queue—>review	queue=Sales
21	review—>reply	
22	reply—>exit	
23	agent—>reply	agent=Andy
24	agent—>exit	agent=Andy

The following paragraphs (and Tables Table 3-19 through Table 3-22) describe the actions shown in this overview in greater detail.

In Table 3-19, Records 1, 2, and 3 are typical. Record 2 indicates that Answer Agent sent out a reply. Record 3 indicates that the reply exited the queue management system.

Table 3-19. Event Record Scenario-Part I

Record #	Event Record	Agent/Queue Assignment
1	entry—>Answer Agent	
2	Answer Agent—>reply	
3	reply—>exit	

In Table 3-20, Record 4 indicates that Answer Agent assigned the request to the Sales queue. Record 5 is the result of the same action in the queue flow time series. Record 6 shows an Agent was assigned to the request while Records 7, 8, and 10 indicate that the Agent either asked for, or was under, Supervisor review and that the Supervisor approved the reply. (If the Supervisor had not approved the reply, Record 8 would not exist.) Record 9 simply records an event that the Agent, Andy, sent a reply. Without Record 9, there would be no way to acknowledge that Andy sent the reply. Record 10 indicates that the reply exited the Queue Manager.

Table 3-20. Event Record Scenario-Part II

Record #	Event Record	Agent/Queue Assignment
4	Answer Agent—>queue	queue=Sales
5	entry—>queue	queue=Sales
6	queue—>agent	queue=Sales, agent=Andy
7	agent—>review	agent=Andy
8	review—>reply	
9	agent—>reply	agent=Andy
10	reply—>exit	

In Table 3-21, Records 11, 12, and 13 show that Agent Andy reassigned the request to Agent Jim in the Service queue. Note that reassignments to Agents in different queues require a *hop* through the Queue Manager. If Andy had assigned the request to an Agent in the Sales queue, these records would be replaced by an `agent–>agent` event record.

Table 3-21. Event Record Scenario-Part III

Record #	Event Record	Agent/Queue Assignment
11	agent—>queue	agent=Andy, queue=Service
12	queue—>queue	assignedQueue=Sales, forwardedQueue=Service
13	queue—>agent	queue=Service, agent=Jim

Table 3-22 compares Records 14 through 17 with the Records 7 through 10. In both cases, Agents were under Supervisor review when they sent a reply. However, Record 8 indicates that the Supervisor approved the reply since a reply was sent. While Record 16 indicates the Supervisor disapproved the reply and, instead, reassigned the original request to another queue. Furthermore, Record 15 indicates a flow from the queue to review while a similar record is missing from the Record 7 through 10 set. The Records 14 through 17 indicate the Agent *closed* the request with a reply and thus the request *flowed* to the review queue. In the previous case, the Agent did not close the request. Indeed, the Agent later reassigned the request.

Table 3-22. Event Record Scenario-Part IV

Record #	Event Record	Agent/Queue Assignment
14	agent—>review	agent=Jim
15	queue—>review	queue=Service
16	review—>queue	queue=Sales
17	agent—>queue	agent=Jim, queue=Sales
7	agent—>review	agent=Andy
8	review—>reply	
9	agent—>reply	agent=Andy
10	reply—>exit	

Based on the information in previous paragraphs, Records 18 through 24 of Table 3-18 should be self-explanatory.

You can do some simple analysis on this data. The total number of replies sent out is the number of `reply–>exit` records, which is 3 (Records 3, 10, and 22). The number of replies sent by human Agents is the number of `agent–>reply` records, which is 2 (Records 10 and 22). The number of Supervisor reviews is the number of `agent–>review` records, which is 3 (Records 7, 14, and 19). The number of reviews approved by Supervisor is the number of `review–>reply` records, which is 2 (Records 8 and 21).

To calculate the number of queues (not including the review or the entry queues) involved in this scenario, we count the records involved in the queue flow time series. Note that the relevant records are `queue–>queue`, `queue–>exit` and `queue–>review`. There are three such records in this scenario: Records 12, 15, and 20. By examining and grouping on the queue indicated by `assignedQueueKey`, note that two assignments were made to the Sales queue and one to the Service queue.

The section that follows discusses the columns of the `queueEvent` fact records.

queueEvent Independent Dimensions and Facts

For the following discussion, let `x–>y` represent any event record. For example, if `x–>y` represents the event `agent–>queue`, then `x` represents the Agent part of the event and `y` represents the queue part of the event.

This classification includes:

- `businessUnitKey` – Identifies the business unit in which the event occurred.
- `requestNumber`, `commentsKey`, `requestDuration`

  See "requestEvent", on page 30 for more information.
- `dateKey`, `timeOfDay`, and `eventDuration`

  `dateKey` indicates the date when the event occurred. `timeOfDay` is in seconds, since midnight, of the event time. Together, these are referred to as the *time stamp* of the event.

For all event types, the time stamp refers to the time when `x` started.

  `eventDuration` is the time difference, in seconds, between the time stamp of `x` and the time when `y` occurred.

  For events participating in a flow of a time series (see "Overlapping Time Series of Events", on page 35 for more information.), such as `agent–>queue`, the `eventDuration` also indicates how long `x` lasted. For example, in this case `eventDuration` indicates how long a request was on the Agent's desktop before reassignment to a queue.

  For events not participating in a flow, such as `agent–>reply`, the `eventDuration` indicates the elapsed time since the time stamp until `y` occurred and does not indicate the duration of `x`. For example, in this case, `eventDuration` is the amount of time a request was on the Agent's desktop before a reply was sent and does not indicate the total time the request was on the Agent's desktop since the Agent may have reassigned the request much later.
- `eventDuration (semi-additive)`

  It is safe to add the `eventDurations` of `agent–>queue` records to get an indication of how long requests stay on Agents' desktops. However, is it not correct to add the `eventDurations` of `agent–>reply` records.
- `systemDuration`

  This includes the total duration even when the message is transferred from one Business Unit to another.
- `requestStatusKey`

  This indicates the state of the request (OPEN, PEND, CLOSE). The status is CLOSE if the request was closed between `x` and `y`. Otherwise, if the request was pended at least once, then the status is PEND. Otherwise the status is OPEN.
- `eventHandlingDuration (semi-additive)`

This is the cumulative time, within the time interval of this event, that the associated request was OPEN.

Note that the request state transitions tracked by `requestEvent` are on a different overlapping time series than the events tracked by this fact table. Therefore, while a request is assigned to a queue or an Agent, it may be pended several times without being reassigned to another queue or Agent.

- `SLAValueKey`

  Every queue, at all times, has a set of values associated with it known as *Service Level Agreement* (SLA) values.

  The SLA values can change over time. The `SLAValueKey` indicates the values that were in effect at the beginning of the event. The queue is identified by `assignedQueueKey`.

- `SLATagKey`

  This is a qualitative measure of how well the system meets the SLA. For example, if a queue's SLA goal is 4 hours at the time of this event and it was on the Agent's desktop for 6 hours, the SLA of the queue was not honored by the Agent. In this case, the `SLATagKey` value is *Exceeds SLA*. To comply with the SLA value of this queue, the Agent must close the request in less than 4 hours.

  An additional qualitative measure of how close it came to exceeding the SLA is provided. If the interval is less than the *Warning* setting of the queue in effect at the time, then `SLATagKey` indicates a value of *Normal*. If it is between the *Warning* setting and the *Critical* setting, the indicated value is *Warning*, and finally if it is between the critical and goal setting, then the indicated value is *Critical*. Beyond that, the indicated value is *Exceeds SLA*.

You may have noticed that `eventDuration` is used in this example. Some applications use the `requestDuration` or `eventHandlingDuration` for their analysis.

To accommodate these differences, the `SLATagKey` is calculated by the ETL Tools such that the value of the `eventTag` of the dimension `SLATag` (see "SLATag", on page 27) is determined using `eventDuration` as previously described, the value of `requestTag` is determined using `requestDuration`, and the value of `handlingTag` is determined using `eventHandlingDuration`.

Using this structure, it is then possible for an application to choose how to determine the SLA satisfaction qualities.

- `isFirstQueue (non-additive)`

  This is set to `Y` if this is the first assignment to queue for this request; otherwise set to `N`.

  The queue is indicated by `assignedToQueue`.

- `isFirstAgent (non-additive)`

  This is set to `Y` if this is the first assignment to Agent for this request; otherwise set to `N`.

  The Agent is indicated by `assignedToAgent`.

- `assignedQueueKey`

  For events of the type `queue-->y` or `agent-->y`, this indicates the queue where the event took place. For most other events, this indicates a system queue or the `NULL` queue.

- `assignedByAgentKey`

  For events of type `agent-->Y`, this indicates the Agent who assigned the request. For other event types, this indicates the `NULL` Agent.

- `assignedAgentKey`

For events of type agent–>Y, this indicates the Agent responsible for the event. For other event types, this indicates the NULL Agent.

- forwardedQueueKey

  For x–>queue and x–>agent events, this indicates the queue into which the event eventually flowed. For other events, it indicates a system queue or the NULL queue.

- forwardedByAgentKey

  For events of type x–>agent, this indicates the Agent desktop which forwarded the request. For other event types, this indicates the NULL agent.

- forwardedAgentKey

  For events of type x–>agent, this indicates the Agent desktop into which the request flowed. For other event types, this indicates the NULL agent.

queueEvent Dimensions Depending on requestNumber

This classification includes: customerKey and inboxKey. See "requestEvent", on page 30 for more information.

queueEvent Dimensions Depending on the messageNumber

This classification includes: headerKey and actionKey. See "queueEvent Dimensions Depending on requestNumber", on page 48 for more information.

queueEvent Dimensions, Facts, & Measures Depending on Event Type

This classification includes:

- assignmentKey indicates the type of event. (See "Queue Flow records", on page 40 for more information).
- messageNumber (non–additive) – For all event types, if the assignedQueueKey or the forwardedQueueKey indicates the reply queue, then the messageNumber indicates the reply message. Otherwise, it indicates the original message associated with the request.

messageEvent

Although this table is noted as a fact table, it is not. Its primary key is the messageNumber and some other fact tables have messageNumber as the foreign key. Therefore, this is actually a dimension table. However, this table has some foreign keys to enable searches. Thus, it acts like a fact table.

This is the only "fact" table that does not have dateDim dimension or event durations.

More practically, this is a repository for the message text bodies. It includes the following:

- messageNumber – Primary key.
- businessUnitKey – Identifies the business unit in which the event occurred.
- customerKey – Indicates the customer associated with this message.
- headerKey – Set of headers associated with the message.
- actionKey – Work action and message type.

- `inboxKey` – Email inbox that received this message.
- `ETLRunKey` – run key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).
- `messageBody` – Binary large object (BLOB) which is the actual body text.
- `otherHeaders` – Other non–parsed headers associated with the message.
- `isSecure` – Set to `Y` if message is secure (which means it can be viewed through Portal); otherwise set to `N`.

agentEvent

This fact table tracks the changes to the availability of the Agents.

It is important to understand that all logon/online events, activities which are most commonly performed by real agents, are recorded in this table regardless of whether the person involved in the event is a real agent. Specifically, supervisors are included.

This table includes:

- `dateKey` – Date of the event.
- `agentKey` – Identifies the Agent.
- `timeKey` (non–additive) – Indicates the time the event occurred.
- `ETLRunKey` – run key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).
- `isLoggedIn` (non–additive) – Set to `Y` if the Agent was online during this event; otherwise set to `N`.
- `isOnline` (non–additive) – Set to `Y` if the Agent was available during this event; otherwise set to `N`.
- `eventDuration` (semi–additive) – The duration of this event.
  The event type is determined by `agentKey`. It does not make sense to aggregate the `eventDuration` of all Agents, as that comprises an overlapping time series.
- `isCurrent` (non–additive) – Set to `Y` if, at the time of ETL; otherwise set to `N`.
  This event is the most recent state of the Agent.

SLAEvent

This fact table tracks the changes of the SLA values for queues over time. This table includes:

- `businessUnitKey` – Identifies the business unit in which the event occurred.
- `dateKey` – Indicates the date of the event.
- `queueKey` – Indicates the queue to which this event applies.
- `SLAValueKey` – Indicates the SLA values in effect at the time of this event.
- `timeKey` (non–additive) – Indicates the time the event occurred.
- `ETLRunKey` – run key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).

- `eventDuration` (semi–additive) – Indicates the duration between the events where a given queue's SLA values were changed.

  The event type is determined by the `queueKey`.
- `isCurrent` (non–additive) – Set to `Y` if, at the time of ETL event, this event is the most recent setting of the queue.

KBEvent

`KBEvent` is a fact table that tracks the use of the Knowledge Base by the Intelligence Engine component.

It is important to understand that the `KBEvent` does not represent the entire Knowledge Base but only parts of it that were used to process and understand an incoming email message.

The Intelligence Engine goes through a series of steps as it processes an email message for relevant content to determine the appropriate response. During processing, it collects information and logs them to various product database tables. The ETL component transforms this data into a *star schema* centered by the `KBEvent` fact table and various dimension tables.

Processing Overview of the Intelligence Engine Component

The following description is not intended to be technically precise; various details are omitted. Instead it provides a conceptual overview which presents the contents and semantics of the `KBEvent` and its related tables.

The Intelligence Engine has two fundamental goals: comprehension; understanding the intent of the message (that is: what the message is about) and action; how to process the message based on the comprehension.

Comprehension

To understand the incoming message, the Intelligence Engine decomposes the message and reconstructs it in a form it can understand. The Intelligence Engine decomposes the message by breaking it into sets of Text Patterns (Regular Expressions) and Phrases by applying various Rules established in the Knowledge Base, and reconstructs it into sets of Features which eventually lead to determining the Intent of the message.

The Text Patterns and Phrases identified during decomposition are stored in the Datamart dimension tables `textPattern` and `phrase`, respectively. These tables only identify the logical name of the Text Patterns and Phrases, not the actual text.

The Text Pattern and Phrases found in the message are used to identify a set of Features in the message. The Features identified are stored in the Datamart dimension table `feature`.

The Intelligence Engine identifies the set of Phrases which identify a Feature. Therefore, there is a one-to-many relationship between Features identified and the set of Phrases used to identify a Feature. This relationship is stored in the Datamart group table `featureGroup`.

The identifying one-to-many relationship between a Feature and the set of Text Patterns found is not recorded by the Intelligence Engine and is not stored in the Datamart. That is, the Intelligence Engine does not identify the set of Text Patterns used to identify a Feature.

The Intelligence Engine does identify the set of Features and Text Patterns found in a message and that information is represented in the Datamart group tables `featureGroup` and `textPatternGroup` respectively.

It is from this set of identified Features that the Intelligence Engine derives a set of intents of the message which concludes the comprehension phase.

Actions

Once the Intelligence Engine identifies a set of intents for the message, it filters the set to identify the most likely or relevant intents of the message. The intents are scored (table `KBEvent`, column `intentScore`) and the top-scoring intents are used to *fire* Rules which eventually determine the Action that the Intelligence Engine takes.

All intents are stored in the Datamart dimension table `intent` and the specific Intents for a given message are identified in the `KBEvent` fact table by column `intentKey`.

A top-scoring Intent can fire one or more Rules. The Rules are stored in the Datamart dimension table `rules` and the specific rule for a given intent is identified in the `KBEvent` fact table by column `rulesKey`.

A fired Rule can cause one or more Actions to be taken or suggested by the Intelligence Engine. The name of the Actions are in the Datamart dimension table `responseAction` and the specific action for a given intent and rule are identified in the `KBEvent` fact table by column `responseActionKey`. The name of the action is typically the name of message composition template defined in the Response Library of the Contact Center and manifests itself as a suggested Response on the Agent desktop.

Like Features and Phrases, there is a one-to-many relationship between an intent and the set of fired Rules (since an intent can be used to fire multiple rules) and a one-to-many relationship between a *Rule-Intent* combination and a set of response Actions (since a given rule may cause multiple Actions). These relationships are stored directly in the `KBEvent` fact table. For example, if a given intent, say I1, fires the rules R1, R2, and R3, three rows in the fact table would exist for identifying the pairs (I1, R1), (I1, R2), (I1, R3).

Likewise, if the rule R1 for Intent I1 caused Actions A1 and A2, the Rule R2 caused Action A3, and Rule R3 caused Action A4 and A5, then five rows of the following triplicates would exist in the fact table: (I1, R1, A1), (I1, R1, A2), (I1, R2, A3), (I1, R3, A4), and (I1, R3, A5).

Response Classifications and other `KBEvent` facts

As previously discussed, Actions resulting from the firing of Rules usually generate suggested responses to the Agents.

Agents can choose to accept the suggested responses, or modify them. They may also choose to reject the suggestions and select another from the Response Library.

Responses often have associated Classification Tags that the Supervisor defines. When an Agent selects a response (from the suggested list or from the Response Library), any associated Classification Tags are logged by the email product as the reply given to an incoming email message.

All response classifications are stored in the Datamart dimension table `responseClassification`. The Classification Tags associated with the specific responses an Agent selected are stored in the Datamart group table `responseClassificationGroup` and identified in the fact table `KBEvent` in column `responseClassificationGroupKey`.

The choices an Agent makes with respect to the suggested responses are also tracked by the `KBEvent` fact table. If the agent chose to accept the suggested response (identified by `responseActionKey`), then the column `actionTaken` has the value of `Y`, otherwise `N`. If the agent

chose to edit or modify the suggested response, then the column `actionEdited` has the value `Y`, otherwise `N` to indicate the Agent accepted the suggested response without modification. The value is `NULL` if the agent chose to reject the suggested response (`actionTaken="N"`).

Three columns of the `KBEvent` apply at the message level and not at individual suggested responses.

The column `anyActionProposed` has the value `Y`, if the Intelligence Engine suggested at least one response, otherwise the value is `N`.

If at least one response was suggested, then the column `anyActionTaken` has the value `Y`, if the Agent accepted at least one of the suggestions, otherwise it has the value of `N` to indicate that the Agent rejected all suggested responses. The value is `NULL` if no responses were suggested (`anyActionProposed="N"`).

Of the suggested responses, if at least one response was edited by the Agent, then the column `anyActionEdited` has the value `Y`, otherwise the value is `N` to indicate that all the accepted responses were accepted without modification. The value is `NULL` if no responses were suggested (`anyActionProposed="N"`) or if none of the suggested responses were accepted (`anyActionTaken="N"`).

The value of `NULL` is also used for columns `anyActionEdited` and `actionEdited`, if the Intelligence Engine was unable to determine (or track) whether an Agent modified suggested responses. This occurs with older installed versions of the product.

The `KBEvent` fact table tracks the following information:

- Which Text Patterns (Regular Expressions), Features, Intents (and the score of each Intent) in the request were found in the Knowledge Base. It also groups and assigns an identification number for requests with multiple Text Patterns and Features.
- Which Rules were found in the Knowledge Base.
- Which Actions were fired.
- Which responses in the Response Library were proposed and taken or proposed and edited.

The dimensions in this table are non-additive. This table includes the following columns:

- `requestNumber` – Uniquely identifies an email request.
- `textPatternGroupKey` – Links to the `textPatternGroup` table. Identifies a set of Text Patterns recognized in this message. See “textPatternGroup”, on page 28 for more information.
- `featureGroupKey` – Links to the `featureGroup` table. Identifies a set of Features recognized in this message. See “featureGroup”, on page 20 for more information.
- `intentKey` – Links to the `intent` table. Identifies an Intent from the `featureGroup` and `textPatternGroup`. See “intent”, on page 21 for more information.
- `intentScore` – An integer between `0` (zero) and `100`. Indicates the relevance of the Intent (indicated by `intentKey`) to the request, set to `NULL` if no Intent was found in the request.
- `rulesKey` – Links to the `rules` table. Identifies a Rule fired as a result of the recognized Intent. See “rules”, on page 26 for more information.
- `responseActionKey` – Links to the `responseAction` table. Identifies the Action taken as a result of the Rule fired. See “responseAction”, on page 25 for more information.
- `responseClassificationGroupKey` – Links to the `responseClassificationGroup` table. Identifies a set of Agent response classifications associated with this message. See “responseClassificationGroup”, on page 26 for more information.

- `customerKey` – Links to the `customer` table. Identifies the customer who sent this message. See "customer", on page 18 for more information.
- `actionTaken` – Set to one of the following:

  `Y` if the action indicated by `responseActionKey` was taken by an Agent.

  `N` if the action was not taken.

  `NULL` if no action was proposed for this Rule.

  See the diagram in Figure 3-2.

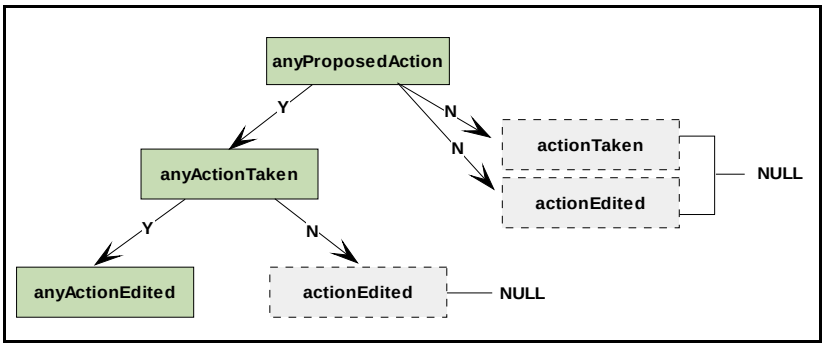

Figure 3-2. Action Algorithm

- `actionEdited` – Set to one of the following:

  `Y` if the action indicated by `responseActionKey` was taken (`actionTaken='Y'`) and edited by an Agent.

  `N` if the action was taken, but not edited.

  `NULL` if the action was not taken by an Agent, or it is unknown whether the Agent edited the proposed action.

  See the diagram in Figure 3-2.
- `anyActionProposed` – Set to one of the following:

  `Y` if at least one action was proposed by the system for some Intent and some Rule for this request (that is, `requestNumber`).

  `N` no action was proposed.

  See the diagram in Figure 3-2.
- `anyActionTaken` – Set to one of the following:

  `Y` if at least one action was proposed (`anyActionProposed='Y'`) and at least one of those actions was taken by the Agent.

  `N` if none of the proposed actions were taken.

  `NULL` if no actions were proposed by the system.

  See the diagram in Figure 3-2.
- `anyActionEdited` – Set to one of the following:

  `Y` if, of the taken actions (`anyActionTaken='Y'`), at least one was edited by an Agent.

  `N` if none of the taken actions were edited.

  `NULL` if none of the proposed actions were taken (`anyActionTaken='N'`) or if it is unknown whether the Agent edited any actions.

  See the diagram in Figure 3-2.

Chapter 4

Web Datamart Specifics

Introduction

This chapter describes the Web portions of the Datamart data model. The intent is to provide sufficient information to enable you to use the analytic tools and reports of your choice. We assume the reader is familiar with database concepts and the Web channel.

The portions of the Datamart populated with Web data use Summary Fact tables as well as Group, Fixed, and Varying Dimension tables. See "Functional Overview", on page 6 for more information about these types of tables.

We have supplied Summary Views for the Datamart to facilitate reporting, see "Summary Views", on page 74 for more information.

Web Fact/Dimension Matrix

The Datamart contains three Summary Fact tables, fifteen Dimension tables, and nine Group (Bridge) tables. This section provides an overview of the fact and dimension tables, showing the relationships between them.

Table 4-1 shows the dimensions associated with each fact table. Fact tables are listed across the top row of the matrix, and dimensions appear in the left–most column.

Table 4-1. Fact/Dimension Matrix

Facts ⇒ **Dimensions ⇓**	requestSummary Fact Table	situationSummary Fact Table	responseSummary Fact Table
additionalInputField			•
additionalInputField			•
additionalInputFieldGroup			•
chatAction			•
chatActionGroup			•

Table 4-1. Fact/Dimension Matrix

Facts ⇒ **Dimensions ⇓**	requestSummary Fact Table	situationSummary Fact Table	responseSummary Fact Table
customAction			•
customActionGroup			•
customer	•		
dateDim	•		•
destinationAction			•
destinationActionGroup			•
emailAction			•
emailActionGroup			•
explanationAction			•
explanationActionGroup			•
initialInputField	•		
initialInputFieldGroup	•		
intent	•	•	
intentGroup	•		
KBGroup	•		
KBInformation	•		
language	•		
response			•
situation		•	
sourcePage	•		
timeDim	•		•

About Tables in the Schema

The fact tables are the logical center of the Datamart, as well as the physical center of the stars in the Datamart schema. These tables contain the information of the most interest to the report designer and are used in almost all reports.

The ETL Tools extract, transfer, and load data from the source transactional database into the Datamart model. This includes data gathered as Web requests from customers.

Web Processing Simplified

Customers use the Web channel during a given Web session by asking a question in a frame (called the Initial Input Form in Analytics). Using Natural Language Processing (NLP), the Intelligence Engine analyzes the question or information, tries to discern the intent of the request, and returns with one of three conclusions for the customer:

- There is one determined intent with corresponding responses or actions.
- There is not enough information to determine the intent (called an ambiguous intent in Analytics), so the Intelligence Engine asks for more information or escalates the request to a service channel.
- There are two or more discerned intents, so the Intelligence Engine displays a page with clarifying questions and asks the customer to choose one possibility.

In the first case, because the Intelligence Engine has sufficient information, it determines the intent. In the other two cases, the Intelligence Engine takes action according to rules provided by the Knowledge Base used when processing the request. At any time in a Web session, the customer can interrupt the process or choose to back up to an earlier point in the conversation and choose another path.

How Analytics Views a Web Session

Analytics reporting allows you to look at portions of Web interaction with a customer. It views interactions within a given time frame and breaks down the Web customer interaction into identifiable segments.

Time Sequences—Date/Time Ranges and Durations

Various concepts of time apply in analyzing Web data. A Web request includes all interactions related to an inquiry starting with the Initial Input inbound interaction. Analytics requires a date/time range to identify the period of time to consider when analyzing Web data.

Analytics uses Durations to represent elapsed time in seconds. The total duration is the difference between the start and end times of a conversation or request. A Duration is made up of either think time or process time. Think time is the time when the customer thinks about or prepares the next inbound interaction. Process time is the time when Web processes the inbound interaction and sends an outbound interaction. In all cases thinkDuration + processDuration = totalDuration.

Conversations Initiated with Web

A conversation begins when the customer first submits an inquiry from an Initial Input form (commonly referred to as the source page). The conversation ends when the customer disconnects or times out, if there is no escalation of the conversation.

A conversation can only be initiated via an inquiry. Each inquiry initiates a new request (and terminates any previous request). So, a conversation must have at least one request. A conversation is comprised of one or more contiguous, non-overlapping requests. See "Inquiries", on page 59 and "Requests", on page 58. Therefore, the extent of the conversation is the extent of all the requests for that Web session—from the beginning of the first request to the end of the last one. Any of these interactions may be a part of a conversation:

- Initial input submission
- Situation selection form
- Situation selection submission
- Additional input form
- Additional input submission
- Actions (in various forms)
- Email submission
- Email confirmation
- Feedback confirmation

The customer has control of the conversation and can always interrupt the Web session or choose to back up to an earlier step in the conversation without proceeding to the next step.

Requests

The request is the entire set of interactions related to an inquiry. So, a request begins with an inquiry, and includes all of the exchanges between the customer and the Intelligence Engine which pertain to that inquiry. Because every inquiry has a situation, a request is also comprised of one or more situations. (See "Inquiries", on page 59 and "Situations", on page 59.)

Pursued, Completed, or Abandoned Requests

In the course of interacting with a customer about a request, the Intelligence Engine may consider the request in any of several states:

- As being pursued.
- As being completed.
- As being abandoned.

Each request is either pursued or it is not. A request is considered pursued when at least one Situation is used. A request is not considered pursued if no Situations are used.

A Request is either completed or it is not. Once each request is pursued, it has at least one used situation. If any of the used situations was completed, the request is considered complete. A request is considered incomplete even if it is pursued, but *none* of its situations is used.

When a request is abandoned, it indicates whether the Intelligence Engine fired the action (this completes the response) for that request. The customer can abandon a request in either of two ways:

- If the customer never chooses any Situation from the Situation Selection form.
- If the customer does not supply needed information on the Additional Input form.

Inquiries

An inquiry is a question or issue raised by a customer in a Web session. The customer submits the inquiry during a conversation using an Initial Input form (in Analytics this data is captured in the initialInputField table). The Intelligence Engine handles an inquiry as a single inbound transaction which leads to further interactions. The typical Web installation includes one input field named Question.

The use of an input field is not required to run the Web channel.

Intents

An intent is the determined meaning of an inquiry. The Intelligence Engine determines the intent of the inquiry using natural language processing (NLP) and a Knowledge Base. Ideally the program finds one intent for each distinct topic raised in the inquiry. It is possible that the Intelligence Engine will not match any intent to the inquiry. The third possibility is that the Intelligence Engine matches many intents because either the customer raised many topics or the inquiry is too ambiguous to determine the intent.

Situations

A Situation is something that the Intelligence Engine can do. A situation is not the action itself, but the possible actions that the Intelligence Engine can take. The Intelligence Engine determines one or more appropriate situations for each inquiry depending upon the determined meaning of the inquiry. If there is only one situation, the Intelligence Engine can respond to the customer. If there is more than one situation, the Intelligence Engine will ask for more information or escalate the inquiry to a service channel.

Some situations match the inquiry and determined intent. For example, here is a typical inquiry/intent/situation scenario where the situation matches the inquiry and the Intelligence Engine can respond to the inquiry:

where is your San Rafael store? = the inquiry
store location intent = the intent
post store addresses = the situation (what the Intelligence Engine can do)

Posting the store address is directly related to the inquiry of where is your San Rafael store.

Other situations may not match the inquiry. As a result of determining the inquiry is ambiguous, the Intelligence Engine might return a mismatched scenario like this:

I want my money = the inquiry
need more information = the intent

ask for more information = the first situation
or escalate to a service agent = the second situation

In this scenario, asking for more information or sending the customer to a service channel does not directly match the inquiry demanding money. The Intelligence Engine has determined it needs more information to resolve the matter.

Used and Completed Situations

Just because the Intelligence Engine presents a situation to a customer does not mean that the customer will use it. Each Situation is either classified as used or not used. A Situation is considered used when a customer pursues a response from a Situation Selection form or the Intelligence Engine automatically pursues a response because it determined only one situation matched the request. A Situation is unused when the customer or the Intelligence Engine does not pursue the response.

Each used Situation is either completed, or it is not. If the Situation is used, it has at least one response. The situation is considered complete when it is both used and its response is also complete. An incomplete Situation is one that is used but where all the situations were abandoned by the customer.

Responses

A response is a single use of a given situation. Consider the ambiguous inquiry where the customer can choose between two situations. If the customer chooses one situation, the Intelligence Engine generates a response. If the customer chooses that same situation a second time, the Intelligence Engine generates another response. Even though those responses are identical, they occurred at two different times, and therefore are different responses.

Now consider the situation where the Intelligence Engine needed more information and asks for it in an Additional Input form. That is a response also—asking for the information. If the customer quits right there, the response was fired but never completed. On the other hand, if the customer supplied the needed data, the Intelligence Engine would act because the response is now complete. The Intelligence Engine will always act upon a completed response (see “Actions”, on page 61). A completed response occurs in these cases:

- When the Intelligence Engine automatically selects a situation (such as the situation example where the Intelligence Engine posted the store address).
- When the customer selects a situation presented in a Situation Selection form (for example, when asked, the customer chooses the situation closest to their inquiry).

Customers might not complete the Additional Input form. Instead they can back track to a prior form, then continue from that form on a different path. If they back up to a Situation form, they can choose another situation (or even the same situation again—it will still be the same inquiry). If they back up to the Initial Input form, they can submit a new inquiry and, therefore begin a new request.

Completed or Aborted Responses

Each response is either completed or aborted. The Intelligence Engine recognizes a situation as completed if it fires at least one action. It recognizes a situation as aborted when the Intelligence Engine fires no action. An aborted response only occurs when the customer is presented an

Additional Input Form but does not complete it. Instead, the customer must back out (or time out) of the Additional Input Form for a response.

Actions

An action is a part of the answer provided to the customer. The Intelligence Engine fires an action when it has a complete response. There can be more than one action for a given reply. The types of actions that can be invoked by the Intelligence Engine include:

- Presenting an explanation in an output field displayed in the customer's Web browser.
- Routing the customer automatically to another Web page.
- Escalating the conversation to a Customer Service Representative (CSR) via email using a customer service email address.
- Escalating the conversation to a CSR via a chat session using a customer service Web page.
- Firing a custom action as defined in the Knowledge Base which may include email notifications or invocations of other programs.

Summary of the Segments of a Conversation

The customer can end up with any number of actions for a given question depending upon the choices the customer or the Intelligence Engine makes during an inquiry. Many segments of a conversation overlap one another:

- A conversation begins with an inquiry and has one or more requests.
- Each request has one or more situations.
- Each situation has any number of responses.
- Each response has any number of actions.

A Simple Customer Conversation

In this case, a customer submits a question from an Initial Input form during a Web session. The form has a single input field that contains the following text:

> "I want to buy a PC."

The Intelligence Engine establishes a conversation for the customer (and terminates any prior request) to track the new inquiry. Using NLP and an assigned Knowledge Base, the Intelligence Engine processes the inquiry and determines two intents:

- Buy personal computer
- Buy power cord

The Knowledge Base generates two possible situations for this request—one for each intent. Since the Intelligence Engine needs more information to complete the process, it presents a Situation Selection form where the customer is expected to select the desired situation. The situations are uniquely identified with the following text:

1. "Would you like to buy a personal computer from our online store?"

2. “Would you like to locate a computer parts store near you?”

The customer then chooses the desired situation by clicking the applicable text (which is a hotlink). If the customer chooses the first situation (assuming no further information is required from the customer), the Intelligence Engine routes the customer to the company’s online store. In this case, the response is fully defined and action is taken immediately. The response, in this case, consists of the single action which is the routing of the customer to the Web site for the company’s online store.

If the customer chooses the second situation, the Intelligence Engine may present an Additional Input form where customers are asked to enter their zip code. When the customer submits the zip code, the Intelligence Engine has all the information it needs to complete the response (buying a power cord from a store in the specified zip code) and fire the action. The Knowledge Base specifies two actions for this situation, The first action displays the locations of the company’s physical stores in the specified zip code. The second is a custom action which notifies the manager of the company’s online store via email that the Intelligence Engine has referred another customer to a real-world store to buy a power cord.

Customer Feedback Values

The customer may or may not provide feedback about the value of the response provided by the Intelligence Engine. Typically, the customer assigns a value by clicking a radio button associated with a value (the value is an integer). The range of possible feedback values is defined by the Intelligence Engine implementation.

Web Fact Tables

Analytics uses Summary fact tables in the Datamart schema for Web. Each Summary fact table specifies a start and end time to define the period each summarization spans. Each record in a fact table includes many facts about the Web requests, situations, or responses. The facts are either pointers to records in the dimension tables or attributes (typically numbers) that directly describe the subject of the table.

requestSummary Fact Table

This summary fact table describes a summary of the usage of an individual request.

The table includes:

- `requestNumber` – primary key; This is the unique number of the request from the Web transactional database.
- `languageKey` – Identifies the language of the request as determined by analysis done by the Intelligence Engine.
- `ETLRunKey` – run key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).
- `conversationNumber` – foreign key; The unique number of the Web conversation.

- `conversationCustomerkey` – foreign key; The unique number of the conversation for the individual customer during the current Web session.
- `startDateKey` – foreign key; The date the conversation began.
- `startTimeKey` – foreign key; The time the conversation began.
- `endDateKey` – foreign key; The date the conversation ended.
- `endTimeKey` – foreign key; The time the conversation ended.
- `sourcePageKey` – foreign key; The source Page is the initial input form used by the customer to submit a Web inquiry. This record points to the sourcePage dimension table record for an individual request.
- `initialInputFieldGroup` – bridge key; This data points to the group which identifies the names of fields used in the initial input when the customers make their first inquiry.
- `KBGroup` – bridge key; This data points to the group with the names of the Knowledge Bases used in processing the request.
- `intentGroup` – bridge key; This data points to the group with the names of the intents that matched the request.
- `totalDuration` – fact or measure; int; Total lapse time of the conversation in seconds. The totalDuration is equal to the thinkDuration + the processDuration.
- `thinkDuration` – fact or measure; int; The lapsed time in seconds when the customer thinks or prepares the next inbound interaction.
- `processDuration` – fact or measure; int; The lapsed time in seconds the Intelligence Engine takes to process the inbound interaction from the customer and send the outbound interaction back.
- `numberofInitialInputFields` – fact or measure; smallint; The number of initial input fields on the Initial Input form.
- `numberOfKBs` – fact or measure; smallint; The number of Knowledge Bases used in processing the inquiry.
- `numberOfAnalyzedInitialFields` – fact or measure; smallint; The number of initial input fields that were analyzed using NLP.
- `numberOfIntents` – fact or measure; smallint; The number of intents in the Knowledge Base.
- `numberOfIMatchedIntents` – fact or measure; smallint; The number intents that matched the inquiry.
- `numberOfSituations` – fact or measure; smallint; The number of situations for a given request. This value is always greater than zero. It indicates whether the inquiry was ambiguous. If there is only one situation, the request is not ambiguous. If there is more than one situation, the request is ambiguous.
- `situations` – fact or measure; smallint; The number of used Situations for a given request. If the value is greater than zero this record indicates whether the request was pursued. If the value equals zero, the request was not pursued.
- `numberOfUnusedSituations` – fact or measure; smallint; The number of incomplete situations (the situation was not used) for a given request.
- `numberOfCompletedSituations` – fact or measure; smallint; The number of used situations that resulted in actions. This value along with the numberOfUsedSituations indicates whether the response was completed. If the value is greater than zero, the response is complete. If the value is zero, the response is considered aborted.

- averageFeedbackActualValue – fact or measure; float; The average value of feedback from the customer.

responseSummary Fact Table

This summary fact table describes a summary of the usage of an individual response. A record exists for each response generated. If a response needs additional information from the customer and the customer does not supply it, the usage may not include an action.

The table includes:

- responseNumber – primary key.
- ETLRunKey – primary key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).
- conversationNumber – foreign key; The unique number of the Intelligence Engine conversation.
- requestNumber – foreign key; This is the unique number of the request from the Web transactional database.
- situationNumber – foreign key; The identifying number of the situation used as a part of the response.
- responseKey – foreign key; The key value which points to the response dimension table.
- startDateKey – foreign key; The value that points to the record which defines the starting date of the response.
- startTimeKey – foreign key; The value that points to the record which defines the starting time of the response.
- actionDateKey – foreign key; The date the action was taken.
- actionTimeKey – foreign key; The time the action was taken.
- additionalInputFieldGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Additional Input Fields.
- explanationActionGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Explanation actions.
- destinationActionGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Destination actions.
- emailActionGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Email actions.
- chatActionGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Chat actions.
- customerActionGroup – bridge key; This key value points to the Group table which identifies and points to the data regarding the Custom actions.
- numberOfAdditionalInputFields – fact or measure; smallint; The number of Additional Input fields used in the interaction.
- numberOfActions – fact or measure; smallint; This value indicates whether the response was completed (greater than zero) or aborted (zero).
- numberOfEscalations – fact or measure; smallint; The total number of escalations for a response.

- numberOfEscalationnsToEmail – fact or measure; smallint; The number of escalations via email for a response.
- numberOfEscalationsToChat – fact or measure; smallint; The number of escalations via a chat session for a response.
- numberOfAnswers – fact or measure; smallint; The total number of the class of actions which include an explanation presented as words or a destination presented as a Web page.
- numberOfAnswersByDestination – fact or measure; smallint; The number of the class of actions which include Answers as a destination presented as a Web page.
- numberOfAnswersByExplanation – fact or measure; smallint; The number of the class of actions which include Answers as an explanation presented as words.
- numberOfCustomActions – fact or measure; smallint; The total number of the class of actions which include custom actions as defined in the Knowledge Base.
- feedbackActualValue – fact or measure; smallint; The actual value the customer entered regarding satisfaction with the response.

situationSummary Fact Table

This summary fact table describes the summary of the usage of an individual situation for each request. If the Intelligence Engine determines one intent for the request, the request is considered unambiguous and has one identified situation. Otherwise there can be many situations which the Intelligence Engine presents to the customer on the Situation Selection form.

The table includes:

- situationNumber – primary key; The unique number for this situation.
- ETLRunKey – primary key; This key identifies the source database that supplied the data for the ETL as well as facts about the run (such as time and date).
- conversationNumber – foreign key; Points to a table that identifies the unique number for the conversation related to this situation.
- requestNumber – foreign key; Points to a table that identifies the unique number for the request related to this situation.
- situationKey – foreign key; The value that points to a table that identifies the unique situation.
- intentKey – foreign key; Points to a table that identifies the intent that best matches the situation. Generally, the Knowledge Base uses a single situation for one intent. If the Intelligence Engine doesn't match the intent with a situation, Analytics records lists no intent here.
- numberOfResponses – fact or measure smallint; The value that indicates whether the Situation was used (greater than zero) or not (equal to zero).
- numberOfAbortedResponses – fact or measure smallint; The number of responses that were aborted during the customer interaction.
- numberOfCompletedResponses – fact or measure smallint; This value indicates whether the Situation was completed (greater than zero) or not (equal to zero).

Web Dimension Tables

This section describes each Group and Dimension table. As necessary, it includes the database column types, the possible values of the columns, and sample column values. See "Functional Overview", on page 6 for more information.

additionalInputField

Related fact tables: responseSummary

This table is a varying-attribute dimension table. It defines the individual additional input field names (as defined in the Knowledge Base), regardless of the usage of the actual fields. The actual usage is defined in other tables that reference these definitions.

The table includes:

- `additionalInputFieldKey` – primary key; The key value to which tables using the additionalInputFieldKey point.
- `name` – string; A unique name for the Additional Input field on the Additional Input form where the customer enters additional information needed to determine the intent of the inquiry.

additionalInputFieldGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the additional input fields used when soliciting additional information so the Intelligence Engine can send a response to the customer.

Individual fields are specific to a single Knowledge Base.

This table includes:

- `additionalInputFieldGroup` – bridge key; The individual names of the Additional Input Fields.
- `KBInformationKey` – foreign key; This value points to the KBInformation table which identifies Knowledge Bases that are used in the Datamart.
- `additionalInputFieldKey` – foreign key; This value points to the additionalInputField table which defines the individual additional input field names.
- `value` – dimension attribute; This value is a string.

chatAction

Related fact tables: responseSummary

This table is an unshared varying entity dimension table defining chat actions. Chat actions are actions the Intelligence Engine takes when it escalates the customer interaction to a chat session (with a CSR via a Web page).

Individual actions are specific to a single Knowledge Base.

This table includes:

- `chatActionKey` – primary key; The key value to which other tables point when referring to the chatAction table.
- `KBInformationKey` – foreign key; The key value that identifies which record to read in regarding the Knowledge Base which fired the action.
- `URL` – dimension attribute; the value is a string; The address of the Web page used for this particular escalation.

chatActionGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the chat actions for a response. These are considered "escalations" because the customer is put in contact with a CSR.

This table includes:

- `chatActionGroup` – bridge key; The individual names of the chat actions.
- `chatActionKey` – foreign key; The key value which points to records defining the chatActions.

customAction

Related fact tables: responseSummary

A varying entity (unshared) dimension table defining custom actions.

Individual actions are specific to a single Knowledge Base.

This table includes:

- `customActionKey` – primary key; The key value which tables using the customActionKey point to.
- `KBInformationKey` – foreign key; The key value that identifies which record to read regarding the Knowledge Base which fired the action.
- `name` – dimension attribute; a string value; The name of the custom action as defined in the Knowledge Base.

customActionGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the custom actions for a response.

This table includes:

- `customActionGroup` – bridge key; The individual names of the custom actions.
- `customActionKey` – foreign key; The key value that points to the tables defining the individual custom actions.

customer

See "customer", on page 18

dateDim

See "dateDim", on page 18

destinationAction

Related fact tables: responseSummary

An unshared varying entity dimension table which defines destination actions. These are Web pages to which the Intelligence Engine routes customers as a result of response actions.

Individual actions are specific to a single Knowledge Base.

This table includes:

- `destinationActionKey` – primary key; The key value which tables pointing to the destinationActionKey point to.
- `KBInformationKey` – foreign key; The key value that identifies which record to read regarding the Knowledge Base which fired the action.
- `URL` – dimension attribute; a string value; The URL of the Web page where customers are sent as a result of an action.

destinationActionGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the destination actions for a response. These are considered "answers" to a customer's inquiry because they directly address the inquiry.

This table includes:

- `destinationActionGroup` – bridge key; Identifies the individual destination actions for a response.
- `destinationActionKey` – foreign key; The key value points to the destinationActionKey in the destinationAction table.

emailAction

Related fact tables: responseSummary

An unshared varying entity dimension table identifying email actions.

Individual actions are specific to a single Knowledge Base.

This table includes:

- `emailActionKey` – primary key; The key value which tables pointing to the emailActionKey point to.
- `KBInformationKey` – foreign key; The key value points to the record regarding the Knowledge Base which fired the email action.
- `ToAddress` – dimension attribute; a string value; The email address which the action sends the email to.

emailActionGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the email actions for a response. These are considered "escalations" because the customer is put in contact with a CSR.

This table includes:

- `emailActionGroup` – bridge key;
- `emailActionKey` – foreign key; The key value which points to the emailActionKey in the emailAction table.
- `FromAddress` – dimension attribute; a string value; The email address which the action sends the email from.

explanationAction

Related fact tables: responseSummary

An unshared varying entity dimension table defining explanation actions.

Individual actions are specific to a single Knowledge Base.

This table includes:

- `explanationActionKey` – primary key; The key value which tables pointing to the explanationAction points to.
- `KBInformationKey` – foreign key; The key value that points to records regarding the Knowledge Base which fired the action.
- `explanation` – dimension attribute, a string value; The explanation to be displayed in the customer's browser.

explanationActionGroup

Related fact tables: responseSummary

This table is a Group or Bridge table that identifies the explanation actions for a response. These are considered "answers" because they directly address the inquiry.

This table includes:

- `explanationActionGroup` – bridge key;
- `explanationActionKey` – foreign key; The key value which points to the explanationActionKey in the explanationAction table.

initialInputField

Related fact tables: requestSummary

A varying attribute dimension table defining initial input field names that are used in the Datamart. This table defines individual initial input field names (as defined in the Knowledge Base), regardless of their usage. Their actual usage is defined in other tables that reference these definitions.

Whether the field is analyzed by the NLP is defined in the Knowledge Base. However, over time, that definition can be changed. Therefore, whether analysis is done during different usages of a field with a given field may be different. Since the report designer is only interested in whether the field is analyzed at the time of usage, we have no need of an entity dimension for input fields. (The bridge table describing the field usage is sufficient.)

This table includes:

- `initialInputFieldKey` – primary key; The key value which tables pointing to the initialInputField table points to.
- `name` – dimension attribute; the value is a string; The individual name of the Initial Input field.

initialInputFieldGroup

Related fact tables: requestSummary

This table is a Group or Bridge table that identifies the initial input fields comprising an inquiry.

This table includes:

- `initialInputFieldGroup` – bridge key;
- `initialInputFieldKey` – foreign key; The key value that points to the initialInputFieldKey in the initialInputField table.
- `isAnalyzed` – dimension attribute; a boolean value; Identifies if the field is analyzed as a part of determining the intent of the inquiry.
- `value` – dimension attribute; a string value;

intent

See "intent", on page 21

intentGroup

Related fact tables: requestSummary

This table is a Group or Bridge table that identifies the intents related to a request. This includes the intents that were found to match the request's initial input as well as the intents that were determined to match each situation that is used (whether manually selected by the customer or automatically selected by the Intelligence Engine).

Individual intents are specific to a single Knowledge Base.

Scores are recorded as floating point numbers between 0 and 1 where 0 indicates a complete mismatch and 1 indicates a perfect match. No score is recorded for an intent that did not match the initial input. Note that the intents which match any situation for the request (as recorded in the responseSummary table) are recorded here and may not have matched the initial input. No score is recorded for such intents.

This table includes:

- `intentGroup` – bridge key;
- `KBInformationKey` – foreign key; The key value that points to the record regarding the Knowledge Base which fired the intent.
- `intentKey` – foreign key; The key value which points to the intentKey in the intent table.
- `score` – dimension attribute; float; The score the Intelligence Engine gave this intent when determining the intent of the inquiry.

KBGroup

Related fact tables: requestSummary

This table is a Group or Bridge table that identifies the Knowledge Bases used in the Datamart.

This table includes:

- `KBGroup` – bridge key;
- `KBInformationKey` – foreign key; The key value that points to the record regarding the Knowledge Base which fired the intent.

KBInformation

"KBInformation", on page 21

response

Related fact tables: responseSummary

This table is an unshared varying entity dimension table. It defines the responses (as specified in the Knowledge Base) regardless of their usage.

This table includes:

- `responseKey` – primary key; The key value that identifies the response.
- `KBInformationKey` – foreign key; The key value that identifies which record to read regarding the Knowledge Base which analyzed the inquiry.
- `name` – dimension attribute; the value is a string; A unique name for the response as defined in the Knowledge Base.

situation

Related fact tables: situationSummary

This table is an unshared varying dimension table. It identifies the situations as defined by the Knowledge Bases regardless of their usage.

This table includes:

- `situationKey` – primary key; The unique number that identifies this individual situation.
- `KBInformationKey` – foreign key; The key value that identifies which record to read regarding the Knowledge Base which analyzed the inquiry.
- `situationName` – dimension attribute; The value is a string; a unique name for the situation as defined in the Knowledge Base.
- `disambiguationText` – dimension attribute; a string value; The text displayed in the customer's browser asking the customer to indicate which situation matches the inquiry.

sourcePage

Related fact tables: requestSummary

An unshared varying entity dimension table defining the source pages where customers submit their inquiries. These are the Web pages that are configured as Initial Input forms.

This table includes:

- `sourcePageKey` – primary key; The key value which tables referencing the sourcePage table point to.
- `name` – dimension attribute; the value is a string; A unique name for the Source Page as defined in the Knowledge Base.

timeDim

See "timeDim", on page 29.

Summary Views

Analytics includes five summary views:

- actionList
- converstationSummary
- allRequestSummary
- mailRequestSummary
- webRequestSummary

The views eliminate the need to use complex SQL or UNION logic that is not supported directly in reporting tools such as Crystal Reports. It is important to understand that this view is provided as a convenience to the user. It is likely to underperform when used in unconstrained queries. Therefore, we do not require the report designer to utilize this view in any manner.

ActionList View

An action is a part of the reply provided to the customer. The Intelligence Engine fires an action when it has a completed response (when either it has determined the intent of the request or the customer has supplied enough information for the Intelligence Engine to determine the intent). The Intelligence Engine will always act upon a completed response. If the response is not complete (as when the customer abandons a request), the Intelligence Engine will not take any actions for the response. There can be more that one action for a given response. The types of actions that can be invoked by the Intelligence Engine include:

- Presenting an explanation in an output field displayed in the customer's Web browser.
- Routing the customer automatically to some form of destination such as another Web page.
- Escalating the conversation to a Customer Service Representative (CSR) via email using a customer service email address.
- Escalating the conversation to a CSR via a chat session using a customer service Web page.
- Firing a custom action as defined in the Knowledge Base which may include email notifications or invocations of other programs.

The Rows of the actionList View

Each row of the actionList represents an individual action. Every action recorded in the Datamart is listed in the View.

The Columns of the actionList View

The columns of the View include:

- **responseNumber** – The number of the response that fired the action.
- **actionClass** – The general classification of the action: the Intelligence Engine, Escalation, or Custom.
- **actionType** – The specific type of the action (which is class-specific).
- **actionDescription** – A description of the specific action (which is type-specific).

The three classes for actionType reflect the action the Intelligence Engine takes:

- **The Intelligence Engine** – The Intelligence Engine-class actions can either be an explanation presented as words in the customer's browser or a destination presented in the form of Web page.
- **Escalation** – Escalation-class actions direct the interactions to a service channel via an email or chat session.
- **Custom** – Custom-class actions are defined in the Knowledge Base and can include email notifications or invocations of other programs.

There are five types of actionDescription:

- **Explanation** – The text presented to the customer.
- **Destination** – The URL to which the customer was routed.
- **Email** – The address to which the request was sent.
- **Chat** – The URL at which the chat was initiated.
- **Custom** – The name (from Knowledge Base) of the custom action.

conversationSummary View

It is important to understand that this view is provided as a convenience to the user. It is likely to underperform when used in unconstrained queries. Therefore, we do not require the report designer to utilize this view in any manner.

A conversation is a thread of related interactions with one customer, that is, the set of inquiries made by the customer coupled with the responses to those inquiries generated by agents (or Brightware on behalf of those agents). A conversation is initiated upon receipt of a request which is not part of an existing conversation. Such a request can be received through any one of the channels listed in the channel dimension table (Web, Email, Portal, or Chat). A conversation is, therefore, comprised of one, or more, requests which are all related to each other.

The Rows of the conversationSummary View

Each row of the conversationSummary view represents an individual conversation. Every conversation recorded in the Datamart is listed in the view.

The Columns of the conversationSummary View

The following are definitions for each column in the view:

- **conversationNumbe**r – The number that uniquely identifies the conversation in the Datamart. This number is referenced by conversationNumber in other tables including requestSummary, situationSummary, and responseSummary.
- **startDateKey** – The key of the dateDim record that describes the date on which the conversation started.
- **startTimeKey** – The key of the timeDim record that describes the time (on the start date) when the conversation started.
- **endDateKey** – The key of the dateDim record that describes the date on which the conversation ended.

- **endTimeKey** – The key of the timeDim record that describes the time (on the end date) when the conversation ended.
- **totalDuration** – The total number of seconds for which the conversation lasted. For example, the number of seconds that elapsed from the start time on the start date until the end time on the end date.
- **numberOfRequests** – The number of requests that comprise this conversation. (Each is recorded in the requestSummary table.)
- **customerKey** – The key of the customer record that describes the customer for which the conversation was invoked. Note that, even though we assume a conversation is invoked for a single customer, it is possible that the customer may identify him/herself differently during the conversation and, therefore, generate multiple customer records. In such cases, we utilize a single such customer record (which is chosen randomly). If there is no customer associated with any request in the conversation, this column is null.

allRequestSummary View

A request is an inquiry made by a user and all of its related interactions (including replies and prompts for clarification).

The Rows of the allRequestSummary View

Each row of the allRequestSummary view represents an individual request, whether it is a mail request or a web request. Every request recorded in the Datamart is listed in the view.

The Columns of the allRequestSummary View

The columns of the allRequestSummary view include:

- **requestNumber** – The number that uniquely identifies the request in the Datamart. This number is referenced by requestNumber in other tables and views.
- **conversationNumber** – The number that identifies the conversation of which this request is part.
- **requestStatusKey** – The status of the request (open, closed, etc.)
- **channelKey** – Identifies the channel through which the request was received.
- **languageKey** – Identifies the language of the inquiry for the request as determined by Brightware during analysis of the inquiry.
- **startDateKey** – Identifies the date on which the request was opened.
- **startTimeKey** – Identifies the time of day at which the request was opened.
- **endDateKey** – Identifies the date on which the request was closed.
- **endTimeKey** – Identifies the time of day at which the request was closed.
- **totalDuration** – The total number of seconds the request was active. (This is the elapsed time between the start date/time and the end date/time.)
- **conversationCustomerKey** – Identifies the customer for which the conversation was initiated.
- **conversationBusinessUnitKey** – Identifies the initial business unit.

mailRequestSummary View

A mail request is a request which is received by the mail-handling part of Brightware. This includes all requests received through the Portal, Chat, and Email channels.

The Rows of the mailRequestSummary View

Each row of the mailRequestSummary view represents an individual mail request. Every mail request recorded in the Datamart is listed in the view.

The Columns of the mailRequestSummary View

The columns of the mailRequestSummary view are identical to those of the allRequestSummary view.

webRequestSummary View

A web request is a request which is received through the Web channel.

The Rows of the webRequestSummary View

Each row of the webRequestSummary view represents an individual web request. Every web request recorded in the Datamart is listed in the view.

The Columns of the webRequestSummary View

The columns of the webRequestSummary view are identical to those of the allRequestSummary view.

Appendix

A

Data Model Diagram

Entity Relationship for the Datamart

The first page is the “global” view of the diagram. The subsequent pages present an expanded view of the same diagram.

Each entity (box) in the diagram indicates a table in the Datamart and the lines connecting them indicate a primary/foreign key relationship.

This diagram is part of the documentation and is not intended to be used in isolation. edocs reserves the right to change the Datamart model or its presentation at anytime in the future without prior notice.

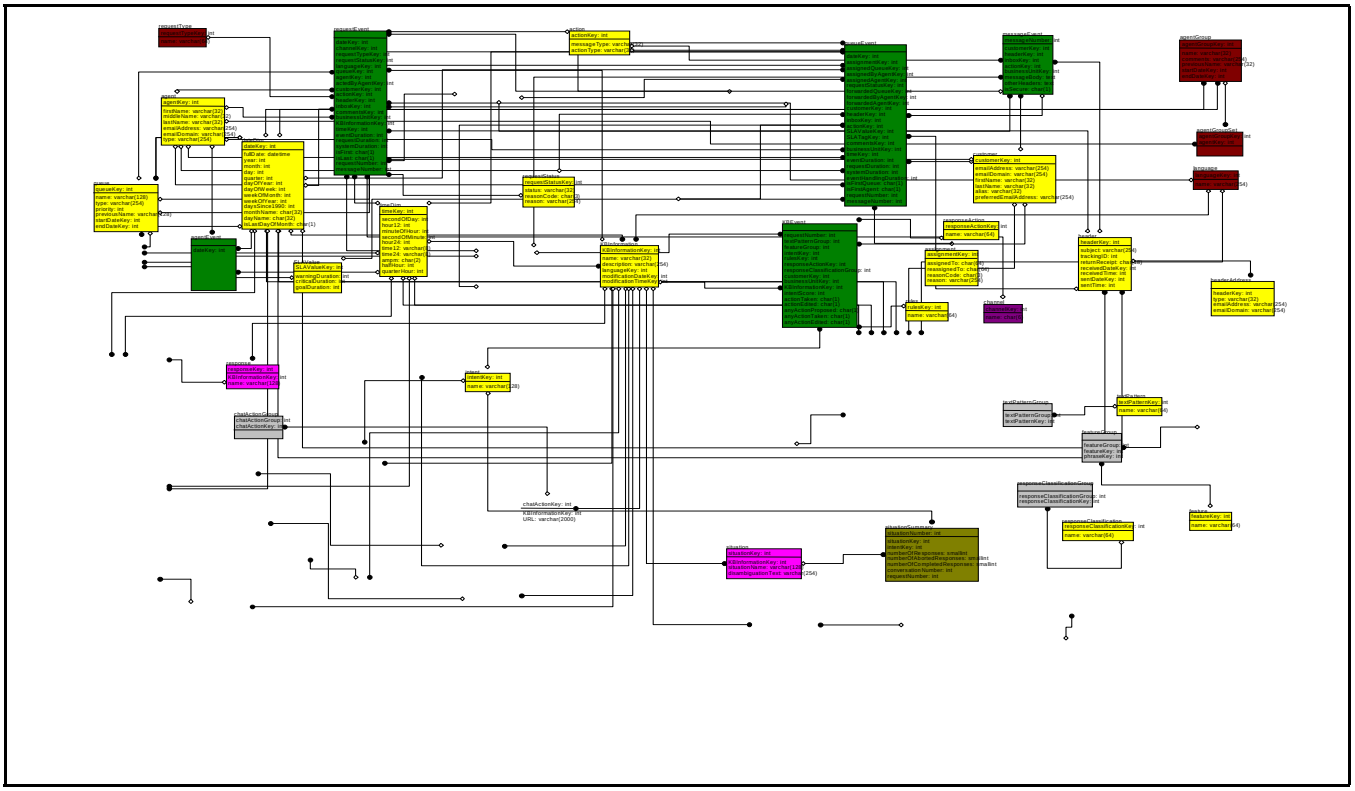

Figure A-1. The Big Picture

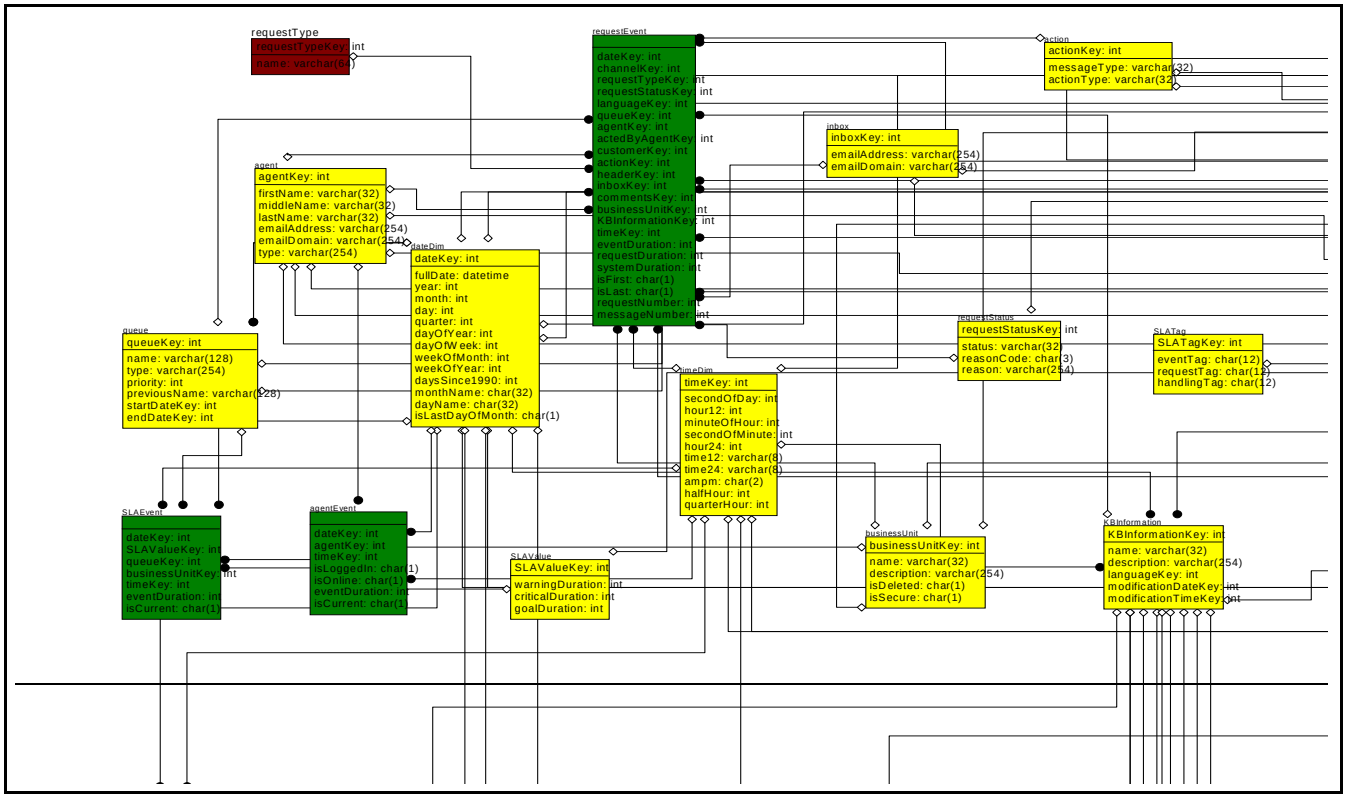

Figure A-2. Upper Left Quadrant

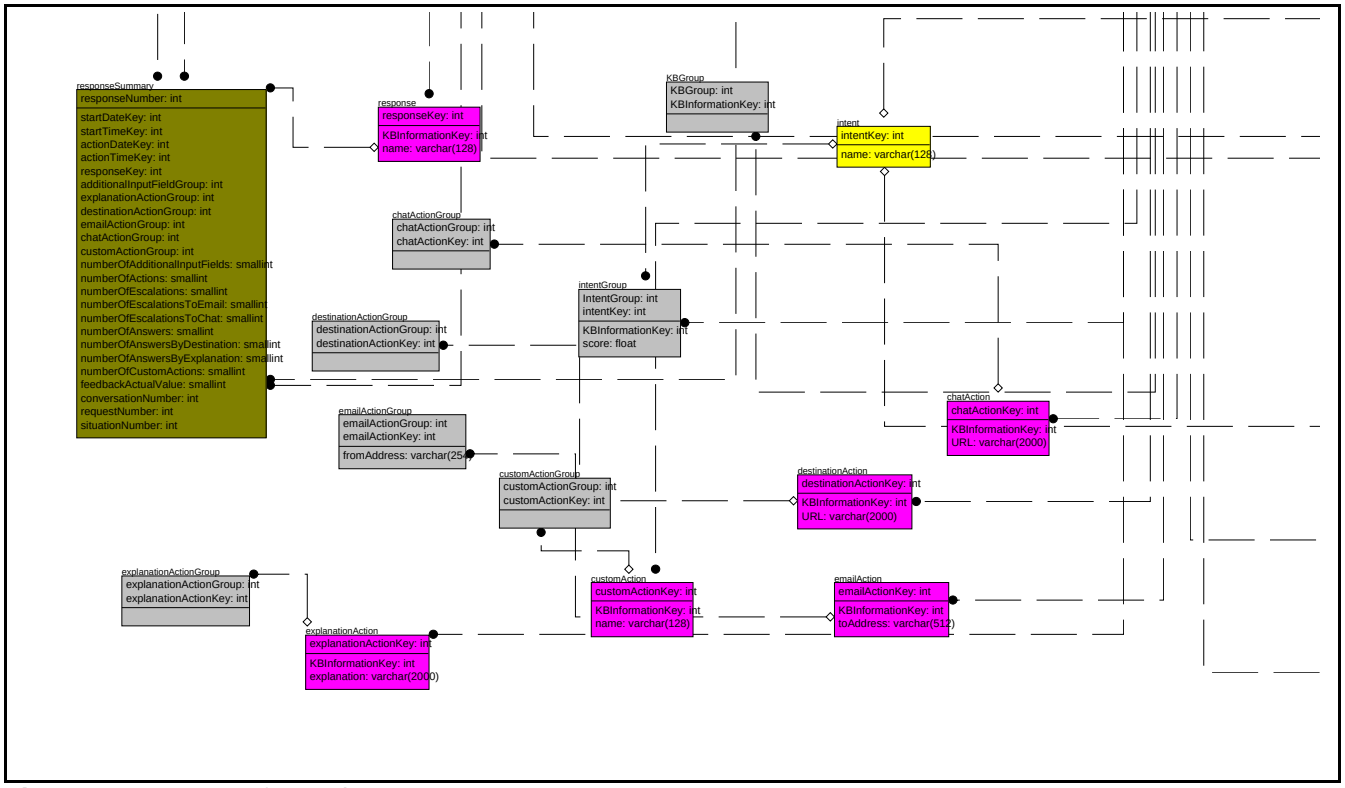

Figure A-3. Lower Left Quadrant

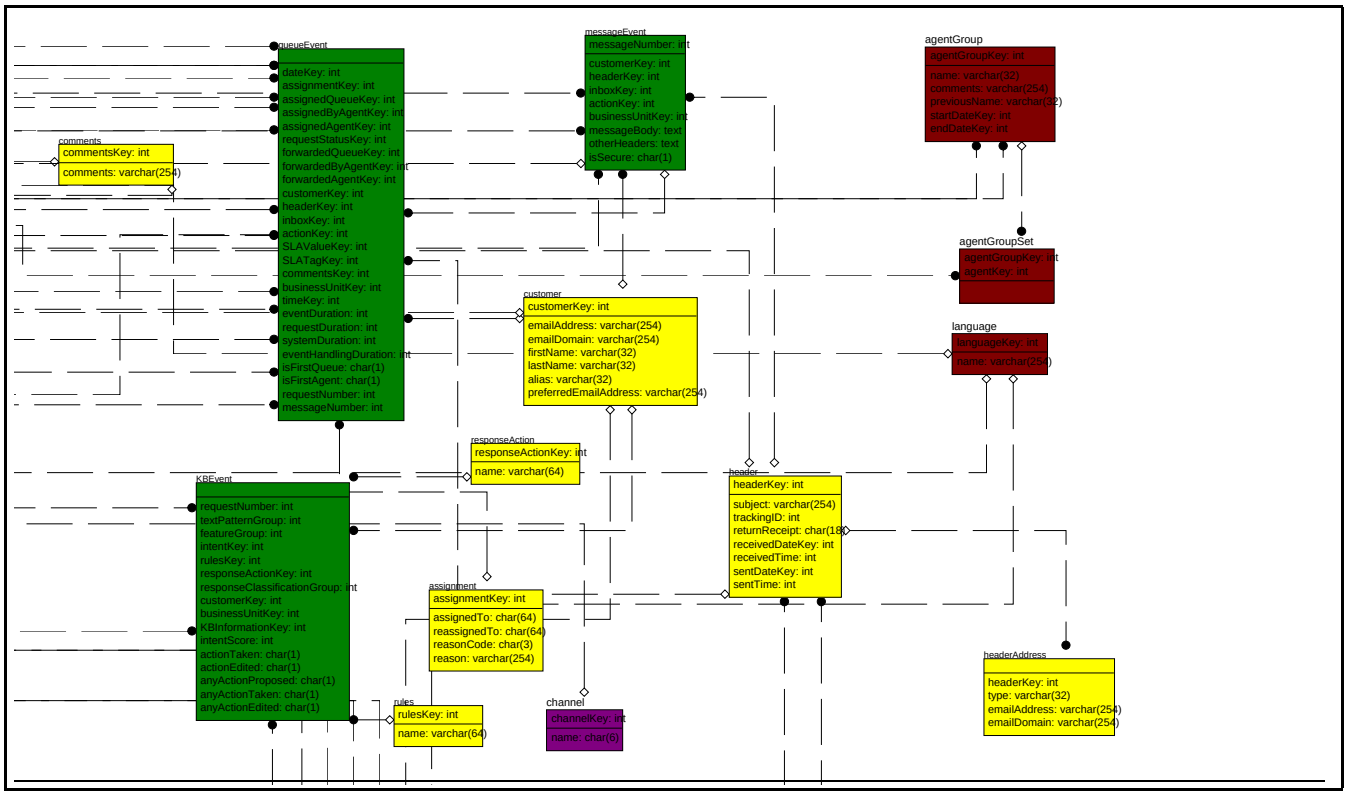

Figure A-4. Upper Right Quadrant

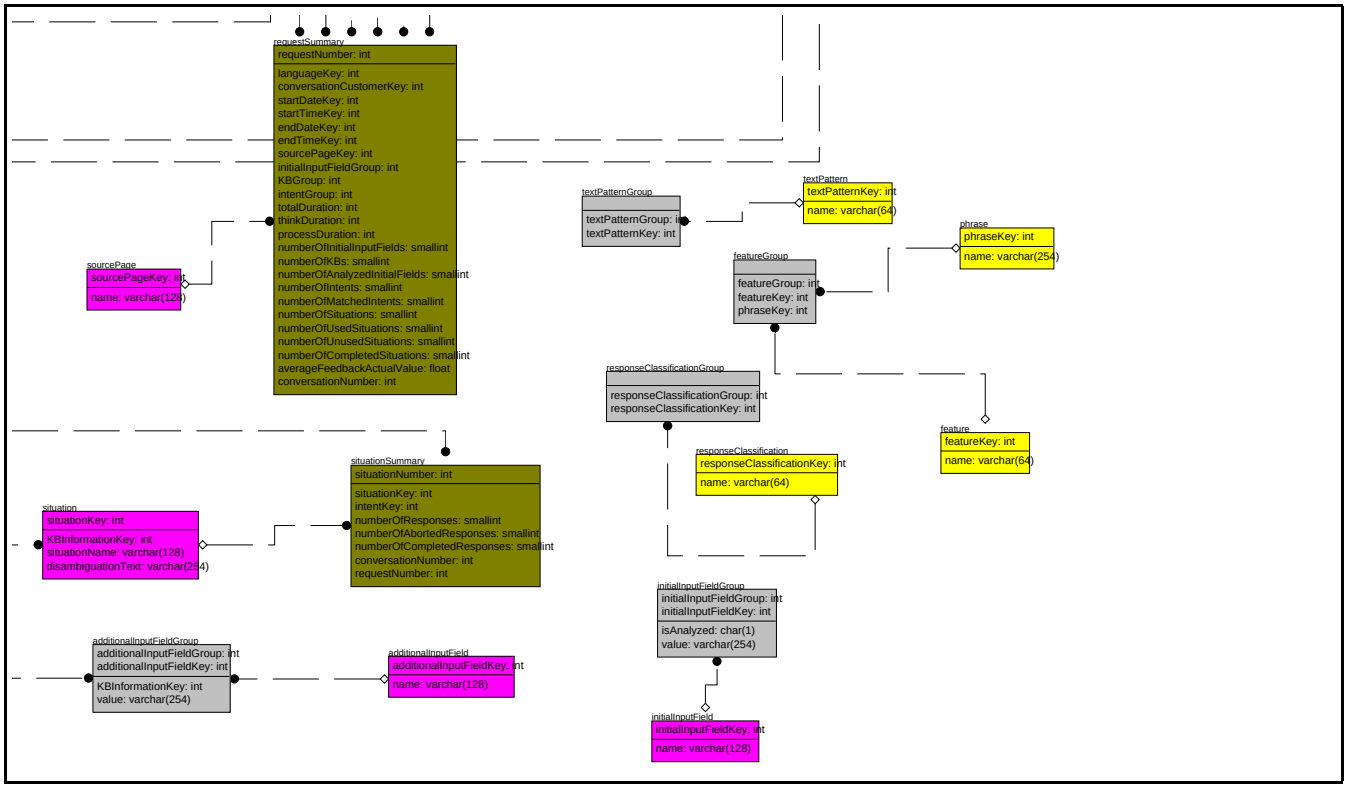

Figure A-5. Lower Right Quadrant

Index