MTX- rev12

DRAFT VERSION

DO NOT DISTRIBUTE WITHOUT PERMISSION

Contents

Chapter 1

Rating Overview

MATRIXX Charging Application can perform both online and offline rating in support of next-generation networks and legacy systems. It can also handle hybrid sets of convergent, prepaid, and postpaid services, enabling service providers to have a diverse subscriber base.

MATRIXX Charging Application uses *subscriber*, *device*, *pricing*, and *wallet* information to associate *events* and *sessions* with the correct subscriber and to determine which rates to use and balances to update. In order for rating to occur, the subscriber and the subscriber's device must have an active status. If either is suspended or inactive, a rating denial message is returned to the network.

You can set up simple rates for subscriber usage, for example flat calling rates, or more complex pricing based on any field in a network message or any field in the subscriber, group, and device *MDC (MATRIXX data container)*. For example, a data rate can be based on the content provider, the quality of service, or the subscriber's loyalty status. You can change the information that is captured in the network message or an MDC and introduce new attributes as frequently as needed, without having to update the core system or define custom code. This allows you to set up subscriber-specific pricing and create innovative service offerings. For example, you can give subscribers real-time incentives during service usage and charge them additional fees when they use more than a monthly grant. Or, you can have a gaming service that charges subscribers based on the number of times they play, the number of points they score, or the length of a gaming session. MATRIXX Charging Application determines which rates to use during rating based on the parameter values that are currently valid.

During rating, the MATRIXX in-memory databases are updated and an *EDR (Event Detail Record)* is logged. Each EDR is added to the *MEF (MATRIXX Event File)* and published to enterprise systems. Activities that do not result in updates to the database, for example, subscriber queries, do not generate EDRs.

Online Charging Scenarios

MATRIXX Charging Application handles the following online charging scenarios using the Diameter Credit-Control application in accordance with 3GPP standards.

- Event-based charging:
    - Event charging with unit reservation (ECUR) — Authorizes a request for usage that is performed in a single transaction. An INITIAL_REQUEST message reserves the units beforehand and a TERMINATE_REQUEST message marks the end of the usage, which is charged for afterward. The reservation of units occurs before service delivery and the balance debit operation is performed after the conclusion of service delivery.
    - Immediate event charging (IEC) — Authorizes a request for usage that is performed in a single transaction, such as SMS and MMS events. This type of charging is used when it is certain that the requested service event will be successful. An initial event request message requests a direct debit to have the charges (not reservations) applied in full, before the service is delivered. The usage is not authorized unless the entire charge can be applied. The request indicates which service has been activated or how many units have been consumed. If the event fails, for example, due to a network outage, all or part of the amount charged for the usage event can be refunded. The credit control process for events is controlled by the corresponding CC-Request-Type EVENT_REQUEST that is sent with Credit-Control-Request (CCR) for a given event.
- Session charging with unit reservation (SCUR) — Authorizes a request for continuous usage, such as voice or data, with an INITIAL_REQUEST message to reserve units beforehand, a series of UPDATE_REQUEST messages during usage to reserve more units if necessary, and a TERMINATE_REQUEST message containing the last Used-Units. The termination message is used to clear any related reservations and charge for total usage.

Both ECUR and SCUR events are session-based and require multiple transactions between the network application and the MATRIXX Engine. ECUR begins with a request to reserve units before delivering services, followed by an additional request to report the actual used units to MATRIXX Charging Application upon service termination. With SCUR, it is also possible to include one or more intermediate request messages to the network application to report currently-used units, and to reserve more units if required. In both cases, the session state is maintained by MATRIXX Charging Application.

For ECUR and SCUR, units are reserved before service delivery and committed upon service completion. Units are reserved with INITIAL_REQUEST and committed with a TERMINATION_REQUEST. For SCUR, units can also be updated with UPDATE_REQUEST. For all three types of online charging, MATRIXX Charging Application complies with the Diameter Credit Control Application (DCCA) defined in RFC 4006.

For more information about the Diameter Credit-Control application, see *Diameter Integration*.

Offline Charging Scenarios

MATRIXX Charging Application can charge offline by reading *CDR (charging data record)* and saving the results to EDRs so the information can be imported into a third-party system for billing or analytical processing.

For more information about CDR processing, see the discussion about processing call detail records.

Rating and Unit Determination

Rating and unit determination are sub-functions for online charging that affect online charging principles. Both can be centralized, where they are performed by MATRIXX Charging Application, or decentralized, where they are performed by the Diameter client.

Unit determination is the calculation of the number of units that are determined before the start of service delivery. They include service units, data volume, and length of time. With centralized unit determination, MATRIXX Charging Application determines the number of units that a certain subscriber can consume based on a service identifier received from the network. With the decentralized unit determination, the Diameter client determines how many units are required to start service delivery and requests these units from MATRIXX Charging Application. After checking the subscriber's balance, MATRIXX Charging Application returns the number of granted units to the Diameter client, which is then responsible for delivering the service and limiting service delivery to the corresponding number of granted units.

Rating is the calculation of a price for the units. With centralized rating, MATRIXX Charging Application determines the price of the units. With decentralized Rating, the Diameter client determines the price of the units.

This chapter includes the following topics:

Event Types

The MATRIXX Engine supports multiple built-in event types that are defined in a tree hierarchy based on pricing configuration and stored in the pricing database. You can define custom subordinate event types (sub events) for an event type.

For example, a service provider offering data services can define new event types based on the VoIP phone app a subscriber uses. Each subordinate event type definition contains information about the activity being performed. It can also include data from a parent event type and the *EDR (Event Detail Record)* generated can contain the parent's data when it is a descendant of the parent EDR. The data recorded for each subordinate event type can vary, depending on its parent event type. For example, a subordinate **data** event type has different parental information than a subordinate **voice** event type. When you compile a pricing plan and load it into the pricing database, the custom event type name and ID are saved in the event type object. These values are used to look up the object when a usage message is received.

Each service type that is a leaf node in the service type hierarchy must be associated with exactly one event type. The MATRIXX Engine supports several service types out-of-the-box, including voice, data, and messaging. IT personnel can define additional service types and event types as necessary. If a service type has accessory services, each accessory service is handled as a subordinate event type. For example, a voice service might be bundled with two accessory services: call forwarding and conference calling. Each accessory service is translated into a separate event type that is subordinate to the voice event.

The `EventTypeArray` in the `MtxPrimaryEvent` MDC defines the event type that was initiated, representing it numerically in the MDC as an array of strings such as 4, 1.2, or 1.3.5. The top-level event types are:

- ROOT = 0
- USAGE = 1
- RECURRING = 2
- FIRST_USAGE = 3
- PURCHASE = 4
- CANCEL = 5
- BALANCE_ADJUST = 6
- BALANCE_TOPUP = 7
- SUSPEND = 9
- RESUME = 10
- SESSION_END = 11
- WRITE_OFF = 12
- NOTIFICATION_DONE = 13
- OBJECT_CREATE = 14
- STATUS_CHANGE = 15
- USAGE_REFUND = 16
- OBJECT_DELETE = 18
- BALANCE_TRANSFER = 19
- BALANCE_ROLLOVER = 20
- ROLLOVER_BALANCE_ADJUST = 21
- SESSION_CONTEXT_END = 22
- FORFEITURE = 23
- PAYMENT_AUTHORIZATION = 24
- PAYMENT_SETTLEMENT = 25
- AUTO_RENEW = 26
- PAYMENT = 27
- POLICY_SESSION_START = 28
- POLICY_CHANGE = 29
- RECHARGE = 30
- TRANSFER_TO_BILLED_AR = 31

- BILLING_CYCLE_CHANGE = 32
- PERIOD_TERMINATION = 33
- SUBSCRIBER_ADD_DEVICE = 34
- SUBSCRIBER_REMOVE_DEVICE = 35
- GROUP_ADD_MEMBERSHIP = 36
- GROUP_REMOVE_MEMBERSHIP = 37
- REFUND = 38
- PAYMENT_REFUND = 39
- PERIOD_WRITE_OFF = 40
- PURCHASED_ITEM_CYCLE_CHANGE = 41
- RECHARGE_REQUEST = 42
- PURCHASED_ITEM_ACTIVATION = 43
- BALANCE_THRESHOLD = 44
- PURCHASED_ITEM_TRANSITION_TO_INACTIVE = 45
- METER_PERIOD_CLOSE = 46
- CONTRACT_LATE_CHARGE = 47
- FINANCE_CONTRACT_PRINCIPAL_PAYMENT = 48
- MISSED_CONTRACT_CHARGE = 49
- CONTRACT_FINANCE = 50
- CONTRACT_DEBT_PAYMENT = 51
- PURCHASED_ITEM_MODIFY = 52
- COMPONENT_METER_PERIOD_CLOSE = 53
- SUBSCRIPTION_MODIFY = 54
- GROUP_MODIFY = 55
- USER_MODIFY = 56
- DEVICE_MODIFY = 57
- SUBSCRIPTION_CREATE = 58
- GROUP_CREATE = 59
- DEVICE_CREATE = 60
- USER_CREATE = 61
- SUBSCRIPTION_DELETE = 62
- GROUP_DELETE = 63
- DEVICE_DELETE = 64
- USER_DELETE = 65
- COMPONENT_METER_SUBPERIOD_CLOSE = 66
- CYCLE_ARREARS_RECURRING = 67

- DEBT_PAYMENT = 71
- PURCHASED_ITEM_STATUS_CHANGE = 72
- FEE_CHARGE = 73
- EXTERNAL_PAYMENT_REQUEST = 74
- EXTERNAL_PAYMENT = 75
- PERIOD_END_TIME_CHANGE = 76
- SESSION_MONITOR = 77
- EXTERNAL_PAYMENT_FAILURE = 78
- PURCHASE_FAILURE = 79
- PURCHASED_ITEM_ACTIVATION_FAILURE = 80
- RECURRING_FAILURE = 81
- USAGE_FAILURE = 82
- BALANCE_TRANSFER_FAILURE = 83

Event Generation for Failed Transactions

The Charging Server generates events and notifications at the end of successful transactions and failed operations.

Events are generated for five categories of failure:

- Purchase
- Purchased Item Activation
- Recurring
- Usage Authorization
- Balance Transfer

Table 1-1 *(on page 8)* lists the failure and how it is determined. Each event record includes the failure reason and information specific to each failure.

Table 1-1 Failure Event Categories

Operation	Credit Limit Reached	Balance Floor Reached	Denial Per Rate Table	No Permission Per Life Cycle Status
Purchase	PURCHASE_FAILURE	PURCHASE_FAILURE		PURCHASE_FAILURE
Purchased Item Activation	PURCHASED_ITEM_ACTIVATION_FAILURE	PURCHASED_ITEM_ACTIVATION_FAILURE		
Recurring	RECURRING_FAILURE	RECURRING_FAILURE		
Usage Authorization	USAGE_FAILURE		USAGE_FAILURE	USAGE_FAILURE
Balance Transfer	BALANCE_TRANSFER_FAILURE	BALANCE_TRANSFER_FAILURE		

For more information about each event, see the MATRIXX event detail records (EDR) in *MATRIXX Engine Integration*.

Failure Operation and Event Types

Table 1-2 *(on page 9)* describes the failure operation and event types.

Table 1-2 Failure Event Mapping

Operation Type	Event Type	Create Event (Default Value)
balance_transfer_failure_group	BALANCE_TRANSFER_FAILURE	false
balance_transfer_failure_subscriber	BALANCE_TRANSFER_FAILURE	false
purchase_failure_device	PURCHASE_FAILURE	false
purchase_failure_group	PURCHASE_FAILURE	false
purchase_failure_subscriber	PURCHASE_FAILURE	false
purchased_item_activation_failure_-device	PURCHASED_ITEM_ACTIVATION_-FAILURE	false
purchased_item_activation_failure_-group	PURCHASED_ITEM_ACTIVATION_-FAILURE	false
purchased_item_activation_failure_-subscriber	PURCHASED_ITEM_ACTIVATION_-FAILURE	false
recurring_failure_group	RECURRING_FAILURE	false
recurring_failure_subscriber	RECURRING_FAILURE	false

For more information about event types, see the discussion about event types in *Pricing and Rating*. For more information about operation and event types, see the discussions about operation types and event mapping properties in *My MATRIXX Help*.

Purchase Operation

When a purchase operation fails, the operation is aborted and the following codes are recorded in the event:

- CREDIT_LIMIT_REACHED – If failure is due to insufficient funds.
- BALANCE_FLOOR_REACHED – If failure is due to reaching the balance floor threshold.
- PERMISSION_DENIED – If failure is due to an invalid life cycle state.

See *Subscriber Management (SubMan) API* for more information about Subman API responses.

If the purchase operation involves multiple offers or bundles, there is a single failure event generated for the purchase operation.

If the purchase operation is wrapped inside a multi-request that fails:

- If the offer owner is created with the purchase operation in the multi-request, there is no failure event because the entire multi-request fails.
- If the offer owner was previously created and the multi-request contains the purchase operation that failed, then a failure event is generated.

Usage Authorization Operation

During usage authorization for a rating group, if an error is to be returned in the message response, a UsageFailure event generates to record the same failure result returned in the message response. The authorization could fail due to insufficient funds or a denial error from the pricing components.

Chapter 2

Pricing Overview

The MATRIXX pricing model is highly flexible and enables you to create innovative service offerings and tailor pricing to subscriber needs. It is compatible with the Telemanagement Forum Shared Information/Data (SID) model.

You create a *pricing plan* to define how to charge *subscribers* for your services, which events to rate, and which policies to apply. Pricing administrators define pricing plans and configure charging and rating behavior by using My MATRIXX, an intuitive, easy-to-use, web-based application. You can easily set up simple and complex pricing and policy decisions due to its simplistic data-driven design. For example, simple pricing can be a recurring charge associated with a monthly device rental. Complex pricing can include a one-time purchase charge, a monthly grant of assets, a monthly usage charge, an overage charge, and a discount—all of which can be combined to provide an overall price for *product offer* or *bundle*.

You deploy pricing by using a Pricing Compiler application which transforms the data into multi-dimensional arrays of algebraic formulas, which you then load into the pricing database. The engine is not aware of the pricing concepts behind the equations—it only needs to execute the math to calculate the correct answer. Because MATRIXX Charging Application uses equations rather than pricing data during rating, calculations are performed consistently across any charge configuration, from the simplest SMS event to the most complex session, in a precise and efficient manner.

There is no tradeoff between pricing complexity and performance, so more data can be processed without reducing performance or decreasing efficiency of a MATRIXX blade. In addition, as new pricing concepts are developed, they are transformed into equations the engine already understands, so future pricing models can be deployed quickly without the cost and risk of customizing code. All items that make up a pricing plan are reusable and easy to assemble. Custom coding is not required to define pricing.

For information about the My MATRIXX interface and configuring pricing, see *My MATRIXX Help*.

This chapter includes the following topics:

Pricing Plans

Pricing plans define the pricing and policies that allow you to control how you provide services to subscribers and how you are paid for those services. Pricing plans are comprised of catalog items, offers, bundles, and supporting objects required to implement pricing.

Catalog items are the pricing items that you offer to subscribers. Catalog items are extensions of offers or bundles that have been customized for a specific group of subscribers. Offers contain pricing and policy components, and a list of the *balances* and *meters* to impact during rating. The pricing and policy components are comprised of items that enable you to choose different rates, policies, and balances in real-time, based on variable conditions that are valid during rating.

- A product offer includes:
    - A product.
    - A pricing item such as a price or policy component to be valid and compiled. See the discussion about product offers in related links for the complete list.
    - A collection of configurable pricing parameters with default values.

- A price component is a charge, discount, or grant. Each price component implements the business logic that determines how products and services are charged for when network messages are received and recurring pricing occurs.
- A policy component implements the business logic that decides which policy profile to return to a PCRF (Sy policy) or PCEF (Gx policy) for a given set of usage circumstances. Policy components enable you to track a subscriber's service usage and change that subscribers quality of service (QoS) based on that usage.
- A balance is a pricing object that represents a currency or asset quantity that is updated by subscriber usage and pricing, and can be used to influence policy decisions. Balances can have credit limits and thresholds set on them and can have associated filters that determine when they can be used.
- A meter is a pricing object that tracks subscriber usage or charges against a balance. Meters can have credit limits and thresholds set on them and can be used to apply different rates or policies.
- A decision table is a multi-dimensional matrix that contains one or more normalizers that together define a condition set. Decision tables can be used by rate tables, policy tables, GL transaction type tables, GL account tables, balance filters, and offer priority generators to assign a decision result to each condition set.

Product offers can be grouped into product bundles before being offered as a catalog item. This allows you to sell multiple offers as one unit. Bundles can include their own price components that override the price components defined in the product offer. This allows you to provide bundle-specific charges, grants, and discounts. Product offers can be reused in multiple bundles. Note that a finance contract offer is purchased as a separate catalog item and cannot be part of a bundle.

Offers and bundles contain all rating and policy components and most pricing details. The remaining pricing details and metadata fields are added to the offers and bundles as they are configured into catalog items. A catalog item is a customized product offer or bundle that your Catalog Users create. Catalog items can be a tangible or intangible item that you market, sell or lease to customers for profit. It represents the provisioning of a service, resources, or both. Catalog Users create catalog items by:

- Customizing offers or bundles with any pricing changes allowed for that offer.
- Adding other commercial metadata fields in the form of name/data type/value tuples. A metadata field could, for example, reference a logo file or a set of terms and conditions for a specific market segment.

Figure 2-1 *(on page 11)* illustrates several product offers in a pricing plan.

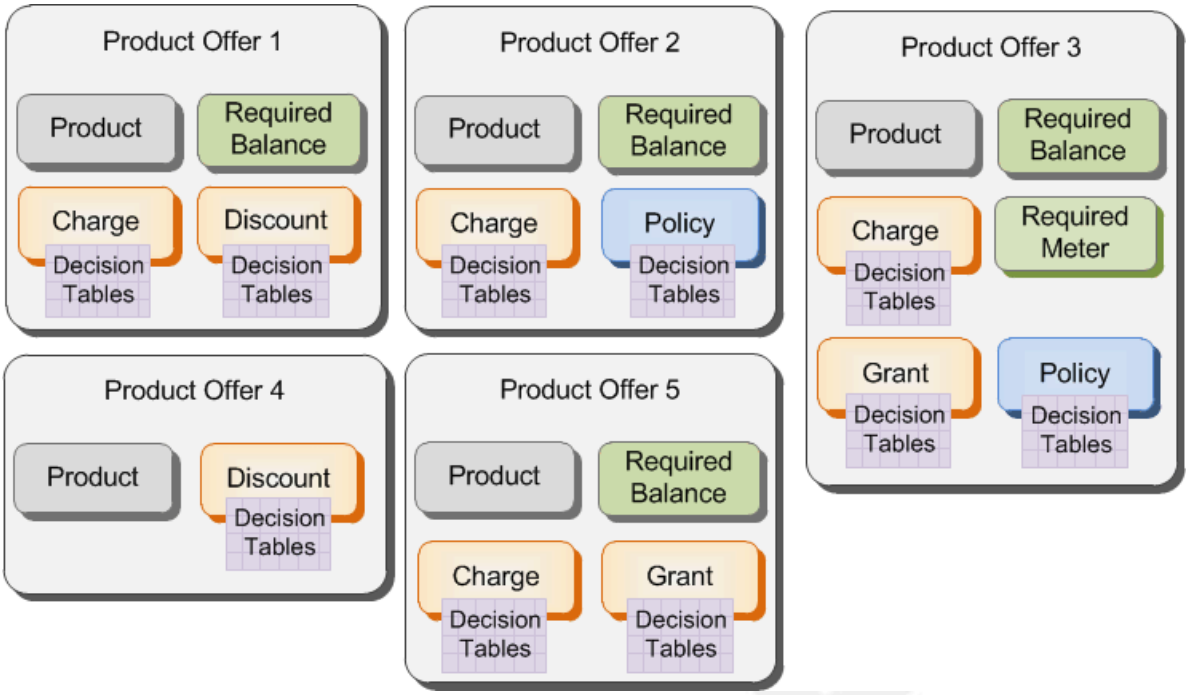

Figure 2-1 Product Offers in a Pricing Plan

The five product offers in Figure 2-1 *(on page 11)* can be included in a bundle, allowing you to sell all five offers as a single unit. Each product offer in a bundle can include one or more products, each representing a different service. You then create a catalog item from a bundle, adding any pricing changes or commercial items that it requires.

Each offer or bundle can be used to create multiple catalog items as needed. This enables you to create elaborate charging and discounting models and take a "create once, update everywhere" approach to pricing. This results in fewer items in a pricing plan, which makes it easier to manage additions and changes. For more information, see the discussions about catalog item revisions, product offers, and product offer versions and revisions.

Balances and Meters

When you define *rates*, you specify which *balances* and *meters* to impact when rating occurs. The impacts are a result of the *charges*, *discounts*, and *grants* applied, and represent the financial liabilities to *subscribers* and *groups*.

Balance Transfers

A *balance* transfer is a *subscription* management transaction that moves an amount from one balance instance to another balance instance.

The following rules apply to balance transfers:

- Transfer balances between any two prepaid *G/L balance* instances of the same balance unit, for example, minutes or megabytes.
- The source balance and the target balance can be in the same *wallet* or in different wallets.

  For example, a *subscription* belonging to a family sharing *group* can transfer 100 voice minutes from his or her minutes balance to the group minutes balance, or to another family member. Note that with this example the balances are in the same group hierarchy, but this is not a requirement.
- The source and target balances can be *simple balances* or *periodic balances*.

For periodic balances, only the current balance interval is impacted. Transfers to or from on demand balances are not supported.

- The source balance can be expired. The target balance cannot be expired.
- You cannot transfer balances between an *actual currency balance* and a *pseudo-currency balance*.

  Transfers must be from an actual currency balance to an actual currency balance or from a pseudo-currency balance to a pseudo-currency balance. For more information about actual currency balances and pseudo-currency balances, see the discussion about actual currency and pseudo-currency balances.
- When MATRIXX Engine is configured to include GL information in events, the source and target balances must be either both liability assets or both non-liability assets.

  For example, you cannot transfer a liability asset balance to a non-liability asset balance.
- The amount being transferred must be specified either as a positive absolute value or as a percentage of the available amount in the source balance.

  The entire amount in the source balance is available for transfer, regardless of any reservations against that amount. However, if the specified transfer amount exceeds the available amount, the operation will fail. Any transferred amount that had been reserved in the source balance will no longer be reserved in the target balance.

Note: If the transfer amount removes the entire amount from the source balance, and that balance is configured to automatically expire, the balance is not expired upon completion of the transfer request. The next time Task Manager runs, as configured during MATRIXX Engine configuration, Task Manager processes the expiration.

Because the target balance amount decreases (for prepaid balances, the balance amount is negative), after a balance transfer, customers can decide whether to also adjust the *credit floor* of the target balance during the transfer operation. The credit floor is the starting value from which to calculate percentage threshold notification amounts, and is used to manage *notifications* for prepaid balances. The credit floor adjustment options are:

- No adjustment (1)
- Adjust by the transferred amount (2)
- Adjust by source credit floor amount (3)

The default is No adjustment. The adjustment amount for each option is calculated as described in Table 2-1 *(on page 13)*.

Table 2-1 Calculating Adjustment Amounts

Option	Behavior
1	No credit floor amount is calculated and the credit floor of the target balance is not adjusted.
2	The amount by which to adjust the target balance's credit floor is calculated by using the transferred amount.
3	The amount by which to adjust the target balance's credit floor is calculated based on whether the transfer amount is a percentage or an absolute value. • If the transfer amount is a percentage, the calculation uses the percentage of the source credit floor amount. For example, if the source credit floor is 10 MB and the transfer amount is 10%, the amount by which to adjust the target balance's credit floor is 1 MB (10% of 10 MB = 1 MB).

Table 2-1 Calculating Adjustment Amounts (continued)

Option	Behavior

	• If transfer amount is an absolute amount other than 0, the calculation used is:
	`(transferred amount ÷ source gross amount) × source credit floor amount`
	For example, say the source credit floor is 1000 MB, with 500 MB in the source balance, and 200 MB are transferred. The calculation is:
	`200 MB ÷ 500 MB = .4 MB`
	`.4 MB × 1000 = 400`
	The amount by which to adjust the target credit floor is 400 MB.
	• If the transfer amount is an absolute amount of 0, the credit floor of the target balance is adjusted by an amount of 0.

After the credit floor adjustment amount is determined, the way it is applied to the target balance credit floor is based on whether the target balance is a simple balance or a periodic balance and the **Grant Credit Floor Adjust** value in the balance template:

- Simple balances — If the **Grant Credit Floor Adjust** value that is set in the balance template is **Grant Amount plus Balance Amount**, the new credit floor is equal to the current balance amount plus the calculated credit floor adjustment amount. If the **Grant Credit Floor Adjust** value is **Actual Grant Amount**, the new credit floor is equal to the calculated credit floor adjustment amount.

  The simple balance Grant Credit Floor Adjust values are:

  - Grant Amount plus Balance Amount
  - Grant Amount
- Periodic balances — The new credit floor is equal to the current credit floor plus the calculated credit floor adjustment amount.

The balance transfer operation generates a MtxBalanceTransferEvent for the event initiator. This event type contains optional Reason and Info fields that are populated from the balance transfer API and a field that indicates if the transfer was to or from the event initiator. The Reason field is also copied to the MtxEventChargeNormalizationInfo MDC to be used for normalizations on General Ledger (GL) account data. In addition, when an amount is transferred from a balance:

- If the balance is a currency, revenue is recognized according to the GL information in the MtxBalanceTransferEvent EDRs (both primary event and secondary event).
- If the balance is a non-liability asset, there is no action because no GL information is generated in MtxBalanceTransferEvent EDRs.
- If the transferred balance is a liability asset, the asset and its deferred revenue amount is transferred from the source wallet to the target wallet. Revenue is recognized later when the transferred asset is consumed. The GL account names and transaction type specified in the GL information in the transfer event override those specified in the GL information from the original purchase event.

Note: Balance transfers do not generate threshold notifications and do not trigger auto-expiration.

For more information about auto-expiring balances, see *Pricing and Rating*.

For information about using the SubMan APIs to transfer a balance amount, see *Subscriber Management (SubMan) API*.

For information about normalizing on the transfer reason for GL accounts, see *Pricing and Rating*.

Failed Balance Transfers

If the balance transfer operation fails due to insufficient funds, a MtxBalanceTransferFailureEvent generates to record the failure under the following conditions:

- You have configured the `balance_transfer_failure` operation types to generate an event.
- The specified transfer amount is greater than the available balance amount.

For more information about event failure, see the discussion about event generation for failed transactions in *Pricing and Rating*.

Cycle Processing

The Charging Server supports balance cycles, billing cycles, and purchased item cycles.

For balance and billing cycles, each subscriber or group wallet defines one or more cycles. A wallet defines one billing cycle, and each periodic balance defines its own cycle determined by the period boundaries of the balance. Pricing components, such as charges, discounts, and grants, can be defined that are triggered each time a cycle enters a new period. The execution of these pricing components at each new period is called recurring processing.

For information specific to billing cycles, balance cycles, and purchased item cycles, see the following sections.

Recurring Processing

Recurring processing can be triggered when MATRIXX Task Manager sends a message to the Charging Server because a trigger time in a *subscriber* or *group wallet* is reached.

When processing the message, the charging server executes recurring processing on any cycle for which recurring processing has not been done. In addition, recurring processing can be triggered when a network or subscription management message is received. In this case, the Charging Server invokes pending recurring processing for the subscriber or group that is the target of the message.

The general order for recurring processing for a subscriber or group is:

1. Purchased item cycles.

   Cycle definitions are included in the subscription data, which includes information about the specific period boundary offset, the cycle state (such as recurring failure), and how the cycle is aligned (for example, with the balance cycle, the billing cycle, and so forth).

2. Balance cycle.

   Note: You cannot control the order of processing among the balance cycles. All the cycles of a subscriber or group are done before moving up to do recurring processing for a parent group.

3. Billing cycle.

All charges associated with a cycle are applied first, before any grants or rollovers. In addition, cycle processing occurs before any subscriber event processing. If a catalog item is purchased or canceled in the middle of a balance or billing cycle, the pricing is applied with any proration requirements. Recurring processing does not use product offers or bundles that have been canceled and for which the cancel end time has been reached. Pricing revisions are from

the beginning of the cycle period, but the Charging Server does not use the product offer or bundle even if it was canceled after the start of the period.

For more information about external payment requests, see the discussion about balances with external payment requests.

When recurring charges are applied to balances associated with external payment requests, during recurring processing for purchased item cycles the external payment request records in the wallet are updated according to the following:

- If no external payment request is found, an external payment request is generated. The payment due date is the greater of the recurring time and the `PaymentDueDate` in MtxPurchasedOffer.
- If the external payment is due, recurring processing is successful and there is no additional action.
- If the external payment is pending, the collected payment is processed, the balance is credited, the `PaymentStatus` is updated to paid, and an MtxExternalPaymentEvent is generated.

For more information about external payments, see the discussion about balances with external payments.

First-use charges and recurring processing are triggered by the G/L balance only. A subscriber or group virtual balance does not trigger first-use charges or recurring processing.

The `RecurringFailureStatus` field in the MtxWalletObject (bill cycles) and MtxPurchasedOffer (purchased item cycles) MDCs indicates if recurring processing succeeded in a cycle period on the first attempt. You can use this field in normalizations in bill cycle recurring components and purchased item cycle recurring components. You might want to use this normalization in a situation where you want to grant an additional amount or charge less if processing succeeds on the first attempt.

The `RecurringFailureStatus` field values are:

- 0 — Recurring processing has not failed for the cycle on the first attempt.
- *Any other value* — Recurring processing failed at least once for the cycle period (but not on the first attempt). Normalizers should not assume which non-zero value is used, or make distinctions between different non-zero values.

Note: You cannot use this field for normalizations in balance cycle recurring components. In addition, normalizations with this field are not applicable to cycle arrears recurring processing for a bill cycle because credit limits are ignored and processing does not fail. For more information about normalizers, see the discussion about normalizers.

For more information about recurring processing failure, see the discussion about recurring processing failure.

Recurring charges and grants applied during purchase or resumption of an offer are recorded in separate recurring events (MtxRecurringEvent).

Table 2-2 *(on page 16)* describes the MtxRecurringEvent EDR fields which provide information about the triggering cycle.

Table 2-2 MtxRecurringEvent Fields

Field	Type	Description
AssociatedEventId	String	Set only if event is triggered by Purchase or Resume event. Value is the ID of the associated Purchase or Resume event.
CycleType	UINT32	Type of cycle. Values are:

Table 2-2 MtxRecurringEvent Fields (continued)

Field	Type	Description
		• 1 = Balance cycle • 2 = Billing cycle • 3 = Purchased Item cycle
CycleStartTime	Datetime	Start time of recurring cycle.
CycleEndTime	Datetime	End time of recurring cycle.
CycleBalanceTemplateId	UINT64	Pricing ID of balance template. Set only if cycle type is balance cycle.
CycleBalanceResourceId	UINT32	Resource ID of balance instance. Set only if cycle type is balance cycle.
CycleIntervalId	UINT32	If the cycle type is balance cycle, this is the periodic balance interval Id. If cycle type is billing cycle, this is the billing cycle interval Id. For purchased item cycles, this is the interval ID for the cycle period.
CyclePurchasedItemResourceId	UINT32	Set only for purchased item cycles. The value is the resource ID of the purchased offer or bundle with the cycle.
CycleCatalogItemId	UINT64	Set only for purchased item cycles. The value is the ID of the catalog item associated with the purchased offer or bundle with the cycle.
CycleCatalogItemExternalId	String	Set only for purchased item cycles. External Id of the catalog item associated with the purchased offer or bundle with the cycle.
PaymentCount	unsigned int32	The installment payment number that the recurring cycle being processed corresponds to. Applicable only for contract offers associated with a payment schedule.
TotalPaymentCount	unsigned int32	The total number of installment payments in the payment schedule Applicable only for contract offers associated with a payment schedule.
PaymentPeriodStartTime	datetime	The start time of the period for which the payment applies. Applicable only for contract offers associated with a payment schedule.
PaymentPeriodEndTime	datetime	The end time of the period for which the payment applies. Applicable only for contract offers associated with a payment schedule.

For more information about MtxRecurringEvent, see *MATRIXX Engine Integration*.

The MATRIXX Charging Application does not attempt recurring processing for a cycle period after that period ends, even if a network message for usage in the previous period is received. Recurring processing can be invoked for the current period of a cycle, even if recurring processing did not succeed for previous periods. For information about recurring processing failures, see the discussion about recurring processing failures.

For information about configuring recurring processing, see the discussion about Task Manager configuration in *Installation and Configuration*.

Recurring Processing Failure

Recurring processing can fail when a subscriber or group has insufficient funds or credit to pay recurring charges.

If charges and their applicable discounts cannot be fully applied during cycle processing, the cycle is not processed. As a result, no price components get applied and the state of the balance (for balance cycles) or subscriber or group (for billing cycles and purchased item cycles) remains unchanged.

Note: You can specify in the offer or bundle cycle data properties, or during purchase using the SubMan APIs, that processing can continue if purchase components are applied successfully, but there is insufficient credit to pay the recurring charge at purchase time. For more information, see the discussion about allowing recurring processing failure at purchase.

In cases where recurring processing fails, recurring processing is retried periodically in the current cycle and again in the next cycle. Recurring processing is also retried in the following situations in the current cycle:

- Every time there is usage or there are queries on the subscriber.
- A SubMan operation is executed that could cause recurring processing to succeed.

  For example:

  - A top-up is made.
  - A product offer or bundle is purchased.
  - A balance is adjusted.
  - A change is made to the attributes of a subscriber or group.
- Until the period ends.

Recurring processing, retries for each cycle are performed in-order of when the recurring processing was due. If there are two or more recurring cycles with the same recurring due date, they are processed in priority order. Priority is not evaluated across all recurring cycles. A numerically lower priority value has a higher priority. Consider the following:

- Recurring processing fails today with two independent offers of priority 10 and 9 with the same recurring cycle time.
- Tomorrow another offer is set for recurring processing. It has priority 20.

In this scenario, the order of processing is:

- Due Yesterday: offer priority 9 then offer priority 10.
- Due Today: offer priority 20.

Note: Recurring processing for a cycle period ends when the period ends; network messages for usage from a previous period do not trigger recurring processing in the current period. Recurring processing failure on a purchased item cycle that specifies continue after failure is `false` stops other purchased item cycles, but does not stop recurring processing for balance cycles or billing cycles. If multiple offers with the same purchased item cycle are associated with grace period profiles, when recurring processing fails for the first offer and continue after failure is `false`, recurring processing stops. If the offers are associated with grace period profiles, only the first offer transitions to the grace status. The other offers have not executed recurring processing and there is no status transition for those offers.

For more information about balance, billing, and purchased item cycle processing, see the discussion about cycle processing.

Recurring Processing Failure Notifications

Configure balance templates, billing profile templates, and product offer cycle data in My MATRIXX to reference a notification profile which defines when notifications are sent after recurring processing fails (the failure to apply recurring charges). Notifications include the following information.

- If the failure was for a balance cycle, billing cycle, or purchased item cycle.
- If the failure was for a balance cycle, the balance template ID and the balance resource ID.
- The OID and external ID of the purchased item owner.
- If the failure was for a purchased item cycle, the resource ID of the purchased item, the catalog item ID of the purchased item, and the external ID of the catalog item.
- The start time and the end time of the cycle period for which recurring processing failed.
- An estimate of recurring charges and grants (advice of charge) for the cycle.

Note: This information can be used to determine the amount of a balance recharge or payment. Credit limits of actual currency balances are ignored and balances are not updated in the database. If recurring processing failed due to insufficient funds in a pseudo-currency balance or an asset balance, no estimate is provided. The advice of charge information is included in the MtxNotification balance impact fields.

Each recurring processing failure notification profile can be configured to send a notification on failure and multiple notifications (minutes, hours, days, weeks, months, or years) relative to the time of the initial recurring failure for the period in which the failure occurred. If the balance instance is periodic, a balance expiration notification is triggered when the instance expires and when each individual entry in the instance expires.

If a recurring failure notification profile is defined 'On Event' and the offer fails, a recurring failure notification is sent on the initial recurring failure. If you do not define the profile to send multiple notifications, subsequent recurring failures within the same cycle do not generate a notification.

Note: Recurring processing failure notifications cease when recurring processing is successful. If recurring processing succeeds (for example, due to a product offer purchase) subsequent to a failure and before the time to generate the notification, the notification is not generated. For example, if the recurring failure event is set to be five minutes after the event and recurring processing succeeds before the notification is scheduled to be sent, no failure notification is sent.

For more information about configuring balance templates, billing profile templates, and notification profiles, see *My MATRIXX Help*.

Recurring Failure Events

When a recurring operation fails due to insufficient funds, the system does not raise an exception or abort the operation, instead it logs a message and continues executing.

The MATRIXX Engine can retry the recurring operation if there is any failure and track the number of recurring failures.

For more information about event types, see the discussion about event types in *Pricing and Rating*. For more information about operation and event types, see the discussions about operation types and event mapping properties in *My MATRIXX Help*.

Status Life Cycles

Use the MtxStatusConditionRecurringFailure condition to configure a subscriber or group status life cycle to transition after a billing cycle, balance cycle, or purchased item recurring processing failure. The transition can be delayed by a configured amount of time relative to the time of the initial recurring failure for the period. For balance cycles, you can specify a balance class and optionally a balance template in the condition. The condition is triggered if recurring processing fails for any balance of the specified balance class or for the specific balance template.

For more information about the status life cycles, see the discussion in status life cycles in *Subscriber Management (SubMan) API*. For more information about configuring status life cycles, see the discussion in status life cycles in *My MATRIXX Help*.

Chapter 3

MATRIXX Event Detail Records (EDRs)

The MATRIXX Engine generates EDRs for all activities that can trigger rating, such as usage, catalog item purchases and cancelations, recurring cycle processing, or the first use of a balance. MATRIXX Engine also generates EDRs for non-rated events such as forfeitures. You configure event generation in My MATRIXX, including the events to generate and whether to add a custom container with additional mapped fields.

EDRs can contain information from the network message, such as the subscriber ID, device ID, and event type initiated. They can also contain information about the product offers used during rating, the charges, discounts, and grants that update balances, and updated balances and meters. For aggregated balances, only the impacts to the general ledger (G/L) balance are recorded. For debits associated with one-time usage events, such as SMS, if the usage event fails, the refund information is also included in the EDR.

Each EDR has a unique identifier (the `EventId`) which has a minimum of two parts:

- A three-character encoded event date for internal use. For example, `F2F`.
- An event ID value (string). This is the value assigned by the MATRIXX Engine. For example, `0:1:52:7`.

`F2F0:1:52:7` is an example of an `EventId` with an encoded event date.

In addition to the `EventId`, each EDR also includes:

- An `EventTime`.
- (Usage EDRs) a `Duration`.
- If the EDR is associated with secondary events, a `SecondaryEventList` to identify them. Also a placeholder attribute called `SecondaryEventList` which is initially empty, and is later populated with any related secondary events by the Event Streaming Framework.

Each EDR contains the balance impacts for a single *wallet* (subscription or group) only. EDRs also include the following information when a zero amount is applied to offers and balances:

- An optional, configurable per-engine prefix value for use in multi-engine installations.
- Charges.
- Discounts.
- Grants.
- Adjustments.
- cancelation refunds.
- cancelation forfeitures.

Note: Tax information for charges, discounts, and refunds is also included in EDRs when applicable. For information about taxes, see the discussion about taxes in *Pricing and Rating*.

Multiple EDRs can be generated per network message in the following circumstances:

- If an activity impacts multiple wallets, then multiple EDRs are generated with one primary event associated with the event initiator's wallet, and one or more secondary events. Secondary events reference the primary event `EventId` and do not duplicate the primary event balance impact information. When primary and secondary EDRs are generated, they are always in the same `MtxEventRecordData`.

For other impacted wallets, the MATRIXX Engine generates secondary events that record impacts to balances in those wallet. Secondary events can include some information from the primary event, such as the ID of the event initiator, and certain tax-related fields. Secondary events are included in one batch with the associated primary event. Secondary events reference their primary event using the primary event `EventId`.

- When the action triggers recurring processing or processing associated with first use of a balance. A separate EDR is created for the action and for the recurring processing or first use processing.
- When a Diameter Credit-Control (CC) message includes usage for multiple service contexts, each of which is rated separately. A separate EDR is created for each service context.
- When rating a session and an INTERIM message is received (regardless of whether the INTERIM message has a rated amount). A separate EDR is created for each INTERIM message and the STOP message.

Extend an EDR with additional fields by changing the service type definition or the event type definition in My MATRIXX. Any events added to parent service type definitions are also included in the EDR for a child service type. The fields can be located in a message MDC, and the subscription, group, device, and user MDCs. The fields can also be located in the `MtxMultiServiceData` MDC, which allows administrators to populate EDRs with fields from specific service contexts. The source field must be located in the `MultiServiceList` element (in the message MDC) for a service context.

To add fields from non-usage events, you must create an event mapping to map the event containing the additional fields to an operation type, such as `recurring_subscription`. For more information about configuring event type and service type definitions, and event mapping, see *My MATRIXX Help*.

EDRs are stored in the Event database. When the Event database reaches the maximum allocated size, EDRs are deleted starting with the oldest to make room for new EDRs. The Event database allows recent events to be included in query results before they have been written to an external database. SubMan queries can retrieve in-process aggregations associated with a specific device, subscription, or group. Each event contains balance impacts for one subscription or group. To create a complete event query result, query both the external and the in-memory Event database, and then remove any duplicated events (events found in both databases). A query of the external database by itself does not include events generated shortly before the query.

Note: The Event database only allows insert and delete operations; EDRs in the Event database cannot be updated or modified.

EDRs are written to the transaction log as part of the commit operation for the transaction that rated the event. This ensures that an EDR is always logged if balances in the database were impacted. Activities that do not result in updates to the database, for example, subscription queries, do not generate EDRs. EDRs are written in the transaction log in MDC format. The Transaction Server changes to a new log file based on time and size configuration parameters.

EDR details are contained in *MATRIXX Event Files (MEFs)*. For an overview of MEFs, see the discussion about MATRIXX Event Files.

A MATRIXX Engine can optionally load EDRs into the MATRIXX *Event Repository*. The Event Repository is an optional component separate from the MATRIXX Engine platform that serves as a long-term event storage repository. Each EDR includes a `DeleteCode` which can be used to remove EDRs from the Event Repository using the Event Purge utility. For more information about the MATRIXX Event Repository and the Event Purge utility, see *MATRIXX Engine Integration*.

How and where EDRs are stored (whether they are loaded into the Event database, loaded into the Event Repository, or written into MEFs) has no effect on the EDR contents.

For more information about MtxNotification, see the discussion about notification data relationships in *MATRIXX Engine Integration*.

This chapter includes the following topics:

Customizing Non-Usage Events

Customizing Non-Usage Events

You can map any operation (such as a recurring charge or grant) for an object (subscriber, group, device) to an event (such as a custom MyStatusChangeEvent) and define the fields to be included in the Event Data Record (EDR).

For example, consider a custom MDC, CsrBalanceAdjustEvent that has a field `csr_name`, that extends MtxBalanceAdjustEvent. You can include the `csr_name` field in EDRs by mapping the operation balance_adjust_subscriber to the custom CsrBalanceAdjustEvent EDR. You must follow the procedure described in this section to include the custom field in EDRs.

When you create a service type definition, you can specify the fields that are included in EDRs in **Event Field Mappings** in My MATRIXX for the usage event associated with that service type. The process for including custom fields from non-usage events in an EDR is slightly different.

Perform the following steps to customize non-usage events and add fields to EDRs.

1. During MATRIXX Engine configuration, create a custom MDC that extends a non-usage event.
2. In My MATRIXX, create an event and field mapping.
    a. In Event Types:
        i. Create an event based on the custom MDC.
        ii. Create a field mapping that maps incoming fields to an event type MDC.
    b. In Event Mappings:
        i. Edit an event mapping (such as recurring_subscriber) and map the same event to that operation type.
        ii. To include the custom fields in EDRs, set **Create Event** to "true".

For example, consider the following event type definition:

- ID: 32
- Path: MtxRecurringEvent | CustomRecurringEvent
- Name
- MtxNormalizedEventType
- ContainerName
- TypeId
- Application

The base MDC for `CustomRecurringEvent` is `MtxRecurringEvent`. The event mapping is:

- ID: 2
- Op: recurring_subscriber
- Event Type Id: 32
- Create Event: true

Non-usage EDRs can be generated when an object (device, subscriber, or group) is created or its status changes. The following code shows an example of a status change event in an EDR for a `status_change_group` operation mapped to the base `MtxStatusChangeEvent`.

```
<MtxStatusChangeEvent>
    <StatusBefore>1</StatusBefore>
    <DescriptionBefore>Active</DescriptionBefore>
    <StatusAfter>2</StatusAfter>
    <DescriptionAfter>Inactive</DescriptionAfter>
    <IsSysInit>0</IsSysInit>
    <ObjectId>0:1:5:1084</ObjectId>
    <ExternalId>group externalId</ExternalId>
    <EventTypeArray>
        <value>15</value>
    </EventTypeArray>
    <InitiatorId>0:1:5:1084</InitiatorId>
    <InitiatorExternalId>group externalId</InitiatorExternalId>
    <Flags>1</Flags>
    <WalletId>0:1:5:1085</WalletId>
    <EventTime>2012-02-12T12:20:12.021120Z</EventTime>
    <EventId>0:1:52:847</EventId>
</MtxStatusChangeEvent>
```

Operation Types

Use operation types to customize non-usage events. The operation type in an event mapping must match an event type. For example, the recurring_subscriber operation can only correspond to a recurring event, such as Subscriber-CustomRecurringEvent.

Table 3-1 *(on page 24)* lists the supported operation types.

Do not remove. Adam owns this file. Please consult with him before making any changes. https://j.m0012242008-.com/git/projects/MTX/repos/mtx/browse/data/data_container_subtype/PriceLoaderOperationType.xml#3

Table 3-1 Event Mapping Operation Types

ID	Operation Type	Default Create Event Value
0	undefined	true
1	usage	true
2	recurring_subscriber	true
3	recurring_group	true
4	first_usage_subscriber	true
5	first_usage_group	true
6	purchase_subscriber	true
7	purchase_group	true
8	purchase_device	true
9	cancel_subscriber	true
10	cancel_group	true
11	cancel_device	true
12	balance_adjust_subscriber	true
13	balance_adjust_group	true
14	balance_topup_subscriber	true

Table 3-1 Event Mapping Operation Types (continued)

ID	Operation Type	Default Create Event Value
15	balance_topup_group	true
17	suspend_subscriber	true
18	suspend_group	true
19	suspend_device	true
20	resume_subscriber	true
21	resume_group	true
22	resume_device	true
23	session_end	true
24	write_off_subscriber	true
25	write_off_group	true
26	notification_success	true
27	notification_failure	true
28	object_create_subscriber	false
29	object_create_group	false
30	object_create_device	false
31	status_change_subscriber	true
32	status_change_group	true
33	status_change_device	true
34	usage_refund	true
35	balance_validity_change_subscriber	true
36	balance_validity_change_group	true
37	object_delete_subscriber	true
38	object_delete_group	true
39	object_delete_device	true
40	balance_transfer_subscriber	true
41	balance_transfer_group	true
42	balance_rollover_subscriber	true
43	balance_rollover_group	true
44	rollover_balance_adjust_subscriber	true
45	rollover_balance_adjust_group	true
46	session_context_end	false
47	forfeiture_subscriber	true
48	forfeiture_group	true
49	payment_authorization	true
50	auto_renew_subscriber	true

Table 3-1 Event Mapping Operation Types (continued)

ID	Operation Type	Default Create Event Value
51	auto_renew_group	true
52	payment_settlement	true
54	policy_session_start	true
55	policy_change	true
56	payment_subscriber	true
57	payment_group	true
58	recharge_subscriber	true
59	recharge_group	true
60	transfer_to_billed_ar_subscriber	true
61	transfer_to_billed_ar_group	true
62	billing_cycle_change_subscriber	false
63	billing_cycle_change_group	false
64	period_termination_subscriber	true
65	period_termination_group	true
66	subscriber_add_device	false
67	subscriber_remove_device	false
68	group_add_membership	false
69	group_remove_membership	false
70	refund_subscriber	true
71	refund_group	true
72	payment_refund	true
73	period_write_off_subscriber	true
74	period_write_off_group	true
75	purchased_item_cycle_change_subscriber	false
76	purchased_item_cycle_change_group	false
77	purchased_item_cycle_change_device	false
78	recharge_request	true
79	purchased_item_activation_subscriber	true
80	purchased_item_activation_group	true
81	purchased_item_activation_device	true
82	balance_threshold_subscriber	false
83	balance_threshold_group	false
84	purchased_item_transition_to_inactive_subscriber	false
85	purchased_item_transition_to_inactive_group	false
86	purchased_item_transition_to_inactive_device	false

Table 3-1 Event Mapping Operation Types (continued)

ID	Operation Type	Default Create Event Value
87	meter_period_close_subscriber	true
88	meter_period_close_group	true
89	contract_late_charge_subscriber	true
90	contract_late_charge_group	true
91	finance_contract_principal_payment_subscriber	true
92	finance_contract_principal_payment_group	true
93	missed_contract_charge_subscriber	true
94	missed_contract_charge_group	true
95	contract_finance_subscriber	true
96	contract_finance_group	true
97	contract_debt_payment_subscriber	true
98	contract_debt_payment_group	true
99	contract_debt_payment_device	true
100	purchased_item_modify_subscriber	true
101	purchased_item_modify_group	true
102	purchased_item_modify_device	true
103	component_meter_period_close_subscriber	true
104	component_meter_period_close_group	true
105	subscription_modify	false
106	group_modify	false
107	user_modify	false
108	device_modify	false
109	subscription_create	false
110	group_create	false
111	device_create	false
112	user_create	false
113	subscription_delete	false
114	group_delete	false
115	device_delete	false
116	user_delete	false
117	component_meter_subperiod_close_subscriber	true
118	component_meter_subperiod_close_group	true
119	status_change_user	true
120	cycle_arrears_recurring_subscriber	true
121	cycle_arrears_recurring_group	true

Table 3-1 Event Mapping Operation Types (continued)

ID	Operation Type	Default Create Event Value
127	debt_payment_subscriber	true
128	debt_payment_group	true
129	purchased_item_status_change_subscriber	true
130	purchased_item_status_change_group	true
131	purchased_item_status_change_device	true
132	fee_charge_subscriber	true
133	fee_charge_group	true
134	external_payment_request_subscriber	true
135	external_payment_request_group	true
136	external_payment_subscriber	true
137	external_payment_group	true
138	period_end_time_change	false
139	external_payment_failure_subscriber	false
140	external_payment_failure_group	false
141	purchase_failure_subscriber	false
142	purchase_failure_group	false
143	purchase_failure_device	false
144	purchased_item_activation_failure_subscriber	false
145	purchased_item_activation_failure_group	false
146	purchased_item_activation_failure_device	false
147	recurring_failure_subscriber	false
148	recurring_failure_group	false
149	balance_transfer_failure_subscriber	false
150	balance_transfer_failure_group	false

Note: For the following operation type pairs, you can enable only one operation type from the pair. If you enable both operation types in the pair, an error is generated.

- object_create_subscriber / subscription_create
- object_create_group / group_create
- object_create_device / device_create
- object_delete_subscriber / subscription_delete
- object_delete_group / group_delete
- object_delete_device / device_delete

Chapter 4

Charging Configuration

Charging Configurations contains interfaces you use to create and manage *balance*, pricing, *offer* and *bundle*, general, and application-related components.

To create balance, pricing, offer and bundle, general, and application-related components in Charging Configurations, you must have the Domain role of Pricing User. To delete these components, you must have both the Domain roles of Pricing User and Pricing Admin.

This chapter includes the following topics:

Application Configuration

Application Configuration components are used by My MATRIXX for creating pricing plan items and are used by MATRIXX Charging Application and MATRIXX Policy Application for charging and rating. Application Configuration components that you can create in the sub-interfaces of this interface include billing cycle profile templates, event types, event mappings, event lifespan codes, notification types, service types, and status life cycle states, transitions, and actions.

You must have the Domain role of Configuration User to create Application Configuration components.

Service Types

Each network message received for subscriber usage is associated with a service type, and each category of usage within the network message is associated with a service context.

Each service type and its service contexts control how that category of usage is authorized, charged for, and aggregated. Additionally, the service type controls the product offers that are applicable for rating the usage and the type of event data record (EDR) that is put in the MEF file. The price components in a product offer all must be of the same service type, but a bundle can contain product offers associated with different service types.

You can create new service types and service contexts as part of application configuration by using My MATRIXX. The definitions are stored as objects in the Pricing database and the service names are displayed in My MATRIXX as nodes in the Price Components tab as descendants of the **Usage** service. Each service type can have its own child service types. For example, a usage service type can be the parent of a Mobile service type, which can be the parent of Satellite, Cellular, and Wi-Fi service types.

A service type can be the parent of any number of service contexts, which support the Diameter Multiple-Services-Credit-Control (MSCC) AVP. They allow you to define information about a per-context basis, including event usage aggregation and *AQM (Adaptive Quota Management)*.

A service context is similar in many ways to a service type, but is different in the following important ways:

- Each network message is associated with exactly one service type and all network messages in a session must use the same service type. (If a single network message or session carries multiple categories of usage, each category may be associated with its own service context.)
- Offers are selected based on the service type and ancestor service types of the network message. If a service type has categories of usage of different quantities (for example, voice-over-IP versus data), offers for that service type may have separate usage components for each quantity. If the service type has multiple cate-

gories of usage of the same quantity, its offers must normalize on category-specific fields in the network message to discriminate based on category.

- MEF event types and event-field mappings are selected by service type, not service context.

Each service type or service context inherits event-type mappings from its ancestor service types. In the following examples illustrating the differences between service types and service contexts, **Voice**, **Data**, and **SMS** (whether defined as service types or service contexts) inherit mappings from **Cellular**, which inherits mappings from **Mobile**, which inherits mappings from **usage**.

Note: Quota management parameters are not inherited.

- A service type named **Cellular** with service contexts **Voice**, **Data**, and **SMS** allows a single Cellular session to carry all three categories of usage. Offers for **usage**, **Mobile**, or **Cellular** service types may apply different pricing for **Voice** and **Data** either by charging for one in seconds and the other in bytes or by normalizing on fields in the network message to determine which category is currently being rated.
- A service type **Cellular** with child service types **Voice**, **Data**, and **SMS** requires a separate session for each of these three categories of usage. However, each service type could have its own offers, event types, event-field mappings, and even context types (for example, the **Data** service type may have context types **Email**, **Video**, and **Web**).

Identifying Services and Contexts in a Network Message

Each Diameter message must be associated with exactly one service type. The service type is identified by the `ServiceTypeObjectId` field of the MtxDiamRoMsg MDC. This field must have a value, and the value must correspond to the ID of a service type object in the Pricing database. Typically, a selective update is configured to assign a value to `ServiceTypeObjectId` based on the value of the Diameter `Service-Context-Id` AVP.

Each category of usage in a Diameter message is represented by an MtxMultiServiceData MDC in the `MultiServiceList` of the MtxDiamRoMsg MDC. For multiple-category (MSCC) messages, one MtxMultiServiceData MDC is generated for each `Multiple-Services-Credit-Control` (MSCC) AVP in the message. For single-category (non-MSCC) messages, one MtxMultiServiceData MDC is generated for the message itself. Each MtxMultiServiceData MDC is associated with a context type ID whose value comes from one of the following, in order of precedence:

1. MSCC `Service-Identifier` AVP
2. MSCC `Rating-Group` AVP
3. External `Service-Identifier` AVP

If there are multiple MSCC contexts, the `ServiceId` is determined by the first AVP to arrive in the Diameter packet (the MSCC Rating-Group AVP or the MSCC Service-Identifier AVP). If the MSCC `Service-Identifier` AVP and the external `Service-Identifier` AVP both have values, the MSCC `Service-Identifier` AVP value is used.

If the context type ID for a category of usage corresponds to the ID of a context type object in the Pricing database, that context type determines how usage is handled. Otherwise, the service type determines how that usage is handled.

For single-category (non-MSCC) messages, assignment of the `ServiceTypeObjectId` may be determined by both `Service-Context-Id` and `Service-Identifier` to place each category of usage into its own service type, rather than as multiple contexts within a single common service type.

Important: If a CREDIT-CONTROL REQUEST (*CCR*) is received with two MSCC AVPs with the same `Service-Identifier` AVP value or a CCR is received with two MSCC AVPs with the same `Rating-Group` AVP value and at least one of the MSCC AVPs does not have a `Service-Identifier` value, the Charging Server logs an error message and rejects the CCR.

A CCR may include the Multiple-Services-Indicator AVP with a value of 0 (MULTIPLE_SERVICES_NOT_SUPPORTED) or 1 (MULTIPLE_SERVICES_SUPPORTED) to indicate whether the client is capable of handling multiple services independently.

- If the CCR indicates that it supports multiple services, its answer (CCA) includes one Multiple-Services-Credit-Control AVP for each requested service with the Result-Code AVP and additional AVPs for that service.
- If the CCR indicates that it does not support multiple services, its CCA includes one Multiple-Services-Credit-Control AVP for the single requested service. It also uses the Result-Code AVP at the message/command level to indicate if the request failed, which results in termination of the underlying session.

If the CCR does not include the Multiple-Services-Indicator AVP, its CCA uses the Result-Code AVP and additional AVPs only at the message/command level.

Create a Service Type

You can add new usage service types to your base configuration that contain configurations specific to that service, including contexts for that service. Services support the Diameter Multiple-Services-Credit-Control (MSCC) Diameter AVP and allow you to define service contexts and quota information. The quantity information for each context can include configurations for event usage aggregation and Adaptive Quota Management (AQM).

About this task

Service contexts, beats, service context beat groups, event field maps, session field maps, and audit field maps are all optional properties of a service type. If they are not specified, MATRIXX Charging Application does not use them during rating. Note that any event field maps defined in a parent service type are inherited by any child service types and are included in their EDRs.

You can create custom service contexts to use during rating. When you create a new service type, an empty default service context is also created that you must configure with default (fallback) parameters in these categories:

- Rating
- Aggregation settings
- Quota settings
- Authorized quantities
- Diameter final unit information
- A pre-rating generator for the service
- A post-rating generator for the service

The parameters in these categories are treated as a unit. During rating, the charging engine selects the appropriate custom service context to use to rate an event. The rating engine uses the custom service context parameters from each category. If it does not find parameters in a category, it uses the parameters from the corresponding category in the default service context.

For example, if any aggregation settings are present in the custom service context selected by the rating engine, they are used. If the custom service context selected does not have any aggregation settings, it uses all aggregation settings from the default service context.

For information about parameters in these categories, see the discussion about service type context properties.

Note: The pre-rating generator and the post-rating generator can only be set or changed in the default service context.

Procedure

1. In Service Types, click the name of the service from which the new service will be derived. You cannot create a sub service of the root service type. You can create a new usage, recurring, or non-recurring service type, and you can create a service type that is already derived from on of these service types.
2. In Edit Service Type, click **New Sub Service Type**.
3. In Create New Sub Service Type, enter a unique name for the service and click **Create**.
4. Specify general properties for the service type.
    a. In **Event Type Name**, select the event type associated with this service. The event type determines the EDR to generate for usage of this service.
    b. (Optional) In **Failed Event Type**, select the event type to use for usage authorization failure of this service type. This may be usage_failure or other custom values.
    c. In **Quantity Type**, select the quantity that is being measured during service usage. Rates are based on this quantity type.
    d. In **Quantity Units**, select the unit of measurement for the quantity. If the quantity type is monetary, select **None** for the quantity unit.
    e. (Optional) In **Audit Data Container Name**, the name of the MDC in which to store audit data. If you specify an audit MDC, you must also define the **Audit Field Mappings** for it. This enables you to specify the source fields from which to copy data into the audit data container.
    f. (Optional) In **Session Data Container Name**, select the name of a session MDC. This enables you to map a field in the incoming message MDC to a field in the session object MDC to save the value throughout a session.
    g. (Optional) To suppress generation of usage events when the event does not have any balance impacts, select **Suppress Usage Event With No Impact**. Usage events are not generated if the rated usage quantity is zero and if there are no balance impacts.
    h. (Optional) To generate session monitor events, select **Generate Session Monitor Event**. After selecting **Generate Session Monitor Event**, the following fields open and require setting:
        - **Session Monitor Interval** — The interval at which an MtxSessionMonitorEvent EDR generates.
        - **Session Monitor Interval Unit** — The measurement unit of the session monitor interval.
        - **Session Monitor Event Type** — The event type for use in generating session monitor events.
    i. Click **Save**.

    The new service is saved with a default service context created for it. Service contexts support the Diameter Multiple-Services-Credit-Control (MSCC) AVP.
5. Click the `default` service context and enter fallback rating values in the default context in these categories:
    - Rating
    - Aggregation Settings
    - Quota Settings
    - Authorized Quantities
    - Final Unit Information
    - PreRating
    - PostRating

The parameters in these categories are treated as a unit. All parameters in a category are used if no corresponding values are specified in a custom service context. For information about parameters in these categories, see the discussion about service type context properties.

Note: The pre-rating generator and post-rating generator can only be changed on the default service context and are automatically used by all custom context type definitions.

6. (Optional) Add one or more custom service contexts to the service type. Custom service contexts are used by the rating engine to rate events. For details about the service context parameters, see the discussion about service type context properties.

    a. Under Service Context Types, click **New Service Context Type** and in **Create New Service Context**, enter a name and ID for the service context and click **Create** to open the Context Type editor.

    The ID must be a unique integer and must match the ID used by the network device.

    b. Add values for parameters in these categories:

    - Rating
    - Aggregation settings
    - Quota settings
    - Authorized quantities
    - Diameter final unit information

7. (Optional) Under Beat Definitions, click **New Beat Definition** to add one or more beats to the service.

    A beat defines an incremental number of measurement units to which a rate applies, for example 30 kilobytes or 1 minute. My MATRIXX uses these values to populate the beat lists in the Rating Formula interface. You can assign beat values during catalog creation.

    a. In the Beat Definition editor, enter a name for the beat.

    b. (Optional) In **Quantity Units**, select the measurement unit for the beat. The unit is dependent on the quantity type of the event, but not the quantity unit of the event. For example, the event might be measured in megabytes, but the beat can be applied for a number of bytes. If the event has a **Quantity Units** value of none the beat **Quantity Units** is also none.

    c. In **Quantity Multiple**, enter the quantity amount at which to apply the beat. The value must be a positive integer.

8. (Optional) If you defined service contexts for the service, define beat groups for them.
    a. Under Service Context Beat Group Array, click **New Service Context Beat Group**.

    b. In the Service Context Beat Group editor, enter a name for the beat group the window that opens, enter a name for the beat group and enter the IDs of each service context in the beat group as a comma-separated list.

    A Beat Group identifies two or more service contexts that can share the same beat cache amount during the same usage session. For example, say a beat group has two service contexts, A and B. If the beat is 5KB and context A has a total usage amount of 3KB, the 3KB is charged to the balance and 2KB is available for further usage by context A or by context B. When the beat cache is fully consumed, the cycle continues with each additional beat applied for usage during that session. After charging a full beat for usage by a service context, any unused portion of that beat is applied toward subsequent usage in the same service context or beat group. This includes usage after re-authorizations when reporting QHT and FINAL.

9. (Optional) Under Event Field Mappings, click **New Field Mapping** to add one or more event field mappings to this service. This allows you to add fields to the EDR created for the **Event Type** identified.

a. Set values for the following event field mappings properties.
   - Source Container
   - Source Field
   - (Optional) Source Tier
   - Destination Container
   - Destination Field

b. Set the **Group Aggregation?** field.

c. click **Save**.

   For more information about these properties, see the discussion about event field mapping properties.

For information about how information is included in EDRs, see the discussion about event field mapping properties.

10. (Optional) Under Failed Event Field Mappings, click **New Failed Event Field Mapping** to add one or more failed event field mappings to this service.

    a. Set values for the following failed event field mappings properties:
       - Source Container
       - Source Field
       - (Optional) Source Tier
       - Destination Container
       - Destination Field

    b. Set the **Group Aggregation?** field.

    c. Click **Save**.

       For more information about these properties, see the discussion about failed event field mapping properties.

11. If you specified an Audit Container Name, under Audit Data Field Mappings, click **New Audit Field Mapping** to add one or more audit data field mappings to this service. This adds the fields to the audit data MDC after service usage.

    Set values for the following audit data field mapping properties and click **Save**. For more information about these properties, see the discussion about audit data field mapping properties.
    - Source Container
    - Source Field
    - (Optional) Source Tier
    - Destination Container
    - Destination Field

12. If you specified a Session Data Container Name, after clicking **Save**, under Session Field Mappings, click **New Field Mapping** to add one or more session field mappings to this service. This causes the field to persist throughout the service session and makes it available for normalizations.

Set values for the following session field mappings properties and click **Save**. For more information about these properties, see the discussion about session field mapping properties.

- Source Container
- Source Field
- Destination Container
- Destination Field
- Update Values

13. (Optional) If you selected **Generate Session Monitor Event**, click **New Session Monitor Event Field Mapping** to add one or more event field mappings to this service.
    a. Set values for the following session monitor event field mapping properties.
        - Source Container
        - Source Field
        - (Optional) Source Tier
        - Destination Container
        - Destination Field
    b. Click **Save**.

    For more information about these properties, see the discussion about generating session monitor event mapping properties.

Results

The pricing administrator must restart the My MATRIXX session and update the workspace to view the new service type.

Note: The Diameter Gateway maps the `ExternalServiceType` field in the network message to the MATRIXX `ServiceTypeObjectId` MDC field. If your system does not store the service type in the `ExternalServiceType` field, create a SED File to map the external service type to the new service type you created.

What to do next

- To add the new service type to the Pricing database, compile and load the pricing configuration file.
- If you specified a custom MDC and have not yet added it to the base configuration, run the create config.py script to create the new MDC and update the engine configuration.

Change a Service Type

You can change service types and sub service types, and their event and audit data field mappings. Changes to service type names also changes the path of that service type and any the paths of its sub services.

Procedure

1. In Service Types, click the name of the service type to change.
2. You can change any properties other than the Name, Type ID, Parent ID, and ID.

Note: You cannot delete the default service context type.

3. Click **Save**.
4. (Optional) Change event field mappings.
    a. Scroll down to the **Event Field Mappings** table.
    b. Select a mapping from the **Source** column.

    In **Edit Event Field Mapping**, change any of the following properties:

    - Source Container
    - Source Field
    - Destination Container
    - Destination Field
    - Source Tier
    - Group Aggregation?

    For more information about these properties, see the discussion about event field mapping properties.

    c. Click **Save**.
5. (Optional) Change failed event field mappings.
    a. Scroll down to the **Failed Event Field Mappings** table.
    b. Select a mapping from the **Source** column.

    In **Edit Event Field Mapping**, change any of the following properties:

    - Source Container
    - Source Field
    - Destination Container
    - Destination Field
    - Source Tier
    - Group Aggregation?

    For more information about these properties, see the discussion about failed event field mapping properties.

    c. Click **Save**.
6. (Optional) Change audit data field mappings.
    a. Scroll down to the **Audit Data Field Mappings** table.
    b. Select a mapping from the **Source** column.

    In **Edit Audit Data Field Mapping**, change any of the following properties:

- Source Container
- Source Field
- Destination Container
- Destination Field
- Source Tier

For more information about these properties, see the discussion about event field mapping properties.

c. Click **Save**.

7. (Optional) Change session field mappings.

a. Scroll down to the **Session Field Mappings** table.

b. Select a mapping from the **Source** column.

In **Edit Session Field Mapping**, change any of the following properties:

- Source Container
- Source Field
- Destination Container
- Destination Field
- Updated Value

For more information about these properties, see the discussion about session field mapping properties.

c. Click **Save**.

8. (Optional) Change generate session monitor event field mappings.

a. Make any changes to the following:

- **Session Monitor Interval** — The interval at which an MtxSessionMonitorEvent EDR generates.
- **Session Monitor Interval Unit** — The measurement unit of the session monitor interval.
- **Session Monitor Event Type** — The event type for use in generating session monitor events.

b. Click **Save**.

c. Scroll down to the **Session Monitor Event Field Mappings** table.

d. Select a mapping from the **Source** column.

In **Edit Session Monitor Event Field Mapping**, change any of the following properties:

- Source Container
- Source Field
- Destination Container
- Destination Field
- Source Tier

For more information about these properties, see the discussion about generate session monitor event field mapping properties.

e. Click **Save**.

Service Type Properties

A service type definition includes information about the service quantity, service context types, beats, fields to add to an EDR, and fields to add to a session object. A service context can be configured with usage aggregations, quota management and authorization behavior, and final unit indication behavior.

General Service Type Properties

Table 4-1 *(on page 38)* lists the properties that define standard the attributes of the service.

Table 4-1 General Service Type Properties

Attribute	Description
Full Name	The fully qualified name of the service type. This name is created automatically and is used to identify the service parent. You cannot set or change this value.
Type ID	The service type ID, which is generated automatically by My MATRIXX and based off of the parent service type ID. This must map to the corresponding service ID used by the GGSN. You cannot set or change the value.
Parent ID	The ID of the parent service, from which this service is a descendent. This value is derived from the type of service you create. You cannot set or change this value.
ID	The service type ID used by My MATRIXX and saved to the pricing database. You cannot set or change this value.
Name	A descriptive name for the service type. This name is added to the end of the fully qualified name of the service, after any parent service types. It is also displayed in My MATRIXX.
Event Type Name	The name of the event associated with the service type. The event type name determines the EDR to generate for usage of this service. **Important:** Although the ROOT value is listed as an event type, it is not valid and will result in errors if you use it for a service type.
Failed Event Type	Select usage_failure or none. If you select usage_failure, after you save the general properties, you can map the failure events in the Failed Event Field Mappings table. For information about creating a new failed event field mapping, see the discussion about failed event type mappings.
Quantity Type	The quantity type that is measured for the service usage, on which to base rates. Only price components with this quantity type are considered applicable during rating. • The service type's quantity type is only a default for setting up pricing formulas. Each context type within a service type can have a different quantity type.

Table 4-1 General Service Type Properties (continued)

Attribute	Description
Quantity Units	The measurement unit of the service type. The unit is dependent on the Quantity Type and can be one of the following: • in_data, out_data, and total_data: bytes, mbytes, kbytes, gbytes • actual_duration and active_duration: seconds, minutes, hours, days, weeks • service_specific: none. This value is also used when the quantity is 0, as with text messages.
Audit Data Container Name	(Optional) The name of the MDC in which to store audit data. This can be the base MtxAuditData MDC, which has the Aggregation ID or event ID and the raw usage quantity, or a custom MDC based off of it. If you specify an audit MDC, you must also define the Audit Field Mappings for it. This enables you to specify the fields that contain the data you want to copy to the audit data container.
Session Data Container Name	(Optional) The name of a session extension MDC in which to copy data from the incoming message MDC. This enables you to persist the field value throughout a session and use it as the basis of normalizations. The session container can be the base MtxChargingSessionExtension MDC or a custom MDC based off of it. The session extension MDC is stored in the Attr field. If you specify a Session Container Name, you must also define the Session Field Mappings for it. The destination MDC in the field mapping can be the same MDC specified here based off of that MDC.
Suppress Usage Event With No Impact	(Optional) To suppress generation of usage events when the event does not have any balance impacts, select **Suppress Usage Event With No Impact**. Usage events will not be generated if the rated usage quantity is zero and if there are no balance impacts. Note that aggregated usage events are always generated, even if the aggregated usage quantity is zero. Enabling this option may be helpful in scenarios where there is a high load on the mediation system due to many usage events with zero data consumption. For backwards compatibility, the default for this option is not set.
Generate Session Monitor Event	The existing open charging sessions generates Session monitor events per the configuration in Service Type definition at the time of current session start time. The new sessions use the updated information from the Service Type definition. For example, event generation is changed from every 2 hours to every 3 hours , or event generation is disabled, in the middle of the charging session, then the current charging session continues to generate events every 2 hours. Once the current session ends, subsequent charging sessions generate events every 3 hours, or will not generate any session monitor events.

Table 4-1 General Service Type Properties (continued)

Attribute	Description
	After selecting Generate Session Monitor Event, you must set the following fields: • **Session Monitor Interval** — The interval at which an MtxSessionMonitorEvent EDR generates. • **Session Monitor Interval Unit** — The measurement unit of the session monitor interval. • **Session Monitor Event Type** — The event type for use in generating session monitor events.

Event and Failed Event Field Mapping Properties

Note that any event field maps defined in a parent service type are inherited by any child service type and included in its EDR. Table 4-2 *(on page 40)* list the properties you can set.

Table 4-2 Field Mapping Properties

Attribute	Definition	Data Type	Default
Source Container	The name of the MDC containing the field to add to the EDR. This is required if the event type specified or the parent of this event type, does not already contain the field. The Source Container Name must be one of the following MDCs or an extension of it: • Mtx5GRequest • Mtx5GResponse • Mtx5GMultiUnitUsageData • Mtx5GMultiUnitInfoData • MtxChargingSessionExtension • MtxDeviceObject • MtxGroupObject • MtxMsg • MtxMultiServiceData • MtxPolicySessionExtension • MtxSubscriberObject • MtxTenantProfile • MtxUserObject • MtxUserExtension The `MtxMultiServiceData` MDC allows administrators to populate EDRs with fields from specific service contexts. The source field	String	-

Table 4-2 Field Mapping Properties (continued)

Attribute	Definition	Data Type	De-fault
	must be located in the `MultiServiceList` element (in the message MDC) for a service context.		
Source Field	The name of the MDC field to add to the EDR. The value is the full path of the field in the MDC, separated by dots (.). For example, if a field is nested in an array, the syntax is: *array_field_name*.*event_field_name* **Note:** You can map DateTime fields into aggregated events as aggregation grouping fields or non-grouping fields. Use caution when mapping DateTime fields as grouping fields to avoid many different aggregations.	String	-
Source Tier	(Optional) The name of a Tier (level) within a group hierarchy if the source container is a group object. Only group objects above the event initiator in the hierarchy can be source containers. Set the source tier in situations where there is a field, such as an account number associated with a group, that must be included in the sub-scriber events of its group members.	String	-
Destination Container	The name of the MDC containing the event field, for example, MtxVoiceEvent. This value is located in the Event Types list in Appli-cation Configuration. The container name can be a super class of the container for the event type.	String	-
Destination Field	The name of the event object field in which to copy the source field value. The value is the full path of the field in the MDC, separated by dots (.). For example, if a field is nested in an array, the syntax is: *array_field_name*.*event_field_name*	String	-
Group Aggregation?	Configures field-based aggregation of usage data based on the field identified in the Destination Field name. This property is used when event aggregation is enabled. When not aggregating, it has no effect. A value of `true` or `false`.	Boolean	false

Index