FinOps Open Cost and Usage Specification

Version

Publication version 1.4

Status of This Document

This section describes the status of this document at the time of its publication.

This is a published release of the FinOps Open Cost and Usage Specification.

This document was produced by a group operating under the Joint Development Foundation Projects agreement. FOCUS maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information.

Abstract

FOCUS is an open specification for billing data. It defines a common schema for billing data, aligns terminology with the FinOps Framework and defines a minimum set of requirements for billing data. The specification provides clear guidelines for billing data generators to produce FinOps-serviceable data. The specification enables FinOps practitioners to perform common FinOps capabilities such as chargeback, cost allocation, budgeting and forecasting etc. using a generic set of instructions, regardless of the origin of the FOCUS compatible dataset.

Document Use License

Table of Contents

1. Introduction

> Note: The following section is informative and non-normative. It does not define requirements.

FOCUS is a standards development organization (SDO) formed to establish an open, consensus-driven standard for billing data. In the absence of a broadly adopted standard, infrastructure and *service providers* have relied on proprietary billing schemas and inconsistent terminology, making cost data difficult to normalize and act upon across environments. This lack of conformance has forced FinOps *practitioners* to develop best-effort custom normalization schemes for each provider, in order to perform essential FinOps capabilities such as chargeback, cost allocation, budgeting and forecasting.

The FOCUS Specification, developed by a global community of practitioners and vendors, defines a consistent, vendor-neutral approach to billing data. It is designed to improve interoperability between service providers, reduce operational complexity, and enable greater transparency in cloud and SaaS cost management.

1.1. Background and History

This project is supported by the FinOps Foundation. This work initially started under the Open Billing working group under the FinOps Foundation. The decision was made in Jan 2023 to begin to migrate the work to a newly formed project under the Linux Foundation called the FinOps Open Cost and Usage Specification (FOCUS) to better support the creation of a specification.

1.2. Intended Audience

This specification is designed to be used by three major groups:

- Billing data generators: Entities that present consumption-based billing information related to infrastructure and *service providers*, such as (but not limited to):
  - Cloud Service Providers (CSPs)
  - Software as a Service (SaaS) platforms
  - Managed Service Providers (MSPs)
  - Internal infrastructure and service platforms
- FinOps tool *providers*: Organizations that provide tools to assist with FinOps
- FinOps practitioners: Organizations and individuals consuming billing data for doing FinOps

1.3. Scope

The FOCUS working group will develop an open-source specification for billing data. The schema will define data *dimensions*, *metrics*, a set of attributes about billing data, and a common lexicon for describing billing data.

1.4. Design Principles

The following principles were considered while building the specification.

1.4.1. FOCUS is an Iterative, Living Specification

- Incremental iterations of the specification released regularly will provide higher value to practitioners and allow feedback as the specification develops. The goal is not to get to a complete, finished specification in one pass.

1.4.2. Working Backward with Ease of Adoption

- Aim to work backward from essential FinOps capabilities that practitioners need to perform to prioritize the dimensions, metrics and attributes of the cost and usage data that should be defined in the specification to fulfill that capability.
- Be FinOps scenario-driven. Define columns that answer scenario questions; don't look for scenarios to fit a column, each

column must have a use case.
- Don't add dimensions or metrics to the specification just because it can be added.
- When defining the specification, consideration should be made to existing data already in the major cloud service providers' (AWS, GCP, Azure, OCI) datasets.
- As long as it solves the FinOps use case, there should be a preference to align with data that is already present in a majority of the major data generators.
- Strive for simplicity. However, prioritize accuracy, clarity, and consistency.
- Strive to build columns that serve a single purpose, with clear and concise names and values.
- The specification should allow data to be presented free from jargon, using simple understandable terms, and be approachable.
- Naming and terms used should be carefully considered to avoid using terms for which the definition could be confused by the reader. If a term must be used which has either an unclear or multiple definitions, it should be clarified in the glossary.
- The specification should provide all of the data elements necessary for the Capabilities.

1.4.3. Provider-Neutral Approach by Default

- While the schema, naming, terminology, and attributes of many service providers are reviewed during development, this specification aims to be service-provider-neutral.
- Contributors must take care to ensure the specification examines how each decision relates to each of the major cloud service providers and SaaS vendors, not favoring any single one.
- In some cases, the approach may closely resemble one or more service provider's implementations, while in other cases, the approach might be new. In all cases, the FOCUS group (community composed of FinOps practitioners, Cloud and SaaS providers and FinOps vendors) will attempt to prioritize enabling FinOps Capabilities and alignment with the FinOps Framework.

1.4.4. Extensibility

The FOCUS Specification is designed to support evolving FinOps needs across diverse billing models and service provider types.

While the initial focus was on billing data from Cloud Service Providers (CSPs), subsequent versions introduced foundational support for Software as a Service (SaaS) and Platform as a Service (PaaS) providers, including normative columns for pricing currencies, effective cost, and contracted pricing in non-monetary units such as credits or tokens.

The specification supports extensibility through structured naming conventions (e.g., x_ custom columns), conditional requirements, and a version-aware schema approach.

Future versions of FOCUS will consider including additional FinOps capabilities such as forecasting, exchange rate modeling, and anomaly detection, while continuing to support a broader range of billing and cost datasets — including internal infrastructure platforms and marketplace offerings.

1.5. Design Notes

1.5.1. Optimize for Data Analysis

- Optimize columns for data analysis at scale and avoid the requirement of splitting or parsing values.
- Avoid complex JSON structures when an alternative columnar structure is possible.
- Facilitate the inclusion of data necessary for a system of record for cost and usage data to consume.

1.5.2. Consistency Helps with Clarity

- Where possible, use consistent names that will naturally create associations between related columns in the specification.
- Column naming must strictly follow the FOCUS Column Handling requirements.
- Use established standards (e.g., ISO8601 for dates, ISO4217 for currency).

1.6. Typographic Conventions

The keywords `MUST`, `MUST NOT`, `SHOULD`, `SHOULD NOT`, and `MAY` in this specification are to be interpreted as described in BCP14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

1.7. FOCUS Feature Level

Under each column defined in the FOCUS specification, there exists a 'Feature level' designation that describes the column as `Mandatory`, `Conditional`, or `Optional`. Feature level is designated based on the following criteria described in the normative requirements in each column definition:

- If the existence of a column is described with `MUST` with no conditions of when it applies, then the feature level is designated as `Mandatory`.
- If the existence of a column is described as `MUST` with conditions of when it applies, then the feature level is designated as `Conditional`.
- If the existence of a column is described as `SHOULD`, then the feature level is designated as `Recommended`.
- If the existence of a column is described as `MAY`, then the feature level is designated as `Optional`.

1.8. Conformance Checkers and Validators

Validation tools may be employed to determine conformance of data and implementations per this specification.

The FinOps Foundation maintains a validator called the FOCUS Validator which it uses for its own conformance assessments and serves as a reference implementation to support validation activities.

Other validation tools may be developed and made available by third parties. The FOCUS specification does not mandate the use of any particular tool, nor does it maintain a registry of available validators.

2. Supported Features

Note: The following section is informative and non-normative. It does not define requirements.

The FOCUS specification is designed to meet the needs of FinOps practitioners in numerous scenarios. The following section contains features supported by the FOCUS specification. This list does not represent all possible combinations or use of FOCUS data but does represent core capabilities that the FOCUS specification supports.

Supported Feature List

Feature	Description
Account Structures	Supports breaking costs down by billing and sub-accounts to facilitate chargeback and budgeting scenarios.
Billed Cost and Invoice Alignment	Ensures data is consistent with payable invoices regarding total cost and the period of time covered.
Charge Categorization	Supports classification of charges including purchases, usage, tax, credits, and adjustments.
Commit Usage and Under Usage	Tracks the usage and under-usage of commitment discounts and capacity reservations.
Commitment Program Eligibility Details	Identifies which commitment programs each charge qualifies for, supporting coverage rate analysis and uncovered savings identification.
Contract Commitments	Tracks commitments made via contractual agreements using identifiers joined between Cost and Usage and Contract Commitment datasets.
Cost and Usage Attribution	Facilitates the inclusion of provider-defined or user-defined metadata (tags) at a row level for organizational analysis.
Cost Comparison	Supports comparing Billed, Contracted, Effective, and List cost columns to identify savings or amortization.
Custom Columns	Allows the inclusion of additional columns to facilitate reporting capabilities not covered by the standard specification.
Data Generator-Calculated Split Cost Allocation	Enables tracking resources split by internal consumption metrics, common for shared clusters like Kubernetes.
Data Granularity	Supports multiple levels of granularity, from high-level account charges down to individual resource-level data.
Dataset Instance Metadata	Provides metadata describing dataset artifacts, unique identifiers, and alignment with specific FOCUS datasets.
Effective Cost Analysis	Enables analysis of costs after discounts and the amortization of upfront fees to track spending trends.
Invoice Reconciliation	Reconciles Cost and Usage records with Invoice Detail line items to validate that usage costs map back to invoiced amounts.
Location	Provides structured data for regions and availability zones to analyze costs based on deployment geography.
Marketplace Purchases	Supports analysis of marketplace purchase data and reporting Effective Cost for service provider usage.
Participating Entity Identification	Allows identification of entities involved in hosting, invoicing, and data generation (e.g., Service Provider vs. Host Provider).
Recency Metadata	Indicates what portion of a dataset is complete and how recently it was updated to inform FinOps functions like chargeback.
Resource Usage	Enables tracking consumption by providing information on which resources were used, in what quantities, and with what units.
Schema Metadata	Communicates important attributes about data structure, types, and versions to facilitate structure changes.
Service Categorization	Standardizes the classification of services into high-level functional categories and granular subcategories.
Service Provider Services	Aligns costs with familiar service and product offering names for easier reporting and verification.
Verification, Comparison, and Fluctuation Tracking of Unit Prices	Facilitates verification of List and Contracted unit prices and tracks fluctuations over time.

2.1. Account Structures

2.1.1. Description

Different service providers have different account constructs that FinOps practitioners use for allocation, reporting, and more. Organizations may have one or many accounts within one or more service providers and FinOps practitioners may need to review the cost broken down by each account. FOCUS has two types of accounts: a billing account and a sub account.

A billing account is the account where invoices are generated. Each billing account can have one or more sub accounts, which can be used for deploying and managing resources and services. Billing and sub accounts are often used to facilitate allocation strategies and FinOps practitioners must be able to break costs down by billing and sub account to facilitate FinOps scenarios like chargeback and budgeting.

2.1.2. Directly Dependent Columns

- BilledCost
- BillingAccountId
- BillingAccountName
- BillingAccountType
- SubAccountId
- SubAccountName
- SubAccountType

2.1.3. Supporting Columns

- InvoiceId

2.1.4. Example SQL Query

```
SELECT
  BillingAccountId,
  BillingAccountName,
  BillingAccountType,
  SubAccountId,
  SubAccountName,
  SubAccountType,
  SUM(BilledCost)
FROM focus_data_table
WHERE BillingPeriodStart >= ? AND BillingPeriodEnd < ?
GROUP BY
  BillingAccountId,
  SubAccountId
```

2.1.5. Version Introduced

0.5

2.2. Billed Cost and Invoice Alignment

2.2.1. Description

FOCUS data should be consistent with the costs indicated on payable invoices. This is relevant to the total cost of the invoice, as well as the period of time the invoice covers.

2.2.2. Directly Dependent Columns

- BilledCost
- BillingCurrency
- BillingPeriodEnd
- BillingPeriodStart
- InvoiceId

2.2.3. Supporting Columns

- ServiceName

2.2.4. Example SQL Query

```
SELECT
  BillingPeriodStart,
  BillingPeriodEnd,
  InvoiceId,
  SUM(BilledCost)
FROM focus_data_table
GROUP BY
  BillingPeriodStart,
  BillingPeriodEnd,
  InvoiceId
```

2.2.5. Version Introduced

0.5

2.3. Charge Categorization

2.3.1. Description

FOCUS supports the categorization of charges including purchases, usage, tax, credits and adjustments. It includes classification on frequency. It includes classification on correction vs normal entries.

2.3.2. Directly Dependent Columns

- ChargeCategory
- ChargeClass
- ChargeFrequency

2.3.3. Supporting Columns

- BilledCost
- BillingAccountId
- BillingPeriodEnd
- BillingPeriodStart

- CommitmentDiscountId
- CommitmentDiscountType
- ServiceProviderName
- ServiceCategory

2.3.4. Example SQL Query

2.3.4.1. Report on Commitment Discount Purchases

```
SELECT
  MIN(ChargePeriodStart) AS ChargePeriodStart,
  MAX(ChargePeriodEnd) AS ChargePeriodEnd,
  ServiceProviderName,
  BillingAccountId,
  CommitmentDiscountId,
  CommitmentDiscountType,
  CommitmentDiscountUnit,
  CommitmentDiscountQuantity,
  ChargeFrequency,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE ChargePeriodStart >= ? AND ChargePeriodEnd < ?
  AND ChargeCategory = 'Purchase'
  AND CommitmentDiscountId IS NOT NULL
GROUP BY
  ServiceProviderName,
  BillingAccountId,
  CommitmentDiscountId,
  CommitmentDiscountType,
  CommitmentDiscountUnit,
  CommitmentDiscountQuantity,
  ChargeFrequency
```

2.3.4.2. Report on Corrections

```
SELECT
  ServiceProviderName,
  BillingAccountId,
  ChargeCategory,
  ServiceCategory,
  ServiceName,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE BillingPeriodStart >= ? AND BillingPeriodEnd < ?
  AND ChargeClass = 'Correction'
GROUP BY
  ServiceProviderName,
  BillingAccountId,
  ChargeCategory,
  ServiceCategory,
  ServiceName
```

2.3.4.3. Report Recurring Charges

```
SELECT
  BillingPeriodStart,
```

```
  CommitmentDiscountId,
  CommitmentDiscountName,
  CommitmentDiscountType,
  ChargeFrequency,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE BillingPeriodStart  >= ? AND BillingPeriodStart < ?
  AND ChargeFrequency = 'Recurring'
  AND CommitmentDiscountId IS NOT NULL
GROUP BY
  BillingPeriodStart,
  CommitmentDiscountId,
  CommitmentDiscountName,
  CommitmentDiscountType,
  ChargeFrequency
```

2.3.5. Version Introduced

1.0

2.4. Commit Usage and Under Usage

2.4.1. Description

FOCUS supports the tracking of commitment discounts usage and under usage, which can come in the form of commitment discounts or capacity reservations.

2.4.2. Directly Dependent Columns

- CommitmentDiscountID
- CommitmentDiscountStatus
- CommitmentDiscountType
- CapacityReservationID
- CapacityReservationStatus
- CapacityReservationType

2.4.3. Supporting Columns

- BilledCost
- ChargePeriodStart
- ChargePeriodEnd
- EffectiveCost
- ServiceCategory

2.4.4. Example SQL Query for Commitment Discounts

```
SELECT
  ServiceProviderName,
  BillingAccountId,
  CommitmentDiscountId,
  CommitmentDiscountType,
  CommitmentDiscountStatus,
  SUM(BilledCost) AS TotalBilledCost,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
```

```
WHERE ChargePeriodStart >= ? AND ChargePeriodEnd < ?
 AND CommitmentDiscountStatus = 'Unused'
GROUP BY
 ServiceProviderName,
 BillingAccountId,
 CommitmentDiscountId,
 CommitmentDiscountType
```

2.4.5. Example SQL Query for Capacity Reservations

```
SELECT
 ServiceProviderName,
 BillingAccountId,
 CapacityReservationId,
 CapacityReservationStatus,
 SUM(BilledCost) AS TotalBilledCost,
 SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ChargePeriodStart >= ? AND ChargePeriodEnd < ?
 AND CapacityReservationStatus = 'Unused'
GROUP BY
 ServiceProviderName,
 BillingAccountId,
 CapacityReservationId,
 CapacityReservationStatus
```

2.4.6. Version Introduced

1.0

2.5. Commitment Program Eligibility Details

2.5.1. Description

FOCUS supports the identification of *charges* in the Cost and Usage dataset that are eligible for *commitment programs*. The Commitment Program Eligibility Details column captures which *commitment programs* a charge qualifies for, regardless of whether a *commitment* is currently applied. This enables practitioners to calculate eligibility-adjusted commitment coverage rates, identify uncovered savings opportunities, and compare commitment options across providers.

CommitmentProgramEligibilityDetails contains a JSON object with a FOCUS-defined top-level key `CommitmentPrograms` containing an array of objects. Each object has a `ProgramType` property identifying the specific *commitment program*. Both discount-bearing programs (e.g., Flexible Spend Plans, Resource Reservations) and capacity-reservation programs (e.g., Advance Resource Commitments) are included in the same array, distinguished by their `ProgramType` value. *Data generators* may include additional custom keys (prefixed with x_) to pass through extra metadata or provider-specific attributes related to the eligibility. Per the column requirements, service providers may include negotiated *commitment programs* when the usage is eligible and the program is not broadly applicable across the service provider's service catalog. For more information, see the CommitmentProgramEligibilityDetails column definition.

2.5.1.1. Naming Conventions for ProgramType Values

The `ProgramType` property identifies *commitment programs* supported by the provider using readable display names. Per the column requirements, these values:

- Equal CommitmentDiscountType for one object in the CommitmentPrograms array when CommitmentDiscountType is not null.
- Correspond to terminology disclosed by the service provider in public documentation. This guidance is especially relevant for SaaS providers that do not itemize commitment discount application at the row level, where

CommitmentDiscountType is typically not populated.
- Do not encode period length, payment option, or other commitment attributes (e.g., use "Flexible Spend Plan" rather than "1 Year Flexible Spend Plan No Upfront").

2.5.2. Directly Dependent Columns

- CommitmentProgramEligibilityDetails

2.5.3. Supporting Columns

- BilledCost
- CapacityReservationId
- CapacityReservationStatus
- ChargeCategory
- ChargePeriodEnd
- ChargePeriodStart
- CommitmentDiscountId
- CommitmentDiscountStatus
- CommitmentDiscountType
- EffectiveCost
- ServiceName
- ServiceProviderName

2.5.4. Example SQL Queries

> Note: The following examples are informative and non-normative. They do not define requirements.

The FOCUS specification implements commitment eligibility via the CommitmentProgramEligibilityDetails column, which is defined in *JSON object format*.

Because ANSI SQL does not define a standard for parsing JSON, the following queries use BigQuery Standard SQL JSON functions (e.g., `JSON_VALUE`, `JSON_EXTRACT_ARRAY`, `UNNEST`). Similar functions are available in all major SQL engines; the examples can be adapted to accommodate any particular database instance. Non-JSON constructs (CTEs, `NULLIF`) are ANSI SQL and should work without modification.

> Important Consideration: The following queries assume FOCUS-conformant dataset artifacts. Practitioners should verify provider conformance before relying on these queries. Non-conformant dataset artifacts may produce inaccurate results.

Note: The `CommitmentPrograms` array contains all *commitment program* types, including both discount-bearing programs and *capacity reservations*. The first three queries below focus on discount-bearing programs and use CommitmentDiscountId to determine coverage. Capacity reservations are fundamentally different: they secure resource availability rather than provide discounts, and are tracked via CapacityReservationId and CapacityReservationStatus. A separate query for capacity reservation analysis follows. Providers using only custom (`x_`-prefixed) top-level keys would require modified JSON paths.

2.5.4.1. Identify Eligible Uncovered Spend by Program Type

This query identifies *charges* that are eligible for *commitment programs* but are not currently covered by a *commitment discount*. A practitioner running relevant workloads can use this to quantify the savings opportunity per *commitment program* type (e.g., "Flexible Spend Plan" vs. "Resource Reservation") and per *service*.

The query filters to "Usage" charges where CommitmentProgramEligibilityDetails is populated (the charge is eligible) and CommitmentDiscountId is null (no *commitment* is applied). It then expands the `CommitmentPrograms` array to aggregate eligible spend per `ProgramType`. This query uses BilledCost rather than EffectiveCost because the charges are uncovered (CommitmentDiscountId IS NULL), so BilledCost reflects the actual amount paid and the savings opportunity.

Note: When a charge is eligible for multiple *commitment program* types, it appears once per eligible type. Costs are not deduplicated across types, since each type represents an independent purchasing opportunity.

```
SELECT
  CU.ServiceProviderName,
  CU.ServiceName,
```

```
  JSON_VALUE(CP, '$.ProgramType') AS EligibleProgramType,
  SUM(CU.BilledCost) AS EligibleUncoveredCost
FROM focus_data_table CU
CROSS JOIN
  UNNEST(JSON_EXTRACT_ARRAY(CU.CommitmentProgramEligibilityDetails, '$.CommitmentPrograms')) AS CP
WHERE CU.ChargePeriodStart >= ? AND CU.ChargePeriodEnd < ?
  AND CU.ChargeCategory = 'Usage'
  AND CU.CommitmentDiscountId IS NULL
  AND CU.CommitmentProgramEligibilityDetails IS NOT NULL
  -- Replace with provider-specific discount-bearing program types
  AND JSON_VALUE(CP, '$.ProgramType') IN ('Flexible Spend Plan', 'Resource Reservation')
GROUP BY
  CU.ServiceProviderName,
  CU.ServiceName,
  JSON_VALUE(CP, '$.ProgramType')
ORDER BY EligibleUncoveredCost DESC
```

2.5.4.2. Calculate Commitment Discount Coverage Rate with Eligibility-Adjusted Denominator

This query computes a *commitment discount* coverage rate using only eligible *charges* as the denominator. Without eligibility data, practitioners typically divide covered spend by total spend, which produces a coverage rate that includes ineligible charges (e.g., storage services, support fees) in the denominator and may not reflect the actionable coverage opportunity. This query targets discount-bearing programs only; for *capacity reservation* utilization, see the capacity reservation query below.

Note: Unused commitment rows (CommitmentDiscountStatus = "Unused") have CommitmentDiscountId populated and will artificially inflate both the numerator and the denominator if left in the dataset. To calculate a true, utilization-adjusted coverage rate, practitioners should additionally filter out these rows (e.g., AND CU.CommitmentDiscountStatus != 'Unused').

```
WITH CommitmentDiscountEligible AS (
  SELECT
    CU.ServiceProviderName,
    CU.EffectiveCost,
    CU.CommitmentDiscountId
  FROM focus_data_table CU
  WHERE CU.ChargePeriodStart >= ? AND CU.ChargePeriodEnd < ?
    AND CU.ChargeCategory = 'Usage'
    AND (
      -- Include covered rows in the denominator
      CU.CommitmentDiscountId IS NOT NULL
      -- If uncovered, check if the JSON array contains an eligible program type
      OR EXISTS (
        SELECT 1
        FROM UNNEST(JSON_EXTRACT_ARRAY(CU.CommitmentProgramEligibilityDetails, '$.CommitmentPrograms')) AS CP
        WHERE JSON_VALUE(CP, '$.ProgramType') IN ('Flexible Spend Plan', 'Resource Reservation')
      )
    )
)
SELECT
  ServiceProviderName,
  SUM(CASE WHEN CommitmentDiscountId IS NOT NULL THEN EffectiveCost ELSE 0 END) AS CoveredCost,
  SUM(EffectiveCost) AS EligibleCost,
  SUM(CASE WHEN CommitmentDiscountId IS NOT NULL THEN EffectiveCost ELSE 0 END)
    / NULLIF(SUM(EffectiveCost), 0) AS CommitmentCoverageRate
FROM CommitmentDiscountEligible
GROUP BY ServiceProviderName
```

2.5.4.3. Compare Commitment Opportunities Across Providers (Cross-Provider with SaaS)

This query aggregates eligible spend and uncovered eligible spend across all providers, including SaaS platforms. A practitioner managing both CSP and SaaS workloads can identify where uncovered eligible spend is concentrated across

commitment program types.

Note: As with the first query above, when a charge is eligible for multiple *commitment program* types, it appears once per eligible type. Removing the EligibleProgramType grouping without deduplicating would inflate totals.

```
WITH CommitmentDiscountEligible AS (
  SELECT
    CU.ServiceProviderName,
    JSON_VALUE(CP, '$.ProgramType') AS ProgramType,
    CU.EffectiveCost,
    -- EffectiveCost = BilledCost for uncovered rows; used here for ratio consistency
    CASE WHEN CU.CommitmentDiscountId IS NULL THEN CU.EffectiveCost ELSE 0 END AS UncoveredCost
  FROM focus_data_table CU
  CROSS JOIN
    UNNEST(JSON_EXTRACT_ARRAY(CU.CommitmentProgramEligibilityDetails, '$.CommitmentPrograms')) AS CP
  WHERE CU.ChargePeriodStart >= ? AND CU.ChargePeriodEnd < ?
    AND CU.ChargeCategory = 'Usage'
    AND CU.CommitmentProgramEligibilityDetails IS NOT NULL
    -- Replace with provider-specific discount-bearing program types
    AND JSON_VALUE(CP, '$.ProgramType') IN ('Flexible Spend Plan', 'Resource Reservation')
)
SELECT
  ServiceProviderName,
  ProgramType AS EligibleProgramType,
  SUM(EffectiveCost) AS EligibleCost,
  SUM(UncoveredCost) AS UncoveredEligibleCost,
  SUM(UncoveredCost) / NULLIF(SUM(EffectiveCost), 0) AS UncoveredRate
FROM CommitmentDiscountEligible
GROUP BY ServiceProviderName, ProgramType
ORDER BY UncoveredEligibleCost DESC
```

2.5.4.4. Identify Eligible Capacity Reservation Spend

Capacity reservations secure resource availability rather than provide discounts, and are tracked via CapacityReservationId and CapacityReservationStatus rather than the *commitment discount* columns used in the queries above. This query identifies *charges* eligible for capacity-reservation *commitment programs*, distinguishing between used and unused reservations.

The query filters CommitmentProgramEligibilityDetails to rows whose `ProgramType` values correspond to capacity-reservation programs (e.g., "Advance Resource Commitment", "Zonal Resource Commitment"). It then uses CapacityReservationId and CapacityReservationStatus to determine reservation utilization.

Note: The FOCUS specification requires CapacityReservationId to not be null when a charge represents unused capacity, but only recommends populating it when a charge is related to a used *capacity reservation*. Where a data generator does not populate CapacityReservationId on used rows, this query will show those rows with a null CapacityReservationStatus.

```
SELECT
  CU.ServiceProviderName,
  CU.ServiceName,
  JSON_VALUE(CP, '$.ProgramType') AS EligibleProgramType,
  CU.CapacityReservationStatus,
  SUM(CU.BilledCost) AS BilledCost,
  COUNT(*) AS RowCount
FROM focus_data_table CU
CROSS JOIN
  UNNEST(JSON_EXTRACT_ARRAY(CU.CommitmentProgramEligibilityDetails, '$.CommitmentPrograms')) AS CP
WHERE CU.ChargePeriodStart >= ? AND CU.ChargePeriodEnd < ?
  AND CU.ChargeCategory = 'Usage'
  AND CU.CommitmentProgramEligibilityDetails IS NOT NULL
  AND JSON_VALUE(CP, '$.ProgramType') IN ('Advance Resource Commitment', 'Zonal Resource Commitment')
GROUP BY
  CU.ServiceProviderName,
  CU.ServiceName,
  JSON_VALUE(CP, '$.ProgramType'),
  CU.CapacityReservationStatus
ORDER BY BilledCost DESC
```

2.5.5. Version Introduced

1.4

2.6. Contract Commitments

2.6.1. Description

FOCUS supports the tracking of commitments made via contractual agreements between a service provider and a customer. Each row in the Cost and Usage dataset is associated with one or more unique identifiers representing those contracts and contract commitments, stored in a JSON column called Contract Applied. A richer amount of detail that describes those commitments is carried in a separate Contract Commitment dataset, which can be joined to the Cost and Usage dataset to facilitate various queries involving filtering and aggregation.

The Contract Applied column contains several FOCUS-defined properties. For more information, see the definition of Contract Applied here.

2.6.2. Directly Dependent Columns

- CostAndUsage
  - ContractApplied

2.6.3. Supporting Columns

- ContractCommitment
  - BillingCurrency
  - ContractCommitmentCategory
  - ContractCommitmentCost
  - ContractCommitmentDescription
  - ContractCommitmentId
  - ContractCommitmentPeriodEnd
  - ContractCommitmentPeriodStart
  - ContractCommitmentQuantity
  - ContractCommitmentType
  - ContractCommitmentUnit
  - ContractId
  - ContractPeriodEnd
  - ContractPeriodStart

2.6.4. Example SQL Queries

The FOCUS specification implements the application of contract commitments to cost and usage via the *ContractApplied* column, which is defined in *JSON object format*.

Because ANSI SQL does not inherently support the parsing of JSON, the following queries leverage the JSON functions found in BigQuery Standard SQL in order to demonstrate this feature's functionality. Similar JSON functions are available in all major SQL engines; thus, the below examples can be slightly modified to accommodate any particular database instance.

2.6.4.1. Report on Initial Contract Commitment

This query takes inputs of a time range via ChargePeriodStart and ChargePeriodEnd, then presents the aggregation of initial contract commitments from the CostAndUsage dataset per ServiceProviderName and ContractCommitmentId by filtering on the specified time range, along with ChargeCategory of `Purchase`.

```
SELECT
  MIN(CU.ChargePeriodStart) AS ChargePeriodStart,
  MAX(CU.ChargePeriodEnd) AS ChargePeriodEnd,
  CU.ServiceProviderName,
  JSON_VALUE(CA, '$.ContractCommitmentId') AS ContractCommitmentId,
  SUM(CAST(JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') AS FLOAT64)) AS ContractCommitmentAppliedCost
FROM CostAndUsage CU
CROSS JOIN
  UNNEST(JSON_EXTRACT_ARRAY(CU.ContractApplied, '$.Elements')) AS CA
WHERE JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') IS NOT NULL
  AND ChargePeriodStart >= ? AND ChargePeriodEnd < ?
  AND ChargeCategory = 'Purchase'
GROUP BY ServiceProviderName, ContractCommitmentId
ORDER BY ServiceProviderName, ContractCommitmentId
```

2.6.4.2. Report on Usage Against Contract Commitment

This query takes inputs of a time range via ChargePeriodStart and ChargePeriodEnd, then presents the aggregation of the application of contract commitments from the CostAndUsage dataset per ServiceProviderName and ContractCommitmentId by filtering on the specified time range, along with ChargeCategory of `Usage`.

```
SELECT
  MIN(CU.ChargePeriodStart) AS ChargePeriodStart,
  MAX(CU.ChargePeriodEnd) AS ChargePeriodEnd,
  CU.ServiceProviderName,
  JSON_VALUE(CA, '$.ContractCommitmentId') AS ContractCommitmentId,
  SUM(CAST(JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') AS FLOAT64)) AS ContractCommitmentAppliedCost
FROM CostAndUsage CU
CROSS JOIN
  UNNEST(JSON_EXTRACT_ARRAY(CU.ContractApplied, '$.Elements')) AS CA
WHERE JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') IS NOT NULL
  AND ChargePeriodStart >= ? AND ChargePeriodEnd < ?
  AND ChargeCategory = 'Usage'
GROUP BY ServiceProviderName, ContractCommitmentId
ORDER BY ServiceProviderName, ContractCommitmentId
```

2.6.4.3. Report on Usage Against Contract Commitment by Category

This query takes inputs of a time range via ChargePeriodStart and ChargePeriodEnd, then presents the aggregation of the application of contract commitments from the CostAndUsage dataset per ServiceProviderName and ContractCommitmentId by filtering on the specified time range, along with ChargeCategory of `Usage`. It also joins in the ContractCommitment dataset to provide further information about each contract commitment (in this case, the start and end date/time).

```
SELECT
  MIN(CU.ChargePeriodStart) AS ChargePeriodStart,
  MAX(CU.ChargePeriodEnd) AS ChargePeriodEnd,
  CU.ServiceProviderName,
  JSON_VALUE(CA, '$.ContractCommitmentId') AS ContractCommitmentId,
  CC.ContractCommitmentPeriodStart,
  CC.ContractCommitmentPeriodEnd,
  SUM(CAST(JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') AS FLOAT64)) AS ContractCommitmentAppliedCost
FROM CostAndUsage CU
CROSS JOIN
  UNNEST(JSON_EXTRACT_ARRAY(CU.ContractApplied, '$.Elements')) AS CA
INNER JOIN
  ContractCommitment CC
ON
  JSON_VALUE(CA, '$.ContractCommitmentId') = CC.ContractCommitmentId
WHERE JSON_VALUE(CA, '$.ContractCommitmentAppliedCost') IS NOT NULL
  AND ChargePeriodStart >= ? AND ChargePeriodEnd < ?
  AND ChargeCategory = 'Usage'
```

```
GROUP BY ServiceProviderName, ContractCommitmentId, ContractCommitmentPeriodStart, ContractCommitmentPeriodEnd
ORDER BY ServiceProviderName, ContractCommitmentId, ContractCommitmentPeriodStart, ContractCommitmentPeriodEnd
```

2.6.5. Version Introduced

1.3

2.7. Cost and Usage Attribution

2.7.1. Description

Many service providers have features that allow FinOps practitioners to enrich cost and usage data with metadata that is in addition to service provider defined data, in order to analyze FinOps data using organizational, deployment, or other structures. These features may take the form of directly applied metadata or inherited metadata. FOCUS facilitates the inclusion of this metadata at a row level.

2.7.2. Directly Dependent Columns

- Tags

2.7.3. Supporting Columns

- BilledCost
- ConsumedQuantity
- ConsumedUnit
- EffectiveCost

2.7.4. Example SQL Query

```
SELECT
  tags,
  ConsumedUnit,
  SUM(BilledCost),
  SUM(EffectiveCost),
  SUM(ConsumedQuantity)
FROM focus_data_table
WHERE BillingPeriodStart >= ? AND BillingPeriodEnd < ?
GROUP BY
  tags,
  ConsumedUnit
```

2.7.5. Version Introduced

1.0

2.8. Cost Comparison

2.8.1. Description

FOCUS supports comparing cost columns to identify savings from *negotiated discounts*, the impact of *commitment discount* amortization, and differences between *cash-based* and *accrual-based* cost perspectives.

BilledCost represents the *cash-based* view: amounts invoiced by the InvoiceIssuerName in a given *billing period*. EffectiveCost represents the *accrual-based* view: costs recognized when *resources* are consumed, *services* are used, or *contract commitments* are recognized. ListCost provides the pre-discount baseline. ContractedCost reflects pricing after *negotiated discounts*.

Comparing ListCost against ContractedCost quantifies *negotiated discount* savings; comparing ContractedCost against EffectiveCost isolates *commitment discount* savings; comparing EffectiveCost against ListCost shows total combined savings. BilledCost and EffectiveCost diverge when billing timing differs from consumption, such as with *commitment discounts* or prepaid purchases.

When comparing costs across marketplace boundaries, practitioners should be aware that BilledCost is zero for *charges* generated by entities not responsible for invoicing, so BilledCost and EffectiveCost sums may not align within a single dataset when *covering charges* and *covered charges* span multiple *invoice issuers*. This is expected and does not indicate a data quality issue. See the Marketplace Purchases supported feature, Examples: Commitment Discount Flexibility, and Examples: SaaS for additional context.

2.8.2. Directly Dependent Columns

- BilledCost
- ContractedCost
- EffectiveCost
- ListCost

2.8.3. Supporting Columns

- BillingAccountId
- BillingAccountName
- BillingCurrency
- BillingPeriodEnd
- BillingPeriodStart
- ChargeCategory
- ChargePeriodEnd
- ChargePeriodStart
- CommitmentDiscountId
- InvoiceIssuerName
- ServiceName
- ServiceProviderName

2.8.4. Example SQL Queries

2.8.4.1. Discount Effectiveness by Service

```
WITH AggregatedData AS (
  SELECT
    ServiceProviderName,
    BillingAccountId,
    BillingAccountName,
    BillingCurrency,
    ServiceName,
    SUM(EffectiveCost) AS TotalEffectiveCost,
    SUM(BilledCost) AS TotalBilledCost,
    SUM(CASE
        WHEN ChargeCategory = 'Usage' AND BilledCost = 0 AND EffectiveCost != 0
        THEN 0
        ELSE ContractedCost
      END) AS TotalContractedCost,
```

```
    SUM(CASE
        WHEN ChargeCategory = 'Usage' AND BilledCost = 0 AND EffectiveCost != 0
        THEN 0
        ELSE ListCost
       END) AS TotalListCost
  FROM focus_data_table
  WHERE BillingPeriodStart >= ?
    AND BillingPeriodEnd < ?
    AND ChargeClass IS NULL
  GROUP BY
    ServiceProviderName,
    BillingAccountId,
    BillingAccountName,
    BillingCurrency,
    ServiceName
)
SELECT ServiceProviderName,
    BillingAccountId,
    BillingAccountName,
    BillingCurrency,
    ServiceName,
    TotalEffectiveCost,
    TotalBilledCost,
    TotalListCost,
    (1 - TotalContractedCost / NULLIF(TotalListCost, 0)) * 100 AS ContractedDiscount,
    (1 - TotalEffectiveCost / NULLIF(TotalListCost, 0)) * 100 AS EffectiveDiscount
FROM AggregatedData
```

2.8.4.2. Cash vs. Accrual Comparison by Billing Period

```
SELECT
  ServiceProviderName,
  InvoiceIssuerName,
  BillingPeriodStart,
  BillingPeriodEnd,
  SUM(BilledCost) AS TotalBilledCost,
  SUM(EffectiveCost) AS TotalEffectiveCost,
  SUM(EffectiveCost) - SUM(BilledCost) AS CostBasisDifference
FROM focus_data_table
WHERE BillingPeriodStart >= ? AND BillingPeriodEnd < ?
GROUP BY
  ServiceProviderName,
  InvoiceIssuerName,
  BillingPeriodStart,
  BillingPeriodEnd
HAVING ABS(SUM(EffectiveCost) - SUM(BilledCost)) > 0.01
```

2.8.5. Version Introduced

0.5

2.8.6. Version Updated

1.4

2.9. Custom Columns

2.9.1. Description

FOCUS supports the inclusion of custom columns to facilitate reporting capability that is not covered by the columns included in the specification. See Dataset Completeness for requirements on when custom columns should be included.

2.9.2. Directly Dependent Columns

- x_CustomColumn

2.9.3. Example SQL Query

```
SELECT
  BillingPeriodStart,
  x_CustomColumn,
  SUM(BilledCost) AS TotalBilledCost,
FROM focus_data_table
WHERE ServiceName = ?
  AND BillingPeriodStart >= ? AND BillingPeriodStart < ?
GROUP BY
  BillingPeriodStart,
  x_CustomColumn
ORDER BY MonthlyCost DESC
```

2.9.4. Version Introduced

0.5

2.10. Data Generator-Calculated Split Cost Allocation

2.10.1. Description

FOCUS enables tracking of resources split by some internal consumption metrics. This is most common for resources supporting shared usage like compute nodes in a shared cluster (Kubernetes, databases) or storage engines that can share capacity between workloads.

2.10.2. Directly Dependent Columns

- ResourceId
- EffectiveCost
- BilledCost
- AllocatedResourceId
- AllocatedResourceName
- AllocatedMethodDetails
- AllocatedMethodId

2.10.3. Supporting Columns

- ChargeCategory
- ChargePeriodEnd
- ChargePeriodStart
- ServiceProviderName
- ServiceName

2.10.4. Example SQL Query (Find Resources with a Shared Cost)

```
SELECT
  DISTINCT ResourceId
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
  AND AllocatedMethodId IS NOT NULL
```

2.10.5. Example SQL Query (Get Total Effective Cost by ResourceId (Ignore Shared Cost))

```
SELECT
  ResourceId,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
  AND AllocatedMethodId IS NOT NULL
GROUP BY
  ResourceId
```

2.10.6. Example SQL Query (Get Total Effective Cost by AllocatedResourceId)

```
SELECT
  AllocatedResourceId,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
  AND AllocatedMethodId IS NOT NULL
GROUP BY
  AllocatedResourceId
```

2.10.7. Example SQL Query (Find Total Unallocated Split Costs by ResourceId)

```
SELECT
  ResourceId,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
  AND AllocatedMethodId IS NOT NULL AND AllocatedResourceId IS NULL
GROUP BY
  ResourceId
```

2.10.8. Example SQL Query (Find How a Single Resource Has Been Split)

```
SELECT
  ResourceId,
  COALESCE(AllocatedResourceId, 'Unallocated') AS AllocatedResourceId,
  SUM(EffectiveCost) AS TotalEffectiveCost
```

```
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
  AND AllocatedResourceId = ?
GROUP BY
  ResourceId,
  COALESCE(AllocatedResourceId, 'Unallocated')
```

2.10.9. Example SQL Query (Extract JSON from AllocatedMethodDetails)

```
SELECT
  resource_id,
  elements.allocated_ratio,
  elements.usage_unit,
  elements.usage_quantity
FROM
  focus_data_table,
  JSON_TABLE(
    AllocatedMethodDetails,
    '$.Elements[*]' COLUMNS (
      allocated_ratio DECIMAL(10, 2) PATH '$.AllocatedRatio',
      usage_unit VARCHAR(50) PATH '$.UsageUnit',
      usage_quantity DECIMAL(10, 2) PATH '$.UsageQuantity'
    )
  ) AS elements
```

2.10.10. Version Introduced

1.3

2.11. Data Granularity

2.11.1. Description

FOCUS supports multiple levels of cost and usage data granularity. This includes the ability to report on a daily, hourly, or other time period basis. FOCUS also supports the ability for cost and usage data to be provided for high granularity scenarios, such as down to the individual resources. It also supports high level granularity cost and usage data, such as account level, or service level charges.

2.11.2. Directly Dependent Columns

- ResourceId
- ResourceName
- ChargePeriodEnd
- ChargePeriodStart

2.11.3. Supporting Columns

- BilledCost
- ConsumedQuantity
- ConsumedUnit
- EffectiveCost
- ListCost
- PricingCurrency

- PricingUnit

2.11.4. Example SQL Query

```
SELECT
  ChargePeriodStart,
  ChargePeriodEnd,
  ResourceId,
  SUM(EffectiveCost)
FROM focus_data_table
Group by
  ChargePeriodStart,
  ChargePeriodEnd,
  ResourceId
```

2.11.5. Version Introduced

0.5

2.12. Dataset Instance Metadata

2.12.1. Description

FOCUS supports the ability for data generators to provide metadata that describes information about the *dataset artifacts* they provide. This includes properties such as the name of the *dataset instance*, the unique identifier of the *dataset instance*, and the *FOCUS dataset* that it aligns with. This metadata can be used by consumers to understand the context of the data they are receiving, and to ensure that they are working with the correct dataset instance to execute their particular FinOps use cases.

2.12.2. Applicable Metadata

- Dataset Instance
  - Dataset Instance ID
  - Dataset Instance Name
  - FOCUS Dataset ID

2.12.3. Version Introduced

1.3

2.13. Effective Cost Analysis

2.13.1. Description

FOCUS enables practitioners to analyze costs on an *accrual basis*, where expenses are recognized when *resources* are consumed, *services* are used, or *contract commitments* are recognized, regardless of when those costs are invoiced. The EffectiveCost column reflects all applicable pricing adjustments and distributes the cost of *covering charges* (one-time or recurring purchases) to the *covered charges* they offset.

For *charges* that are not *covered charges*, EffectiveCost reflects the same amount as BilledCost. For *covering charges* (e.g., *commitment discount* purchases, prepayments, or marketplace purchases), EffectiveCost is 0 because their cost is recognized on the *covered charges* instead. In *commitment discount* scenarios, CommitmentDiscountStatus distinguishes whether amortized cost was allocated to consumed *resources* or *services* ("Used") or remained unallocated ("Unused").

EffectiveCost is commonly used for *accrual-based* reporting, cost allocation, chargeback, and spending trend analysis. For scenarios involving *commitment discounts* across different payment models, see Examples: Commitment Discount Flexibility. For marketplace and SaaS scenarios, see Examples: SaaS.

2.13.2. Directly Dependent Columns

- EffectiveCost

2.13.3. Supporting Columns

- BillingPeriodEnd
- BillingPeriodStart
- ChargeCategory
- ChargePeriodEnd
- ChargePeriodStart
- CommitmentDiscountId
- CommitmentDiscountStatus
- ConsumedQuantity
- ConsumedUnit
- PricingQuantity
- RegionName
- ServiceName
- ServiceProviderName

2.13.4. Example SQL Queries

EffectiveCost can be analyzed using *billing period* or *charge period* time filters. Billing period aligns with invoice cycles and is useful for financial reporting. Charge period captures when resources were consumed or commitments were recognized, which is more precise for consumption-based analysis.

2.13.4.1. Effective Cost by Service and Region

Aggregates EffectiveCost by billing period to align accrual-based cost reporting with invoice cycles.

```
SELECT
  ServiceProviderName,
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceCategory,
  ServiceName,
  RegionId,
  RegionName,
  PricingUnit,
  SUM(EffectiveCost) AS TotalEffectiveCost,
  SUM(PricingQuantity) AS TotalPricingQuantity
FROM focus_data_table
WHERE BillingPeriodStart >= ? AND BillingPeriodEnd < ?
GROUP BY
  ServiceProviderName,
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceCategory,
  ServiceName,
  RegionId,
  RegionName,
  PricingUnit
```

2.13.4.2. Commitment Discount Effective Cost Breakdown

Analyzes commitment discount amortization by charge period, capturing costs based on when resources were consumed.

```
SELECT
  ServiceProviderName,
  CommitmentDiscountId,
  CommitmentDiscountStatus,
  ChargePeriodStart,
  ChargePeriodEnd,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ChargePeriodStart >= ? AND ChargePeriodEnd < ?
  AND CommitmentDiscountId IS NOT NULL
  AND ChargeCategory = 'Usage'
GROUP BY
  ServiceProviderName,
  CommitmentDiscountId,
  CommitmentDiscountStatus,
  ChargePeriodStart,
  ChargePeriodEnd
```

2.13.5. Version Introduced

0.5

2.13.6. Version Updated

1.4

2.14. Invoice Reconciliation

2.14.1. Description

FOCUS supports the reconciliation of granular cloud consumption records with the formal financial documents issued by an *invoice issuer*. The Invoice Detail dataset represents the definitive financial record of *charges* as they appear on an invoice. By leveraging common identifiers, practitioners can map usage-based costs in the Cost and Usage dataset back to their corresponding line items in the Invoice Detail dataset.

This feature also supports reconciliation across divergent currency grains. When an invoice issuer represents billing and payment currencies at different aggregation levels, the `PaymentCurrencyInvoiceDetailId` provides the necessary lineage to link granular usage records to the aggregate records used for financial settlement.

2.14.2. Directly Dependent Columns

- InvoiceDetail
  - BilledCost
  - InvoiceId
  - InvoiceDetailId
  - PaymentCurrencyBilledCost
  - PaymentCurrencyInvoiceDetailId
- CostAndUsage
  - BilledCost
  - InvoiceId
  - InvoiceDetailId

2.14.3. Supporting Columns

- InvoiceDetail
  - BillingCurrency
  - ChargeCategory
  - InvoiceIssueStatus
  - PaymentCurrency
- CostAndUsage
  - ChargeCategory
  - ServiceCategory

2.14.4. Example SQL Queries

Reconciliation often requires aggregating granular usage data to match the coarser grain of an invoice. The following queries demonstrate how to validate that usage records equal the billed amounts on a legal invoice.

2.14.4.1. Reconcile Cost and Usage to Invoice Detail by Invoice ID

This query validates that the sum of costs for all service usage in the CostAndUsage dataset equals the total non-tax charges in the InvoiceDetail dataset for a specific invoice.

```
SELECT
  COALESCE(ID.InvoiceId, CU.InvoiceId) AS InvoiceId,
  ID.TotalBilledCost_InvoiceDetail,
  CU.TotalBilledCost_CostAndUsage,
  (COALESCE(ID.TotalBilledCost_InvoiceDetail, 0) - COALESCE(CU.TotalBilledCost_CostAndUsage, 0)) AS Variance
FROM (
  SELECT
    InvoiceId,
    SUM(BilledCost) AS TotalBilledCost_InvoiceDetail
  FROM InvoiceDetail
  WHERE ChargeCategory != 'Tax'
  GROUP BY InvoiceId
) ID
FULL OUTER JOIN (
  SELECT
    InvoiceId,
    SUM(BilledCost) AS TotalBilledCost_CostAndUsage
  FROM CostAndUsage
  WHERE ChargeCategory != 'Tax'
  GROUP BY InvoiceId
) CU ON ID.InvoiceId = CU.InvoiceId
WHERE COALESCE(ID.InvoiceId, CU.InvoiceId) = ?
```

2.14.4.2. Reconcile Multi-Currency Settlement Using Lineage IDs

This query demonstrates how to use the `PaymentCurrencyInvoiceDetailId` to reconcile granular records (denominated in the billing currency) against the aggregate records used for payment (denominated in the payment currency). This resolves the "Divergent Grain" problem.

```
SELECT
  PaymentCurrencyInvoiceDetailId,
  SUM(BilledCost) AS TotalBilled_BillingCurrency,
  SUM(PaymentCurrencyBilledCost) AS TotalBilled_PaymentCurrency,
  -- Calculate effective exchange rate for the group
  SUM(PaymentCurrencyBilledCost) / NULLIF(SUM(BilledCost), 0) AS EffectiveExchangeRate
FROM InvoiceDetail
WHERE InvoiceId = ?
GROUP BY PaymentCurrencyInvoiceDetailId
ORDER BY PaymentCurrencyInvoiceDetailId
```

2.14.4.3. Validate Tax Variance

This query identifies the tax component present in the InvoiceDetail dataset that is typically excluded from the CostAndUsage dataset, allowing for a complete three-way match between usage, tax, and the total invoice amount.

```
SELECT
  InvoiceId,
  SUM(CASE WHEN ChargeCategory = 'Tax' THEN BilledCost ELSE 0 END) AS TotalTaxAmount,
  SUM(CASE WHEN ChargeCategory != 'Tax' THEN BilledCost ELSE 0 END) AS TotalServiceAmount,
  SUM(BilledCost) AS GrandTotalPayable
FROM InvoiceDetail
WHERE InvoiceId = ?
GROUP BY InvoiceId
```

2.14.5. Version Introduced

1.4

2.15. Location

2.15.1. Description

FOCUS provides structured location data through region and availability zone information. By documenting geographic deployment locations, practitioners can organize and analyze costs based on where resources and services are deployed. This standardized location data helps practitioners understand the geographical distribution of infrastructure across host providers.

2.15.2. Directly Dependent Columns

- AvailabilityZone
- RegionId
- RegionName

2.15.3. Supporting Columns

- BilledCost
- ChargePeriodEnd
- ChargePeriodStart

2.15.4. Example SQL Query

```
SELECT
  RegionId,
  RegionName,
  AvailabilityZone,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
GROUP BY
  RegionId,
  RegionName,
  AvailabilityZone
```

2.15.5. Version Introduced

1.0

2.16. Marketplace Purchases

2.16.1. Description

FOCUS supports the analysis of cost and usage data for marketplace purchases and their associated costs. It also supports the reporting of EffectiveCost for usage from the service provider.

2.16.2. Directly Dependent Columns

- InvoiceIssuerName
- ServiceProviderName

2.16.3. Supporting Columns

- BilledCost
- EffectiveCost

2.16.4. Example SQL Query on a CSP Marketplace Using the Cost and Usage FOCUS Dataset

```
SELECT
  ServiceProviderName,
  InvoiceIssuerName,
  BillingPeriodStart,
  BillingPeriodEnd,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE ServiceProviderName = '<Example SaaS Provider>'
  AND InvoiceIssuerName = '<Example CSP Marketplace>'
GROUP BY
  ServiceProviderName,
  InvoiceIssuerName,
  BillingPeriodStart,
  BillingPeriodEnd
```

2.16.5. Example SQL Query on a Provider Using the Cost and Usage FOCUS Dataset

```
SELECT
  ChargePeriodStart,
  ChargePeriodEnd,
  ResourceId,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE InvoiceIssuerName = '<Example CSP Marketplace>'
GROUP BY
  ChargePeriodStart,
  ChargePeriodEnd,
  ResourceId
```

2.16.6. Version Introduced

1.0

2.17. Participating Entity Identification

2.17.1. Description

FOCUS allows practitioners to identify the several participating entities involved in resource or service hosting, invoicing, and data generation. The FOCUS Specification includes multiple columns to identify key participating entities, including Service Provider Name, Invoice Issuer Name, Host Provider Name, and Data Generator.

2.17.2. Directly Dependent Columns

- ServiceProviderName
- InvoiceIssuerName
- HostProviderName

2.17.3. Applicable Metadata

- DataGenerator

2.17.4. Example SQL Query

```
SELECT
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceProviderName,
  InvoiceIssuerName,
  HostProviderName,
  ServiceName,
  BillingCurrency,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE BillingPeriodStart >= ? and BillingPeriodEnd < ?
GROUP BY
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceProviderName,
  InvoiceIssuerName,
  HostProviderName,
  ServiceName,
  BillingCurrency
```

2.17.5. Version Introduced

1.1

2.17.6. Version Updated

1.3

2.18. Recency Metadata

2.18.1. Description

FOCUS supports the ability for data generators to provide metadata indicating 1) what portion of a FOCUS dataset artifact is complete (either in total, or per time sector), and 2) how recently it has been updated. This metadata allows practitioners to understand whether a given subset of FOCUS data is subject to further change, which informs when and whether they can perform various FinOps functions such as chargeback.

2.18.2. Applicable Metadata

- Recency
  - Dataset Instance Complete
  - Dataset Instance Last Updated
  - Dataset Instance ID
  - Recency Last Updated
  - Time Sectors
    - Time Sector Start
    - Time Sector End
    - Time Sector Complete
    - Time Sector Last Updated

2.18.3. Version Introduced

1.3

2.19. Resource Usage

2.19.1. Description

FOCUS enables tracking of resource consumption by providing information about which resources were used, in what quantities, and with what units of measure.

2.19.2. Directly Dependent Columns

- ConsumedQuantity
- ConsumedUnit
- ResourceId
- SkuId

2.19.3. Supporting Columns

- ChargeCategory
- ChargePeriodEnd
- ChargePeriodStart
- ServiceProviderName
- ServiceName

2.19.4. Example SQL Query

```
SELECT
  ServiceProviderName,
  ServiceName,
  ResourceId,
  SkuId,
  ConsumedUnit,
  SUM(ConsumedQuantity) AS TotalQuantity
FROM focus_data_table
WHERE ChargeCategory='Usage'
  AND ChargePeriodStart >= ? AND ChargePeriodEnd <= ?
GROUP BY
  ServiceProviderName,
  ServiceName,
  ResourceId,
  SkuId,
  ConsumedUnit
```

2.19.5. Version Introduced

1.0

2.20. Schema Metadata

2.20.1. Description

FOCUS' schema metadata supports communication of important attributes about the data, facilitating notifications about changing structure and database table creation between data generator and consumer. This includes column names, data types, and any other relevant information about the data schema. It also includes information as to the version of FOCUS and Data Generator versioning that the data uses.

2.20.2. Applicable Metadata

- Schema
    - Column Definition

2.20.3. Version Introduced

1.1

2.21. Service Categorization

2.21.1. Description

FOCUS provides a structure for categorizing services based on their core functions. By classifying services into high-level categories and more granular subcategories, practitioners can organize costs according to functional areas. This standardized categorization provides data that practitioners can use in their cost management processes and decision making.

2.21.2. Directly Dependent Columns

- ServiceCategory
- ServiceName
- ServiceSubcategory

2.21.3. Supporting Columns

- BilledCost
- BillingCurrency
- BillingPeriodEnd
- BillingPeriodStart
- ServiceProviderName

2.21.4. Example SQL Query

```
SELECT
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceProviderName,
  ServiceCategory,
  ServiceSubcategory,
  ServiceName,
  BillingCurrency,
  SUM(BilledCost) AS TotalBilledCost
FROM focus_data_table
WHERE BillingPeriodStart >= ? and BillingPeriodEnd < ?
GROUP BY
  BillingPeriodStart,
  BillingPeriodEnd,
  ServiceProviderName,
  ServiceCategory,
  ServiceSubcategory,
  ServiceName,
  BillingCurrency
```

2.21.5. Version Introduced

1.1

2.22. Service Provider Services

2.22.1. Description

FOCUS supports service providers specifying the services and product offerings that they provide their customers that align with the names practitioners are familiar with. This empowers practitioners to analyze cost by service, report service costs by subaccount, forecast based on historical trends by service, and verify accuracy of services charged across service providers.

2.22.2. Directly Dependent Columns

- ServiceCategory
- ServiceName
- ServiceSubcategory

2.22.3. Supporting Columns

- ServiceProviderName
- SkuId

2.22.4. Example SQL Query

```
SELECT
  BillingPeriodStart,
  ServiceProviderName,
  SubAccountId,
  SubAccountName,
  ServiceName,
  SUM(BilledCost) AS TotalBilledCost,
  SUM(EffectiveCost) AS TotalEffectiveCost
FROM focus_data_table
WHERE ServiceName = ?
  AND BillingPeriodStart >= ? AND BillingPeriodStart < ?
GROUP BY
  BillingPeriodStart,
  ServiceProviderName,
  SubAccountId,
  SubAccountName,
  ServiceName
ORDER BY MonthlyCost DESC
```

2.22.5. Version Introduced

0.5

2.23. Verification, Comparison, and Fluctuation Tracking of Unit Prices

2.23.1. Description

When a service provider supports unit pricing concepts, FOCUS allows practitioners to:

- Verify that the correct List Unit Prices and Contracted Unit Prices are applied.
- Compare applied Contracted Unit Prices across different billing accounts and with applied List Unit Prices at specific points in time.
- Track fluctuations in unit prices over time.

2.23.2. Directly Dependent Columns

- ContractedUnitPrice
- ListUnitPrice
- SkuId
- SkuPriceDetails
- SkuPriceId

2.23.3. Supporting Columns

- BillingCurrency
- BillingPeriodId
- ChargePeriodEnd
- ChargePeriodStart

2.23.4. Example SQL Query

```
SELECT DISTINCT
  SkuId,
  SkuPriceId,
  SkuPriceDetails,
  BillingPeriodId,
  ChargePeriodStart,
  ChargePeriodEnd,
  BillingCurrency,
  ListUnitPrice,
  ContractedUnitPrice
FROM focus_data_table
WHERE
  SkuPriceId = ?
  AND ChargePeriodStart >= ?
  AND ChargePeriodEnd < ?
```

2.23.5. Version Introduced

1.0

3. Datasets

FOCUS defines many individual datasets made up of a selected set of columns which abide by the attributes outlined in this FOCUS Specification.

Dataset List

Datasets are sorted first by Feature Level (i.e., Mandatory, then Conditional), then alphabetically by name.

Dataset	Dataset Type	Feature Level	Description
Cost and Usage	Transaction	Mandatory	Describes the cost and usage incurred through using or purchasing a service provider's resources or services.
Billing Period	Reference	Conditional	Describes the billing periods by which cost and usage is invoiced.
Contract Commitment	Reference	Conditional	Describes the terms of contracts agreed between a service provider and a customer.
Invoice Detail	Transaction	Conditional	Describes the cost and usage issued on invoices.

3.1. Cost and Usage

The Cost and Usage dataset is the primary dataset for FOCUS cost and usage data.

The specification for the Cost and Usage dataset defines a group of columns that provide qualitative values (such as dates, resource, and service provider information) categorized as "dimensions" and quantitative values (numeric values) categorized as "metrics" that can be used for performing various FinOps capabilities. Metrics are commonly used for aggregations (sum, multiplication, averaging etc.) and statistical operations within the dataset. Dimensions are commonly used to categorize, filter, and reveal details in your data when combined with metrics. The columns are presented in alphabetical order.

Columns

Column	Column Type	Feature Level	Allows Nulls	Data Type
Allocated Method Details	Dimension / Metric	Recommended	True	JSON
Allocated Method ID	Dimension	Conditional	True	String
Allocated Resource ID	Dimension	Conditional	True	String
Allocated Resource Name	Dimension	Conditional	True	String
Allocated Tags	Dimension	Conditional	True	JSON
Availability Zone	Dimension	Recommended	True	String
Billed Cost	Metric	Mandatory	False	Decimal
Billing Account ID	Dimension	Mandatory	False	String
Billing Account Name	Dimension	Mandatory	True	String
Billing Account Type	Dimension	Conditional	False	String
Billing Currency	Dimension	Mandatory	False	String
Billing Period End	Dimension	Mandatory	False	Date/Time
Billing Period Start	Dimension	Mandatory	False	Date/Time
Capacity Reservation ID	Dimension	Conditional	True	String
Capacity Reservation Status	Dimension	Conditional	True	String
Charge Category	Dimension	Mandatory	False	String
Charge Class	Dimension	Mandatory	True	String
Charge Description	Dimension	Mandatory	True	String
Charge Frequency	Dimension	Recommended	False	String
Charge Period End	Dimension	Mandatory	False	Date/Time
Charge Period Start	Dimension	Mandatory	False	Date/Time
Commitment Discount Category	Dimension	Conditional	True	String

Column	Column Type	Feature Level	Allows Nulls	Data Type
Commitment Discount ID	Dimension	Conditional	True	String
Commitment Discount Name	Dimension	Conditional	True	String
Commitment Discount Quantity	Metric	Conditional	True	Decimal
Commitment Discount Status	Dimension	Conditional	True	String
Commitment Discount Type	Dimension	Conditional	True	String
Commitment Discount Unit	Dimension	Conditional	True	String
Commitment Program Eligibility Details	Dimension	Conditional	True	JSON
Consumed Quantity	Metric	Conditional	True	Decimal
Consumed Unit	Dimension	Conditional	True	String
Contract Applied	Dimension / Metric	Conditional	True	JSON
Contracted Cost	Metric	Mandatory	False	Decimal
Contracted Unit Price	Metric	Conditional	True	Decimal
Effective Cost	Metric	Mandatory	False	Decimal
Host Provider Name	Dimension	Mandatory	True	String
Invoice Detail ID	Dimension	Conditional	True	String
Invoice ID	Dimension	Conditional	True	String
Invoice Issuer Name	Dimension	Mandatory	False	String
List Cost	Metric	Mandatory	False	Decimal
List Unit Price	Metric	Conditional	True	Decimal
Pricing Category	Dimension	Conditional	True	String
Pricing Currency	Dimension	Conditional	False	String
Pricing Currency Contracted Unit Price	Metric	Conditional	True	Decimal
Pricing Currency Effective Cost	Metric	Conditional	False	Decimal
Pricing Currency List Unit Price	Metric	Conditional	True	Decimal
Pricing Quantity	Metric	Mandatory	True	Decimal
Pricing Unit	Dimension	Mandatory	True	String
Region ID	Dimension	Conditional	True	String
Region Name	Dimension	Conditional	True	String
Resource ID	Dimension	Conditional	True	String
Resource Name	Dimension	Conditional	True	String
Resource Type	Dimension	Conditional	True	String
Service Category	Dimension	Mandatory	False	String
Service Name	Dimension	Mandatory	False	String
Service Provider Name	Dimension	Mandatory	False	String
Service Subcategory	Dimension	Recommended	False	String
SKU ID	Dimension	Conditional	True	String
SKU Meter	Dimension	Conditional	True	String
SKU Price Details	Dimension	Conditional	True	JSON
SKU Price ID	Dimension	Conditional	True	String
Sub Account ID	Dimension	Conditional	True	String
Sub Account Name	Dimension	Conditional	True	String
Sub Account Type	Dimension	Conditional	True	String
Tags	Dimension	Conditional	True	JSON

Relationships

The Cost and Usage dataset can be joined to the Contract Commitment dataset through the use of the Contract Commitment ID.

- In the Cost and Usage dataset, Contract Commitment ID is a property within a JSON object array provided in Contract Applied column.
- In the Contract Commitment dataset, Contract Commitment ID is a column.

Dataset A	Dataset A Column	Dataset B	Dataset B Column

Dataset A	Dataset A Column	Dataset B	Dataset B Column
Cost and Usage	Contract Applied	Contract Commitment	Contract Commitment ID

Requirements

CostAndUsage MUST adhere to the following requirements:

- CostAndUsage MUST be present.
- CostAndUsage column presence MUST adhere to the following requirements:
  - CostAndUsage SHOULD include AllocatedMethodDetails when the data generator supports data generator-calculated split cost allocation.
  - CostAndUsage MUST include AllocatedMethodId when the data generator supports data generator-calculated split cost allocation.
  - CostAndUsage MUST include AllocatedResourceId when the data generator supports data generator-calculated split cost allocation.
  - CostAndUsage MUST include AllocatedResourceName when the data generator supports data generator-calculated split cost allocation.
  - CostAndUsage MUST include AllocatedTags when the data generator supports data generator-calculated split cost allocation.
  - CostAndUsage SHOULD include AvailabilityZone when the host provider supports deploying resources or services within an *availability zone*.
  - CostAndUsage MUST include BilledCost.
  - CostAndUsage MUST include BillingAccountId.
  - CostAndUsage MUST include BillingAccountName.
  - CostAndUsage MUST include BillingAccountType when the *invoice issuer* supports more than one possible BillingAccountType value.
  - CostAndUsage MUST include BillingCurrency.
  - CostAndUsage MUST include BillingPeriodEnd.
  - CostAndUsage MUST include BillingPeriodStart.
  - CostAndUsage MUST include CapacityReservationId when the service provider supports *capacity reservations*.
  - CostAndUsage MUST include CapacityReservationStatus when the service provider supports *capacity reservations*.
  - CostAndUsage MUST include ChargeCategory.
  - CostAndUsage MUST include ChargeClass.
  - CostAndUsage MUST include ChargeDescription.
  - CostAndUsage SHOULD include ChargeFrequency.
  - CostAndUsage MUST include ChargePeriodEnd.
  - CostAndUsage MUST include ChargePeriodStart.
  - CostAndUsage MUST include CommitmentDiscountCategory when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountId when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountName when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountQuantity when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountStatus when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountType when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentDiscountUnit when the service provider supports *commitment discounts*.
  - CostAndUsage MUST include CommitmentProgramEligibilityDetails when the service provider supports at least one *commitment program*.
  - CostAndUsage MUST include ConsumedQuantity when the service provider supports the measurement of usage.
  - CostAndUsage MUST include ConsumedUnit when the service provider supports the measurement of usage.
  - CostAndUsage MUST include ContractApplied when the service provider supports *contract commitments*.
  - CostAndUsage MUST include ContractedCost.
  - CostAndUsage MUST include ContractedUnitPrice when the service provider supports negotiated pricing concepts.
  - CostAndUsage MUST include EffectiveCost.
  - CostAndUsage MUST include HostProviderName.
  - CostAndUsage MUST include InvoiceDetailId when the invoice issuer supports payable invoices.
  - CostAndUsage MUST include InvoiceId when the invoice issuer supports payable invoices.
  - CostAndUsage MUST include InvoiceIssuerName.
  - CostAndUsage MUST include ListCost.
  - CostAndUsage MUST include ListUnitPrice when the service provider publishes unit prices exclusive of discounts.
  - CostAndUsage MUST include PricingCategory when the service provider supports more than one pricing category across all *SKUs*.

    - CostAndUsage MUST include PricingCurrency when the service provider supports pricing and billing in different currencies.
    - CostAndUsage MUST adhere to the following PricingCurrencyContractedUnitPrice presence requirements:
        - CostAndUsage MUST include PricingCurrencyContractedUnitPrice when the service provider supports prices in virtual currency and publishes unit prices exclusive of discounts.
        - CostAndUsage SHOULD include PricingCurrencyContractedUnitPrice when the service provider supports pricing and billing in different currencies and publishes unit prices exclusive of discounts.
        - CostAndUsage MAY include PricingCurrencyContractedUnitPrice in all other cases.
    - CostAndUsage MUST adhere to the following PricingCurrencyEffectiveCost presence requirements:
        - CostAndUsage MUST include PricingCurrencyEffectiveCost when the service provider supports prices in virtual currency and publishes unit prices exclusive of discounts.
        - CostAndUsage SHOULD include PricingCurrencyEffectiveCost when the service provider supports pricing and billing in different currencies and publishes unit prices exclusive of discounts.
        - CostAndUsage MAY include PricingCurrencyEffectiveCost in all other cases.
    - CostAndUsage MUST adhere to the following PricingCurrencyListUnitPrice presence requirements:
        - CostAndUsage MUST include PricingCurrencyListUnitPrice when the service provider supports prices in virtual currency and publishes unit prices exclusive of discounts.
        - CostAndUsage SHOULD include PricingCurrencyListUnitPrice when the service provider supports pricing and billing in different currencies and publishes unit prices exclusive of discounts.
        - CostAndUsage MAY include PricingCurrencyListUnitPrice in all other cases.
    - CostAndUsage MUST include PricingQuantity.
    - CostAndUsage MUST include PricingUnit.
    - CostAndUsage MUST include RegionId when the host provider supports deploying resources or services within a region.
    - CostAndUsage MUST include RegionName when the host provider supports deploying resources or services within a region.
    - CostAndUsage MUST include ResourceId when the service provider supports billing based on provisioned *resources*.
    - CostAndUsage MUST include ResourceName when the service provider supports billing based on provisioned resources.
    - CostAndUsage MUST include ResourceType when the service provider supports billing based on provisioned *resources* and supports assigning types to *resources*.
    - CostAndUsage MUST include ServiceCategory.
    - CostAndUsage MUST include ServiceName.
    - CostAndUsage MUST include ServiceProviderName.
    - CostAndUsage SHOULD include ServiceSubcategory.
    - CostAndUsage MUST include SkuId when the service provider supports unit pricing concepts and publishes price lists, publicly or as part of contracting.
    - CostAndUsage MUST include SkuMeter when the service provider supports unit pricing concepts and publishes *price lists*, publicly or as part of contracting.
    - CostAndUsage MUST include SkuPriceDetails when the service provider supports unit pricing concepts and publishes *price lists*, publicly or as part of contracting.
    - CostAndUsage MUST include SkuPriceId when the service provider supports unit pricing concepts and publishes *price lists*, publicly or as part of contracting.
    - CostAndUsage MUST include SubAccountId when the service provider supports a *sub account* construct.
    - CostAndUsage MUST include SubAccountName when the service provider supports a *sub account* construct.
    - CostAndUsage MUST include SubAccountType when the service provider supports more than one possible SubAccountType value.
    - CostAndUsage MUST include Tags when the data generator supports setting user or provider-defined tags.
    - CostAndUsage SHOULD include *custom columns* needed to identify all applied discounts when *FOCUS columns* are not sufficient.
- CostAndUsage MUST conform to CorrectionHandling requirements.
- CostAndUsage MUST conform to DatasetCompleteness requirements.
- CostAndUsage MUST conform to DatasetConfiguration requirements.
- CostAndUsage MUST conform to DeliveryHandling requirements.
- CostAndUsage MUST include *charges* representing unused portions of a *commitment* when the *commitment* is not fully utilized.
- CostAndUsage MUST include separate *charges* representing discounted and non-discounted portions when a discount applies to only a portion of the originally incurred *charge*.
- When the data generator supports data generator-calculated split cost allocation, CostAndUsage MUST adhere to the following requirements:
    - CostAndUsage MUST have its data generator-calculated split cost allocation method documented and accessible to practitioners.
    - CostAndUsage SHOULD offer data generator-calculated split cost allocation on an opt-in basis.
    - CostAndUsage MAY contain records for concepts not related to resource usage, when it aligns with the documented data generator-calculated split cost allocation method.
    - CostAndUsage MAY contain records for unused or unallocated usage from the *origin charge* as separate *allocated*

charges, when it aligns with the documented data generator-calculated split cost allocation method.
    - CostAndUsage MAY contain *allocated charges* with apportioned costs for unused or unallocated usage, when it aligns with the documented data generator-calculated split cost allocation method.
- CostAndUsage SHOULD reflect all applied discounts in *charges* they pertain to.
- CostAndUsage SHOULD NOT represent applied discounts as separate negating or offsetting *charges*.
- CostAndUsage *FOCUS columns* MUST conform to DataGeneratorCalculatedSplitCostAllocationHandling requirements when the data generator supports data generator-calculated split cost allocation.
- CostAndUsage *FOCUS columns* MUST conform to FocusColumnHandling requirements.
- CostAndUsage *FOCUS columns* MUST conform to NullHandling requirements.
- CostAndUsage *custom columns* MUST conform to CustomColumnHandling requirements.

Dataset ID

CostAndUsage

Display Name

Cost and Usage

Description

Describes the cost and usage incurred through using or purchasing a service provider's *resources* or *services*.

Version Introduced

0.5

3.1.1. Allocated Method ID

Allocated Method ID is the unique identifier for the allocated method defined by the service provider which was used for the Data Generator-Calculated Split Cost Allocation. This unique identifier can be used to find how the allocated charge was calculated in the provider's documentation.

3.1.1.1. Requirements

AllocatedMethodId MUST adhere to the following requirements:

- AllocatedMethodId MUST be of type String.
- AllocatedMethodId MUST conform to StringHandling requirements.
- AllocatedMethodId MUST adhere to the following nullability requirements:
    - AllocatedMethodId MUST be null when a *charge* is not related to a data generator-calculated split cost allocation.
    - AllocatedMethodId MUST NOT be null when a *charge* is related to a data generator-calculated split cost allocation.
- Data generator-calculated split cost allocation method documentation MUST reference a single AllocatedMethodId value.

3.1.1.2. Column ID

AllocatedMethodId

3.1.1.3. Display Name

Allocated Method ID

3.1.1.4. Description

A unique identifier defining the method of data generator-calculated split cost allocation.

3.1.1.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.1.6. Version Introduced

1.3

3.1.2. Allocated Method Details

Allocated Method Details provides information about how resources are allocated when usage records are split to support cost allocation requirements.

Allocated Method Details consists of a valid JSON object which contains an array consisting of key-value objects describing the one or more factors that determined the split cost allocation. Each object consists of FOCUS-defined property keys but can be extended to provide additional details about the allocation.

3.1.2.1. Requirements

3.1.2.1.1. Column Requirements

AllocatedMethodDetails MUST adhere to the following requirements:

- AllocatedMethodDetails MUST be of type JSON Object (serialized as a String where necessary).
- AllocatedMethodDetails MUST conform to StringHandling requirements.
- AllocatedMethodDetails MUST conform to JsonObjectFormat requirements.
- AllocatedMethodDetails MUST adhere to the following nullability requirements:
  - AllocatedMethodDetails MUST be null when a charge is not related to a data generator-calculated split cost allocation.
  - AllocatedMethodDetails SHOULD NOT be null when a charge is related to a data generator-calculated split cost allocation.
- AllocatedMethodDetails MUST conform to AllocatedMethodDetailsObject requirements when AllocatedMethodDetails is not null.

3.1.2.2. Allocated Method Details Object

Allocated Method Details consists of a valid JSON object with a top level key of Elements containing an Array of entry objects. Each entry object consists of FOCUS-defined property keys but can be extended to provide additional details about the allocation.

The following section details the normative requirements for the AllocatedMethodDetailsObject and its nested properties. For a logical overview of the expected content, see the Schema Structure and Object Example sections.

3.1.2.2.1. Object Requirements

The AllocatedMethodDetailsObject MUST adhere to the following requirements:

- AllocatedMethodDetailsObject MUST conform to the AllocatedMethodDetailsObjectSchema JSON Schema.
- AllocatedMethodDetailsObject.Elements[*].AllocatedRatio MUST represent the allocated charge's percentage of the

origin charge.

- The sum of AllocatedMethodDetailsObject.Elements[*].AllocatedRatio across all allocated charges related to a single origin charge MUST be equal to 1 (100%).
- AllocatedMethodDetailsObject.Elements[*].UsageUnit SHOULD conform to UnitFormat requirements.
- AllocatedMethodDetailsObject.Elements[*].UsageUnit MUST represent the unit or component of data generator's documented AllocationMethod which was used to determine the AllocatedMethodDetailsObject.Elements[*].AllocatedRatio value.
- AllocatedMethodDetailsObject.Elements[*].UsageQuantity SHOULD capture the quantity or volume of the AllocatedMethodDetailsObject.Elements[*].UsageUnit measured by the data generator that was used to determine the AllocatedMethodDetailsObject.Elements[*].AllocatedRatio value.

3.1.2.2.2. Object Schema Structure

AllocatedMethodDetails contains a structured JSON object defining the allocation properties used to calculate a split cost allocation.

Top-Level Properties

Property	Type	Required	Description
`Elements`	Array	True	The parent array containing one or more objects which communicate information about how an allocated record was calculated.

Elements Object

The `Elements` array contains one or more objects, each of which contains the following entries:

Key	Type	Required	Description
`AllocatedRatio`	Numeric	True	Communicates the percentage of the *Origin Charge* that this *Allocated Charge* derived from the corresponding Allocated Method Id and Usage Unit property.
`UsageUnit`	String	Conditional	Communicates the aspect of the documented Allocation Method Id being used to calculate the Allocated Ratio property and what is being measured by Usage Quantity property. **Condition:** must be present if Usage Quantity is provided.
`UsageQuantity`	Numeric	False	Communicates the volume that was consumed or used, denominated in the Usage Unit property value.

3.1.2.2.3. Object Implementation Guidance

Custom Properties

To facilitate querying data across allocations and across data generators, a data generator may include one or more custom properties. These may be placed at the top level of the object (alongside `Elements`) or nested within the individual `Elements` objects. Custom keys must be prefixed with "x_" followed by PascalCase format (e.g., `x_MyCustomKey`) to make them easy to identify as well as prevent collisions with FOCUS-defined keys.

3.1.2.2.4. Object Example

Here is a basic example of the object format.

- For more detailed examples, please see this column's entry in the JSON Object Examples appendix entry here.
- For the JSON schema, please see Allocated Method Details Object Schema.

```
{
 "Elements" : [ {
  "AllocatedRatio" : 0.05,
  "UsageUnit" : "CPU",
  "UsageQuantity" : 0.5
 }, {
  "AllocatedRatio" : 0.1,
```

```
    "UsageUnit" : "Memory",
    "UsageQuantity" : 4
  } ]
}
```

3.1.2.2.5. Object ID

AllocatedMethodDetailsObject

3.1.2.2.6. Object Display Name

Allocated Method Details Object

3.1.2.3. Column ID

AllocatedMethodDetails

3.1.2.4. Display Name

Allocated Method Details

3.1.2.5. Description

A set of properties describing how resources are allocated in data generator-defined split cost allocation.

3.1.2.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension / Metric
Feature level	Recommended
Allows nulls	True
Data type	JSON
Value format	JSON Object Format
Object	AllocatedMethodDetailsObject

3.1.2.7. Version Introduced

1.3

3.1.3. Allocated Resource ID

An Allocated Resource ID is an identifier assigned by the data generator which cost is being allocated to in a Data Generator-Calculated Split Cost Allocation. The Allocated Resource ID is used to understand what the cost is being allocated to in *charges* where the data generator is allocating costs to something other than the *charge's* ResourceID, as is the case for allocated charges.

3.1.3.1. Requirements

AllocatedResourceId MUST adhere to the following requirements:

- AllocatedResourceId MUST be of type String.
- AllocatedResourceId MUST conform to StringHandling requirements.
- AllocatedResourceId MUST adhere to the following nullability requirements:
  - AllocatedResourceId MUST be null when a *charge* is not related to a data generator-calculated split cost allocation.
  - AllocatedResourceId MUST be null when a *charge* represents the unallocated portion of the origin *charge* after split cost allocation.
  - AllocatedResourceId MUST NOT be null when a *charge* represents the allocated portion of the origin *charge*.
- When AllocatedResourceId is not null, AllocatedResourceId MUST adhere to the following requirements:
  - AllocatedResourceId SHOULD be a locally unique identifier within the associated ResourceId and ChargePeriod.
  - AllocatedResourceId MAY NOT be unique across ResourceId or ChargePeriod values.

3.1.3.2. Column ID

AllocatedResourceId

3.1.3.3. Display Name

Allocated Resource ID

3.1.3.4. Description

The identifier of the object to which cost is allocated in data generator-calculated split cost allocation.

3.1.3.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.3.6. Version Introduced

1.3

3.1.4. Allocated Resource Name

The Allocated Resource Name is a display name which cost is being allocated to in aData Generator-Calculated Split Cost Allocation. The Allocated Resource Name is used to understand what the cost is being allocated to in*charges* where the service provider is allocating costs to something other than the charge's ResourceID, as is the case for allocated charges.

3.1.4.1. Requirements

AllocatedResourceName MUST adhere to the following requirements:

- AllocatedResourceName MUST be of type String.
- AllocatedResourceName MUST conform to StringHandling requirements.

- AllocatedResourceName MUST adhere to the following nullability requirements:
  - AllocatedResourceName MUST be null when AllocatedResourceId is null.
  - AllocatedResourceName MUST NOT be null when AllocatedResourceId is not null.
- AllocatedResourceName MAY duplicate AllocatedResourceId when a separate display name is not applicable.

3.1.4.2. Column ID

AllocatedResourceName

3.1.4.3. Display Name

Allocated Resource Name

3.1.4.4. Description

The display name of the object to which cost is allocated in data generator-calculated split cost allocation.

3.1.4.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.4.6. Version Introduced

1.3

3.1.5. Allocated Tags

The Allocated Tags column represents the set of *tags* assigned to *tag sources* which are specifically applicable to *allocated charges* resulting from a data generator-calculated split cost allocation.

3.1.5.1. Requirements

AllocatedTags MUST adhere to the following requirements:

- AllocatedTags MUST be of type JSON Object (serialized as a String where necessary).
- AllocatedTags MUST conform to StringHandling requirements.
- AllocatedTags MUST conform to KeyValueFormat requirements.
- AllocatedTags MUST adhere to the following nullability requirements:
  - AllocatedTags MUST be null when a *charge* is not related to a data generator-calculated split cost allocation.
  - AllocatedTags MAY be null in all other cases.
- When AllocatedTags is not null, AllocatedTags MUST adhere to the following requirements:
  - AllocatedTags MUST NOT include resource tags already present in Tags.
  - AllocatedTags MUST include all applicable user-defined and data generator-defined tags for the AllocatedResourceId.
  - Tag keys that do not support corresponding values MUST have a corresponding true (boolean) value set.
  - Tag values MUST match the provided values unless true (boolean) is applied to valueless tags.
- Data generator-defined tags MUST adhere to the following requirements:

    - Data generator-defined tag keys MUST be prefixed with a predetermined, data generator-specified tag key prefix that is unique to each corresponding provider-specified *tag scheme*.
    - Data generator-specified tag key prefixes SHOULD be publicly documented.
- User-defined tag keys in all user-defined *tag schemes* MUST include a predetermined, data generator-specified tag key prefix that is unique to each corresponding user-defined *tag scheme* when the data generator has more than one user-defined *tag scheme*.

3.1.5.2. Data Generator-Defined vs. User-Defined Tags

This example illustrates various tags produced from multiple user-defined and data generator-defined *tag schemes*. The first two tags illustrate examples from two different, user-defined *tag schemes*. The second tag is produced from a valueless, user-defined *tag scheme*, so the data generator also applies `true` as its default value.

The last two tags illustrate examples from two different, data generator-defined *tag schemes*.

```
{
   "userDefinedTagScheme1/foo": "bar",
   "userDefinedTagScheme2/foo": true,
   "providerDefinedTagScheme1/foo": "bar",
   "providerDefinedTagScheme2/foo": "bar"
}
```

3.1.5.3. Column ID

AllocatedTags

3.1.5.4. Display Name

Allocated Tags

3.1.5.5. Description

A set of tags assigned to tag sources that are applicable to *allocated charges* in data generator-calculated split cost allocation.

3.1.5.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	JSON
Value format	Key-Value Format

3.1.5.7. Version Introduced

1.3

3.1.6. Availability Zone

An *availability zone* is a host-provider-assigned identifier for a physically separated and isolated area within a Region that

provides high availability and fault tolerance. Availability Zone is commonly used for scenarios like analyzing cross-zone data transfer usage and the corresponding cost based on where *resources* are deployed.

3.1.6.1. Requirements

AvailabilityZone MUST adhere to the following requirements:

- AvailabilityZone MUST be of type String.
- AvailabilityZone MUST conform to StringHandling requirements.
- AvailabilityZone MUST be null when a *charge* is not specific to an *availability zone*.

3.1.6.2. Column ID

AvailabilityZone

3.1.6.3. Display Name

Availability Zone

3.1.6.4. Description

A host-provider-assigned identifier for a physically separated and isolated area within a Region that provides high availability and fault tolerance.

3.1.6.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Recommended
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.6.6. Version Introduced

0.5

3.1.7. Billed Cost

Billed Cost represents the cost of a *charge* as invoiced by the invoice issuer in a given *billing period*. Billed Cost differs from Effective Cost when *covering charges* (e.g., prepaid or postpaid commitment purchases) are recorded separately from the *covered charges* to which they are applied.

For all *charges*, Billed Cost reflects all applicable pricing adjustments (e.g., reduced pricing from *negotiated discounts* or *commitment discounts*). For purchase *charges*, Billed Cost includes any portion invoiced in the given *billing period*. For usage *charges*, Billed Cost excludes any portion *covered* by related purchase *charges* (e.g., *covering charges* such as *commitments*, prepayments, or marketplace purchases), regardless of when those related *charges* are invoiced.

Billed Cost is denominated in the Billing Currency. Billed Cost is commonly used to support FinOps activities, including invoice reconciliation, *cash-based* forecasting, budgeting, and cost allocation.

3.1.7.1. Requirements

BilledCost MUST adhere to the following requirements:

- BilledCost MUST be of type Decimal.
- BilledCost MUST conform to NumericFormat requirements.
- BilledCost MUST NOT be null.
- BilledCost MUST be denominated in the BillingCurrency.
- BilledCost MUST reflect all applicable pricing adjustments, including but not limited to *negotiated discounts*, *commitment discounts*, and other applicable discount programs.
- BilledCost MUST NOT include any portion of a *covered charge* that is offset by a *covering charge*.
- BilledCost MUST be 0 for *charges* that are fully *covered* by one or more *covering charges*.
- BilledCost MUST reflect amounts as invoiced by the InvoiceIssuerName, not estimated or inferred values.
- BilledCost MUST be 0 for *charges* generated by entities that are not responsible or authorized for invoicing, to avoid double-counting when merging multiple *dataset instances*.
- The sum of BilledCost for a given InvoiceId and InvoiceIssuerName MUST NOT differ from the payable amount provided on the corresponding invoice by more than the Rounding Variance Tolerance when the corresponding invoice has been issued.
- The sum of BilledCost MAY differ from preliminary or estimated invoiced amounts when the corresponding invoice has not yet been issued.

3.1.7.2. Implementation Guidance

3.1.7.2.1. Handling Rounding Discrepancies

When validating the sum of BilledCost against the payable amount on an issued invoice, the totals may not be perfectly equal due to precision differences (e.g., 6 or more decimal places in cost and usage data vs. 2 decimal places on the invoice). The requirement allows for a maximum rounding error based on the statistical probability of rounding variance, which grows with the square root of the row count. For more information, see the Rounding Variance Tolerance appendix entry.

3.1.7.3. Column ID

BilledCost

3.1.7.4. Display Name

Billed Cost

3.1.7.5. Description

Cost of a *charge* as invoiced by the *invoice issuer* in a given *billing period*.

3.1.7.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Mandatory
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.7.7. Version Introduced

0.5

3.1.8. Billing Account ID

A Billing Account ID is an invoice-issuer-assigned identifier for a *billing account*. *Billing accounts* are commonly used for scenarios like grouping based on organizational constructs, invoice reconciliation and cost allocation strategies.

3.1.8.1. Requirements

BillingAccountId MUST adhere to the following requirements:

- BillingAccountId MUST be of type String.
- BillingAccountId MUST conform to StringHandling requirements.
- BillingAccountId MUST NOT be null.
- BillingAccountId MUST be a unique identifier within an *invoice issuer*.
- BillingAccountId SHOULD be a fully-qualified identifier.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.1.8.2. Column ID

BillingAccountId

3.1.8.3. Display Name

Billing Account ID

3.1.8.4. Description

The identifier assigned to a *billing account* by the invoice issuer.

3.1.8.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.1.8.6. Version Introduced

0.5

3.1.9. Billing Account Name

A Billing Account Name is a display name assigned to a *billing account*. *Billing accounts* are commonly used for scenarios like grouping based on organizational constructs, invoice reconciliation and cost allocation strategies.

3.1.9.1. Requirements

BillingAccountName MUST adhere to the following requirements:

- BillingAccountName MUST be of type String.
- BillingAccountName MUST conform to StringHandling requirements.
- BillingAccountName MUST NOT be null when the *invoice issuer* supports assigning a display name for the *billing account*.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.1.9.2. Column ID

BillingAccountName

3.1.9.3. Display Name

Billing Account Name

3.1.9.4. Description

The display name assigned to a *billing account*.

3.1.9.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.9.6. Version Introduced

0.5

3.1.10. Billing Account Type

Billing Account Type is an invoice-issuer-assigned name to identify the type of *billing account*. Billing Account Type is a readable display name and not a code. Billing Account Type is commonly used for scenarios like mapping FOCUS and provider constructs, summarizing costs across providers, or invoicing and chargeback.

3.1.10.1. Requirements

BillingAccountType MUST adhere to the following requirements:

- BillingAccountType MUST be of type String.

- BillingAccountType MUST conform to StringHandling requirements.
- BillingAccountType MUST adhere to the following nullability requirements:
  - BillingAccountType MUST be null when BillingAccountId is null.
  - BillingAccountType MUST NOT be null when BillingAccountId is not null.
- BillingAccountType MUST be a consistent, readable display value.

3.1.10.2. Column ID

BillingAccountType

3.1.10.3. Display Name

Billing Account Type

3.1.10.4. Description

An invoice-issuer-assigned name to identify the type of *billing account*.

3.1.10.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	<not specified>

3.1.10.6. Version Introduced

1.2

3.1.11. Billing Currency

Billing currency is an identifier that represents the currency that a *charge* for *resources* or *services* was billed in. Billing Currency is commonly used in scenarios where costs need to be grouped or aggregated.

3.1.11.1. Requirements

BillingCurrency MUST adhere to the following requirements:

- BillingCurrency MUST be of type String.
- BillingCurrency MUST conform to StringHandling requirements.
- BillingCurrency MUST conform to CurrencyFormat requirements.
- BillingCurrency MUST NOT be null.
- BillingCurrency MUST match the currency used in the invoice generated by the *invoice issuer*.
- BillingCurrency MUST be expressed in *national currency* (e.g., USD, EUR).

3.1.11.2. Column ID

BillingCurrency

3.1.11.3. Display Name

Billing Currency

3.1.11.4. Description

Represents the currency that a *charge* was billed in.

3.1.11.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Currency Format

3.1.11.6. Version Introduced

0.5

3.1.12. Billing Period End

Billing Period End represents the *exclusive end bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes *charges* for January since Billing Period Start represents the *inclusive start bound*, but does not include *charges* for February since Billing Period End represents the *exclusive end bound*.

3.1.12.1. Requirements

BillingPeriodEnd MUST adhere to the following requirements:

- BillingPeriodEnd MUST be of type Date/Time.
- BillingPeriodEnd MUST conform to DateTimeFormat requirements.
- BillingPeriodEnd MUST NOT be null.
- BillingPeriodEnd MUST be the *exclusive end bound* of the *billing period*.

3.1.12.2. Column ID

BillingPeriodEnd

3.1.12.3. Display Name

Billing Period End

3.1.12.4. Description

The *exclusive end bound* of a *billing period*.

3.1.12.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.1.12.6. Version Introduced

0.5

3.1.13. Billing Period Start

Billing Period Start represents the *inclusive start bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes *charges* for January since Billing Period Start represents the *inclusive start bound*, but does not include *charges* for February since BillingPeriodEnd represents the *exclusive end bound*.

3.1.13.1. Requirements

BillingPeriodStart MUST adhere to the following requirements:

- BillingPeriodStart MUST be of type Date/Time.
- BillingPeriodStart MUST conform to DateTimeFormat requirements.
- BillingPeriodStart MUST NOT be null.
- BillingPeriodStart MUST be the *inclusive start bound* of the *billing period*.

3.1.13.2. Column ID

BillingPeriodStart

3.1.13.3. Display Name

Billing Period Start

3.1.13.4. Description

The *inclusive start bound* of a *billing period*.

3.1.13.5. Content Constraints

Constraint	Value

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.1.13.6. Version Introduced

0.5

3.1.14. Capacity Reservation ID

A Capacity Reservation ID is the identifier assigned to a *capacity reservation* by the service provider. Capacity Reservation ID is commonly used for scenarios to allocate *charges* for capacity reservation usage.

3.1.14.1. Requirements

CapacityReservationId MUST adhere to the following requirements:

- CapacityReservationId MUST be of type String.
- CapacityReservationId MUST conform to StringHandling requirements.
- CapacityReservationId MUST adhere to the following nullability requirements:
  - CapacityReservationId MUST be null when a *charge* is not related to a *capacity reservation*.
  - CapacityReservationId MUST NOT be null when a *charge* represents the unused portion of a *capacity reservation*.
  - CapacityReservationId SHOULD NOT be null when a *charge* is related to a capacity reservation.
- When CapacityReservationId is not null, CapacityReservationId MUST adhere to the following requirements:
  - CapacityReservationId MUST be a unique identifier within the service provider.
  - CapacityReservationId SHOULD be a fully-qualified identifier.

3.1.14.2. Column ID

CapacityReservationId

3.1.14.3. Display Name

Capacity Reservation ID

3.1.14.4. Description

The identifier assigned to a *capacity reservation* by the service provider.

3.1.14.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True

Constraint	Value
Data type	String
Value format	<not specified>

3.1.14.6. Version Introduced

1.1

3.1.15. Capacity Reservation Status

Capacity Reservation Status indicates whether the *charge* represents either the consumption of the *capacity reservation* identified in the CapacityReservationId column or when the *capacity reservation* is unused.

3.1.15.1. Requirements

CapacityReservationStatus MUST adhere to the following requirements:

- CapacityReservationStatus MUST be of type String.
- CapacityReservationStatus MUST adhere to the following nullability requirements:
  - CapacityReservationStatus MUST be null when CapacityReservationId is null.
  - CapacityReservationStatus MUST NOT be null when CapacityReservationId is not null and ChargeCategory is "Usage".
- When CapacityReservationStatus is not null, CapacityReservationStatus MUST adhere to the following requirements:
  - CapacityReservationStatus MUST be one of the allowed values.
  - CapacityReservationStatus MUST be "Unused" when the *charge* represents the unused portion of a *capacity reservation*.
  - CapacityReservationStatus MUST be "Used" when the *charge* represents the used portion of a *capacity reservation*.

3.1.15.2. Allowed Values

Value	Description
Used	*Charges* that utilized a specific amount of a *capacity reservation*.
Unused	*Charges* that represent the unused portion of a *capacity reservation*.

3.1.15.3. Column ID

CapacityReservationStatus

3.1.15.4. Display Name

Capacity Reservation Status

3.1.15.5. Description

Indicates whether the *charge* represents either the consumption of a *capacity reservation* or when a *capacity reservation* is unused.

3.1.15.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Allowed values

3.1.15.7. Version Introduced

1.1

3.1.16. Charge Category

Charge Category represents the highest-level classification of a *charge* based on the nature of how it is billed. Charge Category is commonly used to identify and distinguish between types of *charges* that may require different handling.

3.1.16.1. Requirements

ChargeCategory MUST adhere to the following requirements:

- ChargeCategory MUST be of type String.
- ChargeCategory MUST NOT be null.
- ChargeCategory MUST be one of the allowed values.
- ChargeCategory MUST be "Usage" when the *charge* represents consumption of a service or resource.
- ChargeCategory MUST be "Purchase" when the *charge* represents acquisition of a service, resource, or *commitment*.
- ChargeCategory MUST be "Tax" when the *charge* represents taxes levied by the relevant authorities.
- ChargeCategory MUST be "Credit" when the *charge* represents a financial incentive or allowance unrelated to other charges.
- ChargeCategory MUST be "Adjustment" when the *charge* represents a billing modification that does not fall into other ChargeCategories.

3.1.16.2. Allowed Values

Value	Description
Usage	Positive or negative *charges* based on the quantity of a service or resource that was consumed over a given period of time including refunds.
Purchase	Positive or negative *charges* for the acquisition of a service or resource bought upfront or on a recurring basis including refunds.
Tax	Positive or negative applicable taxes that are levied by the relevant authorities including refunds. Tax *charges* may vary depending on factors such as the location, jurisdiction, and local or federal regulations.
Credit	Positive or negative *charges* granted by the service provider for various scenarios e.g., promotional credits or corrections to promotional credits.
Adjustment	Positive or negative *charges* the service provider applies that do not fall into other category values.

3.1.16.3. Column ID

ChargeCategory

3.1.16.4. Display Name

Charge Category

3.1.16.5. Description

Represents the highest-level classification of a *charge* based on the nature of how it is billed.

3.1.16.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.1.16.7. Version Introduced

0.5

3.1.17. Charge Class

Charge Class indicates whether a *charge* represents a *correction* to a previously *closed billing period*. Charge Class is commonly used to differentiate such corrections from all other charges, including both regularly incurred *charges* and *corrections* to *open billing periods*.

3.1.17.1. Requirements

ChargeClass MUST adhere to the following requirements:

- ChargeClass MUST be of type String.
- ChargeClass MUST adhere to the following nullability requirements:
  - ChargeClass MUST be null when the *charge* does not represent a correction to a previously *closed billing period*.
  - ChargeClass MUST NOT be null when the *charge* represents a correction to a previously *closed billing period*.
- ChargeClass MUST be "Correction" when ChargeClass is not null.

3.1.17.2. Allowed Values

Value	Description
Correction	Correction to a previously *closed billing period* (e.g., refunds and credit modifications).

3.1.17.3. Column ID

ChargeClass

3.1.17.4. Display Name

Charge Class

3.1.17.5. Description

Indicates whether a *charge* represents a correction to a previously *closed billing period*.

3.1.17.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	Allowed values

3.1.17.7. Version Introduced

1.0

3.1.18. Charge Description

A Charge Description provides a high-level context of a *row* without requiring additional discovery. This column is a self-contained summary of the *charge's* purpose and price. It typically covers a select group of corresponding details across a billing dataset or provides information not otherwise available.

3.1.18.1. Requirements

ChargeDescription MUST adhere to the following requirements:

- ChargeDescription MUST be of type String.
- ChargeDescription MUST conform to StringHandling requirements.
- ChargeDescription SHOULD NOT be null.
- ChargeDescription maximum length SHOULD be provided in the corresponding FOCUS Metadata Schema.

3.1.18.2. Column ID

ChargeDescription

3.1.18.3. Display Name

Charge Description

3.1.18.4. Description

Self-contained summary of the *charge's* purpose and price.

3.1.18.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.18.6. Version Introduced

1.0-preview

3.1.19. Charge Frequency

Charge Frequency indicates how often a *charge* will occur. Along with the charge period related columns, the Charge Frequency is commonly used to understand recurrence periods (e.g., monthly, yearly); forecast upcoming *charges*; and differentiate between one-time and recurring fees for purchases.

3.1.19.1. Requirements

ChargeFrequency MUST adhere to the following requirements:

- ChargeFrequency MUST be of type String.
- ChargeFrequency MUST NOT be null.
- ChargeFrequency MUST be one of the allowed values.
- ChargeFrequency MUST NOT be "Usage-Based" when ChargeCategory is "Purchase".

3.1.19.2. Allowed Values

Value	Description
One-Time	*Charges* that only happen once and will not repeat. One-time *charges* are typically recorded on the hour or day when the cost was incurred.
Recurring	*Charges* that repeat on a periodic cadence (e.g., weekly, monthly) regardless of whether the product or *service* was used. Recurring *charges* typically happen on the same day or point within every period. The charge date does not change based on how or when the *service* is used.
Usage-Based	*Charges* that repeat every time the *service* is used. Usage-based *charges* are typically recorded hourly or daily, based on the granularity of the cost data for the period when the *service* was used (referred to as *charge period*). Usage-based *charges* are not recorded when the *service* is not used.

3.1.19.3. Column ID

ChargeFrequency

3.1.19.4. Display Name

Charge Frequency

3.1.19.5. Description

Indicates how often a *charge* will occur.

3.1.19.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Recommended
Allows nulls	False
Data type	String
Value format	Allowed values

3.1.19.7. Version Introduced

1.0-preview

3.1.20. Charge Period End

Charge Period End represents the *exclusive end bound* of a *charge period*. For example, a time period where Charge Period Start is '2024-01-01T00:00:00Z' and Charge Period End is '2024-01-02T00:00:00Z' includes *charges* for January 1 since Charge Period Start represents the *inclusive start bound*, but does not include *charges* for January 2 since Charge Period End represents the *exclusive end bound*.

3.1.20.1. Requirements

ChargePeriodEnd MUST adhere to the following requirements:

- ChargePeriodEnd MUST be of type Date/Time.
- ChargePeriodEnd MUST conform to DateTimeFormat requirements.
- ChargePeriodEnd MUST NOT be null.
- ChargePeriodEnd MUST be the *exclusive end bound* of the effective period of the *charge*.

3.1.20.2. Column ID

ChargePeriodEnd

3.1.20.3. Display Name

Charge Period End

3.1.20.4. Description

The *exclusive end bound* of a *charge period*.

3.1.20.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False

Constraint	Value
Data type	Date/Time
Value format	Date/Time Format

3.1.20.6. Version Introduced

0.5

3.1.21. Charge Period Start

Charge Period Start represents the *inclusive start bound* of a *charge period*. For example, a time period where Charge Period Start is '2024-01-01T00:00:00Z' and Charge Period End is '2024-01-02T00:00:00Z' includes *charges* for January 1 since Charge Period Start represents the *inclusive start bound*, but does not include *charges* for January 2 since Charge Period End represents the *exclusive end bound*.

3.1.21.1. Requirements

ChargePeriodStart MUST adhere to the following requirements:

- ChargePeriodStart MUST be of type Date/Time.
- ChargePeriodStart MUST conform to DateTimeFormat requirements.
- ChargePeriodStart MUST NOT be null.
- ChargePeriodStart MUST be the *inclusive start bound* of the effective period of the *charge*.

3.1.21.2. Column ID

ChargePeriodStart

3.1.21.3. Display Name

Charge Period Start

3.1.21.4. Description

The *inclusive start bound* of a *charge period*.

3.1.21.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.1.21.6. Version Introduced

0.5

3.1.22. Commitment Discount Category

Commitment Discount Category indicates whether the *commitment discount* identified in the CommitmentDiscountId column is based on usage quantity or cost (aka "spend"). The CommitmentDiscountCategory column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.22.1. Requirements

CommitmentDiscountCategory MUST adhere to the following requirements:

- CommitmentDiscountCategory MUST be of type String.
- CommitmentDiscountCategory MUST adhere to the following nullability requirements:
  - CommitmentDiscountCategory MUST be null when CommitmentDiscountId is null.
  - CommitmentDiscountCategory MUST NOT be null when CommitmentDiscountId is not null.
- CommitmentDiscountCategory MUST be one of the allowed values.

3.1.22.2. Allowed Values

Value	Description
Spend	Commitment discounts that require a predetermined amount of spend.
Usage	Commitment discounts that require a predetermined amount of usage.

3.1.22.3. Column ID

CommitmentDiscountCategory

3.1.22.4. Display Name

Commitment Discount Category

3.1.22.5. Description

Indicates whether the *commitment discount* identified in the CommitmentDiscountId column is based on usage quantity or cost (aka "spend").

3.1.22.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Allowed values

3.1.22.7. Version Introduced

1.0-preview

3.1.23. Commitment Discount ID

A Commitment Discount ID is the identifier assigned to a *commitment discount* by the service provider. Commitment Discount ID is commonly used for scenarios like chargeback for *commitments* and savings per *commitment discount*. The CommitmentDiscountId column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.23.1. Requirements

CommitmentDiscountId MUST adhere to the following requirements:

- CommitmentDiscountId MUST be of type String.
- CommitmentDiscountId MUST conform to StringHandling requirements.
- CommitmentDiscountId MUST adhere to the following nullability requirements:
  - CommitmentDiscountId MUST be null when a *charge* is not related to a *commitment discount*.
  - CommitmentDiscountId MUST NOT be null when a *charge* is related to a *commitment discount*.
- When CommitmentDiscountId is not null, CommitmentDiscountId MUST adhere to the following requirements:
  - CommitmentDiscountId MUST be a unique identifier within the service provider.
  - CommitmentDiscountId MUST be equal to ResourceId when ChargeCategory is "Purchase" and the *charge* represents a purchase of that *commitment discount*.
  - CommitmentDiscountId MUST be equal to ResourceId when ChargeCategory is "Usage" and the *charge* represents an unused portion of that *commitment discount*.
  - CommitmentDiscountId SHOULD be a fully-qualified identifier.

3.1.23.2. Column ID

CommitmentDiscountId

3.1.23.3. Display Name

Commitment Discount ID

3.1.23.4. Description

The identifier assigned to a *commitment discount* by the service provider.

3.1.23.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.23.6. Version Introduced

1.0-preview

3.1.24. Commitment Discount Name

A Commitment Discount Name is the display name assigned to a *commitment discount*. The CommitmentDiscountName column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.24.1. Requirements

CommitmentDiscountName MUST adhere to the following requirements:

- CommitmentDiscountName MUST be of type String.
- CommitmentDiscountName MUST conform to StringHandling requirements.
- CommitmentDiscountName MUST adhere to the following nullability requirements:
  - CommitmentDiscountName MUST be null when CommitmentDiscountId is null.
  - When CommitmentDiscountId is not null, CommitmentDiscountName MUST adhere to the following requirements:
    - CommitmentDiscountName MUST NOT be null when a display name can be assigned to a *commitment discount*.
    - CommitmentDiscountName MAY be null when a display name cannot be assigned to a *commitment discount*.

3.1.24.2. Column ID

CommitmentDiscountName

3.1.24.3. Display Name

Commitment Discount Name

3.1.24.4. Description

The display name assigned to a *commitment discount*.

3.1.24.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.24.6. Version Introduced

1.0-preview

3.1.25. Commitment Discount Quantity

Commitment Discount Quantity is the amount of a *commitment discount* purchased or accounted for in *commitment discount* related *rows* that is denominated in Commitment Discount Units. The aggregated Commitment Discount Quantity across purchase records, pertaining to a particular Commitment Discount ID during its commitment *period*, represents the total Commitment Discount Units acquired with that commitment discount. For committed usage, the Commitment Discount Quantity is either the number of Commitment Discount Units consumed by a *row* that is covered by a *commitment discount* or

is the unused portion of a *commitment discount* over a *charge period*. Commitment Discount Quantity is commonly used in *commitment discount* analysis and optimization use cases and only applies to *commitment discounts*, not *negotiated discounts*.

When CommitmentDiscountCategory is "Usage" (usage-based *commitment discounts*), the Commitment Discount Quantity reflects the predefined amount of usage purchased or consumed. If *commitment discount flexibility* is applicable, this value may be further transformed based on additional, service-provider-specific requirements. When CommitmentDiscountCategory is "Spend" (spend-based *commitment discounts*), the Commitment Discount Quantity reflects the predefined amount of spend purchased or consumed. See Appendix: Commitment Discount Flexibility for more details around *commitment discount flexibility*.

3.1.25.1. Requirements

CommitmentDiscountQuantity MUST adhere to the following requirements:

- CommitmentDiscountQuantity MUST be of type Decimal.
- CommitmentDiscountQuantity MUST conform to NumericFormat requirements.
- CommitmentDiscountQuantity MUST adhere to the following nullability requirements:
  - CommitmentDiscountQuantity MUST be null when SkuPriceId is null.
  - When ChargeCategory is "Usage" or "Purchase" and CommitmentDiscountId is not null, CommitmentDiscountQuantity MUST adhere to the following requirements:
    - CommitmentDiscountQuantity MUST NOT be null when ChargeClass is not "Correction".
    - CommitmentDiscountQuantity MAY be null when ChargeClass is "Correction".
  - CommitmentDiscountQuantity MUST be null in all other cases.
- When CommitmentDiscountQuantity is not null and ChargeCategory is "Purchase", CommitmentDiscountQuantity MUST adhere to the following requirements:
  - CommitmentDiscountQuantity MUST be the quantity of CommitmentDiscountUnit, paid fully or partially upfront, that is eligible for consumption over the *commitment discount's term* when ChargeFrequency is "One-Time".
  - CommitmentDiscountQuantity MUST be the quantity of CommitmentDiscountUnit that is eligible for consumption for each *charge period* that corresponds with the purchase when ChargeFrequency is "Recurring".
- When CommitmentDiscountQuantity is not null and ChargeCategory is "Usage", CommitmentDiscountQuantity MUST adhere to the following requirements:
  - CommitmentDiscountQuantity MUST be the metered quantity of CommitmentDiscountUnit that is consumed in a given *charge period* when CommitmentDiscountStatus is "Used".
  - CommitmentDiscountQuantity MUST be the remaining, unused quantity of CommitmentDiscountUnit in a given *charge period* when CommitmentDiscountStatus is "Unused".

3.1.25.2. Usability Constraints

Aggregation: When aggregating Commitment Discount Quantity for commitment utilization calculations, it's important to exclude *commitment discount* purchases (i.e., when Charge Category is "Purchase") that are paid to cover future eligible *charges* (e.g., *commitment discount*). Otherwise, when accounting for all upfront or accrued purchases, it's important to exclude *commitment discount* usage (i.e., when Charge Category is "Usage"). This exclusion helps prevent double counting of these quantities in the aggregation.

3.1.25.3. Column ID

CommitmentDiscountQuantity

3.1.25.4. Display Name

Commitment Discount Quantity

3.1.25.5. Description

The amount of a *commitment discount* purchased or accounted for in *commitment discount* related *rows* that is denominated in Commitment Discount Units.

3.1.25.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.25.7. Version Introduced

1.1

3.1.26. Commitment Discount Status

Commitment Discount Status indicates whether the *charge* corresponds with the consumption of a *commitment discount* identified in the CommitmentDiscountId column or the unused portion of the committed amount. The CommitmentDiscountStatus column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.26.1. Requirements

CommitmentDiscountStatus MUST adhere to the following requirements:

- CommitmentDiscountStatus MUST be of type String.
- CommitmentDiscountStatus MUST adhere to the following nullability requirements:
  - CommitmentDiscountStatus MUST be null when CommitmentDiscountId is null.
  - CommitmentDiscountStatus MUST NOT be null when CommitmentDiscountId is not null and ChargeCategory is "Usage".
- CommitmentDiscountStatus MUST be one of the allowed values.
- CommitmentDiscountStatus MUST be "Used" when the *charge* utilizes a specific amount of a given CommitmentDiscountId.
- CommitmentDiscountStatus MUST be "Unused" when the *charge* represents the unused portion of the given CommitmentDiscountId.

3.1.26.2. Allowed Values

Value	Description
Used	*Charges* that utilized a specific amount of a commitment discount.
Unused	*Charges* that represent the unused portion of the commitment discount.

3.1.26.3. Column ID

CommitmentDiscountStatus

3.1.26.4. Display Name

Commitment Discount Status

3.1.26.5. Description

Indicates whether the *charge* corresponds with the consumption of a *commitment discount* or the unused portion of the committed amount.

3.1.26.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Allowed values

3.1.26.7. Version Introduced

1.0

3.1.27. Commitment Discount Type

Commitment Discount Type is a service-provider-assigned name to identify the type of *commitment discount* applied to the *row*. The CommitmentDiscountType column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.27.1. Requirements

CommitmentDiscountType MUST adhere to the following requirements:

- CommitmentDiscountType MUST be of type String.
- CommitmentDiscountType MUST conform to StringHandling requirements.
- CommitmentDiscountType MUST adhere to the following nullability requirements:
  - CommitmentDiscountType MUST be null when CommitmentDiscountId is null.
  - CommitmentDiscountType MUST NOT be null when CommitmentDiscountId is not null.

3.1.27.2. Column ID

CommitmentDiscountType

3.1.27.3. Display Name

Commitment Discount Type

3.1.27.4. Description

A service-provider-assigned identifier for the type of *commitment discount* applied to the *row*.

3.1.27.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.27.6. Version Introduced

1.0-preview

3.1.28. Commitment Discount Unit

Commitment Discount Unit represents the service-provider-specified measurement unit indicating how a service provider measures the Commitment Discount Quantity of a *commitment discount*. The CommitmentDiscountUnit column is only applicable to *commitment discounts* and not *negotiated discounts*.

3.1.28.1. Requirements

CommitmentDiscountUnit MUST adhere to the following requirements:

- CommitmentDiscountUnit MUST be of type String.
- CommitmentDiscountUnit MUST conform to StringHandling requirements.
- CommitmentDiscountUnit SHOULD conform to UnitFormat requirements.
- CommitmentDiscountUnit MUST adhere to the following nullability requirements:
  - CommitmentDiscountUnit MUST be null when CommitmentDiscountQuantity is null.
  - CommitmentDiscountUnit MUST NOT be null when CommitmentDiscountQuantity is not null.
- When CommitmentDiscountUnit is not null, CommitmentDiscountUnit MUST adhere to the following requirements:
  - CommitmentDiscountUnit MUST remain consistent over time for a given CommitmentDiscountId.
  - CommitmentDiscountUnit MUST represent the unit used to measure the *commitment discount*.
  - When accounting for *commitment discount flexibility*, the CommitmentDiscountUnit value SHOULD reflect this consideration.

See Examples: Commitment Discount Flexibility for more details around *commitment discount flexibility*.

3.1.28.2. Column ID

CommitmentDiscountUnit

3.1.28.3. Display Name

Commitment Discount Unit

3.1.28.4. Description

The service-provider-specified measurement unit indicating how a service provider measures the Commitment Discount Quantity of a *commitment discount*.

3.1.28.5. Content Constraints

Constraint	Value

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Unit Format

3.1.28.6. Version Introduced

1.1

3.1.29. Commitment Program Eligibility Details

Commitment Program Eligibility Details identifies the *commitment programs* that could potentially cover *charges*, subject to *service provider* constraints. By distinguishing the pool of spend that was eligible to be covered, Commitment Program Eligibility Details provides the fundamental denominator for calculating precise commitment coverage metrics. This allows FinOps practitioners to accurately size the pool of uncovered spend that could realistically be covered by a future commitment. In this context, *commitment programs* include discount-bearing programs such as *commitment discounts* and *capacity reservations*, provided the service provider treats them as *commitments*.

3.1.29.1. Requirements

3.1.29.1.1. Column Requirements

CommitmentProgramEligibilityDetails MUST adhere to the following requirements:

- CommitmentProgramEligibilityDetails MUST be of type JSON Object (serialized as a String where necessary).
- CommitmentProgramEligibilityDetails MUST conform to StringHandling requirements.
- CommitmentProgramEligibilityDetails MUST conform to JsonObjectFormat requirements.
- CommitmentProgramEligibilityDetails MUST NOT be null when a charge is eligible for a *commitment program*, regardless of whether a *commitment* was actually applied to the charge.
- CommitmentProgramEligibilityDetails MUST NOT reflect restrictions (e.g., transient account configurations, quotas) that might temporarily prevent purchase or participation in a *commitment program*.
- CommitmentProgramEligibilityDetails MUST include all publicly available *commitment programs* for which the usage is eligible.
- CommitmentProgramEligibilityDetails MAY include negotiated *commitment programs* when the usage is eligible and the program is not broadly applicable across the service provider's service catalog.
- CommitmentProgramEligibilityDetails MUST NOT include data related to *commitment periods* or payment options.
- CommitmentProgramEligibilityDetails MUST conform to CommitmentProgramEligibilityDetailsObject requirements when CommitmentProgramEligibilityDetails is not null.

3.1.29.2. Commitment Program Eligibility Details Object

Commitment Program Eligibility Details consists of a valid JSON object with a top-level property key `CommitmentPrograms` containing an array of objects describing the specific *commitment programs* available for the usage charge.

3.1.29.2.1. Object Requirements

CommitmentProgramEligibilityDetailsObject MUST adhere to the following requirements:

- CommitmentProgramEligibilityDetailsObject MUST conform to the CommitmentProgramEligibilityDetailsObjectSchema JSON Schema.
- CommitmentProgramEligibilityDetailsObject.CommitmentPrograms[*].ProgramType MUST correspond to a *commitment*

program type supported by the service provider.

- CommitmentProgramEligibilityDetailsObject.CommitmentPrograms[*].ProgramType MUST match CommitmentDiscountType for one object in CommitmentProgramEligibilityDetailsObject.CommitmentPrograms when CommitmentDiscountType is not null.
- CommitmentProgramEligibilityDetailsObject.CommitmentPrograms[*].ProgramType SHOULD correspond to terminology disclosed by the service provider in public documentation.

3.1.29.2.2. Object Schema Structure

Top-Level Properties

Property	Type	Required	Description
CommitmentPrograms	Array	True	Array of objects identifying *commitment programs* for which the usage is eligible.

CommitmentPrograms Object

The CommitmentPrograms array contains one or more objects, each of which contains the following entries:

Key	ValueType	Required	Description
ProgramType	String	True	The specific type of commitment program (e.g., discount or capacity reservation) available for this usage.

3.1.29.2.3. Object Implementation Guidance

Custom Properties

To facilitate querying data across allocations and across service providers, a data generator may include one or more custom properties. These may be placed at the top level of the object (alongside CommitmentPrograms) or nested within the individual CommitmentPrograms objects. Custom keys must be prefixed with "x_" followed by PascalCase format (e.g., x_MyCustomKey) to make them easy to identify as well as prevent collisions with FOCUS-defined keys.

3.1.29.2.4. Object Example

Here is a basic example of the object format.

- For more detailed examples, please see this column's entry in the JSON Object Examples appendix entry here.
- For the JSON schema, please see Commitment Program Eligibility Details Object Schema.

```
{
 "CommitmentPrograms": [
  { "ProgramType": "Flexible Spend Plan" },
  { "ProgramType": "Resource Reservation" },
  { "ProgramType": "Advance Resource Commitment" },
  { "ProgramType": "Zonal Resource Commitment" }
 ]
}
```

3.1.29.2.5. Object ID

CommitmentProgramEligibilityDetailsObject

3.1.29.2.6. Object Display Name

Commitment Program Eligibility Details Object

3.1.29.3. Column ID

CommitmentProgramEligibilityDetails

3.1.29.4. Display Name

Commitment Program Eligibility Details

3.1.29.5. Description

The types of *commitment programs* available for a specific usage row.

3.1.29.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	JSON
Value format	JSON Object Format
Object	CommitmentProgramEligibilityDetailsObject

3.1.29.7. Version Introduced

1.4

3.1.30. Consumed Quantity

The Consumed Quantity represents the volume of a metered SKU associated with a *resource* or *service* used, based on the Consumed Unit. Consumed Quantity is often derived at a finer granularity or over a different time interval when compared to the Pricing Quantity (complementary to Pricing Unit) and focuses on *resource* and *service* consumption, not pricing and cost.

3.1.30.1. Requirements

ConsumedQuantity MUST adhere to the following requirements:

- ConsumedQuantity MUST be of type Decimal.
- ConsumedQuantity MUST conform to NumericFormat requirements.
- ConsumedQuantity MUST adhere to the following nullability requirements:
    - ConsumedQuantity MUST be null when SkuPriceId is null.
    - ConsumedQuantity MUST be null when ChargeCategory is not "Usage", or when ChargeCategory is "Usage" and CommitmentDiscountStatus is "Unused".
    - When ChargeCategory is "Usage" and CommitmentDiscountStatus is not "Unused", ConsumedQuantity MUST adhere to the following requirements:
        - ConsumedQuantity MUST NOT be null when ChargeClass is not "Correction".
        - ConsumedQuantity MAY be null when ChargeClass is "Correction".

3.1.30.2. Column ID

ConsumedQuantity

3.1.30.3. Display Name

Consumed Quantity

3.1.30.4. Description

The volume of a metered SKU associated with a *resource* or *service* used, based on the Consumed Unit.

3.1.30.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.30.6. Version Introduced

1.0

3.1.31. Consumed Unit

The Consumed Unit represents a service-provider-specified measurement unit indicating how a service provider measures usage of a metered SKU associated with a *resource* or *service*. Consumed Unit complements the Consumed Quantity metric. It is often listed at a finer granularity or over a different time interval when compared to Pricing Unit (complementary to Pricing Quantity), and focuses on *resource* and *service* consumption, not pricing and cost.

3.1.31.1. Requirements

ConsumedUnit MUST adhere to the following requirements:

- ConsumedUnit MUST be of type String.
- ConsumedUnit MUST conform to StringHandling requirements.
- ConsumedUnit SHOULD conform to UnitFormat requirements.
- ConsumedUnit MUST adhere to the following nullability requirements:
  - ConsumedUnit MUST be null when ConsumedQuantity is null.
  - ConsumedUnit MUST NOT be null when ConsumedQuantity is not null.

3.1.31.2. Column ID

ConsumedUnit

3.1.31.3. Display Name

Consumed Unit

3.1.31.4. Description

Service-provider-specified measurement unit indicating how a service provider measures usage of a metered SKU associated with a *resource* or *service*.

3.1.31.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Unit Format recommended

3.1.31.6. Version Introduced

1.0

3.1.32. Contract Applied

Contract Applied is a set of properties that associate a *charge* with one or more *contract commitments*, denoted as key-value pairs in a JSON object. Contract Applied allows the practitioner to track the progress of the commitments to which they have agreed with a service provider.

3.1.32.1. Requirements

3.1.32.1.1. Column Requirements

ContractApplied MUST adhere to the following requirements:

- ContractApplied MUST be of type JSON Object (serialized as a String where necessary).
- ContractApplied MUST conform to StringHandling requirements.
- ContractApplied MUST conform to JsonObjectFormat requirements.
- ContractApplied MUST NOT be null when one or more *contract commitments* are applied to the *charge*.
- ContractApplied MUST conform to ContractAppliedObject requirements when ContractApplied is not null.

3.1.32.2. Contract Applied Object

Contract Applied Object consists of a valid JSON object which contains an array of key-value objects describing the one or more contract commitments applied to the *charge*. Each object consists of FOCUS-defined property keys but can be extended to provide additional details about the contract application.

The following section details the normative requirements for the ContractAppliedObject and its nested properties. For a logical overview of the expected content, see the Schema Structure and Object Example sections.

3.1.32.2.1. Object Requirements

ContractAppliedObject MUST adhere to the following requirements:

- ContractAppliedObject MUST conform to the ContractAppliedObjectSchema JSON Schema.
- ContractAppliedObject.Elements[*].ContractId MUST be a unique identifier within the service provider.

- ContractAppliedObject.Elements[*].ContractId SHOULD be a fully-qualified identifier.
- ContractAppliedObject.Elements[*].ContractCommitmentId MUST be a unique identifier within the service provider.
- ContractAppliedObject.Elements[*].ContractCommitmentId SHOULD be a fully-qualified identifier.
- ContractAppliedObject.Elements[*].ContractCommitmentId MUST have one and only one parent ContractAppliedObject.Elements[*].ContractId.
- ContractAppliedObject.Elements[*].ContractCommitmentId MUST match ResourceId when ChargeCategory is "Purchase" and the *charge* represents a purchase of that *contract commitment*.
- ContractAppliedObject.Elements[*].ContractCommitmentId MUST match ResourceId when ChargeCategory is "Usage" and the *charge* represents an unused portion of that *contract commitment*.
- ContractAppliedObject.Elements[*].ContractCommitmentId MAY match ContractAppliedObject.Elements[*].ContractId.
- ContractAppliedObject.Elements[*].ContractCommitmentAppliedCost MUST be denominated in the BillingCurrency.
- ContractAppliedObject.Elements[*].ContractCommitmentAppliedQuantity MUST be denominated in the ContractAppliedObject.Elements[*].ContractCommitmentAppliedUnit.
- ContractAppliedObject.Elements[*].ContractCommitmentAppliedUnit SHOULD conform to UnitFormat requirements.

3.1.32.2.2. Object Schema Structure

ContractApplied contains a structured JSON object defining the allocation and application of a *charge* against specific contract commitments.

Top-Level Properties

Property	Type	Required	Description
Elements	Array	True	The parent array containing one or more objects which communicate information about how contract commitments were applied to the *charge*.

Elements Object

The Elements array contains one or more objects, each of which contains the following entries:

Key	Type	Required	Description
ContractId	String	Yes	A service-provider-assigned identifier for a contract describing the agreed terms between a service provider and a customer. Contracts can include commitment to a certain amount of spend or usage over an agreed period of time.
ContractCommitmentId	String	Yes	A service-provider-assigned identifier describing an agreement agreed between a service provider and a customer.
ContractCommitmentAppliedCost	Decimal	Conditional	The cost of the *charge* applied to the contract line item. It is associated with the contract line item via Contract Commitment ID, and is commonly used for monitoring progress towards fulfilling contractual commitments that may facilitate discounts for *resources* or *services* as agreed between a service provider and a customer. **Condition:** Must be present if Quantity and Unit are not provided.
ContractCommitmentAppliedQuantity	Decimal	Conditional	The quantity of the *charge* applied to the contract line item. It is associated with the contract line item via Contract Commitment ID, and is commonly used for monitoring the progress towards fulfilling contractual commitments that may facilitate discounts for *resources* or *services* as agreed between a service provider and a customer. **Condition:** Must be present if Cost is not provided.

Key	Type	Required	Description
`ContractCommitmentAppliedUnit`	String	Conditional	A service-provider-specified measurement unit for the usage declared in Contract Commitment Applied Quantity. It complements the Contract Commitment Applied Quantity metric. **Condition:** Must be present if Contract Commitment Applied Quantity is provided.

3.1.32.2.3. Object Implementation Guidance

Custom Properties

To facilitate querying data across allocations and across service providers, a data generator may include one or more custom properties. These may be placed at the top level of the object (alongside `Elements`) or nested within the individual `Elements` objects. Custom keys must be prefixed with "x_" followed by PascalCase format (e.g., `x_MyCustomKey`) to make them easy to identify as well as prevent collisions with FOCUS-defined keys.

3.1.32.2.4. Object Example

Here is a basic example of the object format.

- For more detailed examples, please see this column's entry in the JSON Object Examples appendix entry here.
- For the JSON schema, please see Contract Applied Object Schema.

```
{
 "Elements": [
  {
   "ContractId": "12345",
   "ContractCommitmentId": "23456",
   "ContractCommitmentAppliedCost": 500000.00
  },
  {
   "ContractId": "12345",
   "ContractCommitmentId": "34567",
   "ContractCommitmentAppliedQuantity": 10000.00,
   "ContractCommitmentAppliedUnit": "compute_hours"
  }
 ]
}
```

3.1.32.2.5. Object ID

ContractAppliedObject

3.1.32.2.6. Object Display Name

Contract Applied Object

3.1.32.3. Column ID

ContractApplied

3.1.32.4. Display Name

3.1.32.5. Description

A set of properties that associate a *charge* with one or more *contract commitments*.

3.1.32.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension / Metric
Feature level	Conditional
Allows nulls	True
Data type	JSON
Value format	JSON Object Format
Object	ContractAppliedObject

3.1.32.7. Version Introduced

1.3

3.1.33. Contracted Cost

Contracted Cost represents the cost calculated by multiplying *contracted unit price* and the corresponding Pricing Quantity. Contracted Cost is denominated in the Billing Currency and is commonly used for calculating savings based on negotiation activities, by comparing it with List Cost. If *negotiated discounts* are not applicable, the Contracted Cost defaults to the List Cost.

3.1.33.1. Requirements

ContractedCost MUST adhere to the following requirements:

- ContractedCost MUST be of type Decimal.
- ContractedCost MUST conform to NumericFormat requirements.
- ContractedCost MUST NOT be null.
- ContractedCost MUST be denominated in the BillingCurrency.
- When ContractedUnitPrice is null, ContractedCost MUST adhere to the following requirements:
  - ContractedCost of a *charge* calculated based on other *charges* (e.g., when the ChargeCategory is "Tax") MUST be calculated based on the ContractedCost of those related *charges*.
  - ContractedCost of a *charge* unrelated to other *charges* (e.g., when the ChargeCategory is "Credit") MUST be equal to the BilledCost.
- ContractedCost MUST equal the product of ContractedUnitPrice and PricingQuantity when ContractedUnitPrice is not null and PricingQuantity is not null.

3.1.33.2. Usability Constraints

Aggregation: When aggregating Contracted Cost for savings calculations, it's important to exclude either Charge Category "Purchase" *charges* (one-time and recurring) that are paid to cover future eligible *charges* (e.g., commitment discount) or the covered Charge Category "Usage" *charges* themselves. This exclusion helps prevent double counting of these *charges* in the aggregation. Which set of *charges* to exclude depends on whether costs are aggregated on a billed basis (exclude covered *charges*) or accrual basis (exclude Purchases for future *charges*). For instance, *charges* categorized as Charge Category "Purchase" and their related Charge Category "Tax" *charges* for a Commitment Discount might be excluded from an accrual basis cost aggregation of Contracted Cost. This is because the "Usage" and "Tax" charge records provided during the commitment *period* already specify the Contracted Cost. Purchase *charges* that cover future eligible *charges* can be identified by filtering for Charge Category "Purchase" records with a Billed Cost greater than 0 and an Effective Cost equal to 0.

3.1.33.3. Column ID

ContractedCost

3.1.33.4. Display Name

Contracted Cost

3.1.33.5. Description

Cost calculated by multiplying *contracted unit price* and the corresponding Pricing Quantity.

3.1.33.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Mandatory
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.33.7. Version Introduced

1.0

3.1.34. Contracted Unit Price

The Contracted Unit Price represents the agreed-upon unit price for a singlePricing Unit of the associated SKU, inclusive of *negotiated discounts*, if present, while excluding negotiated *commitment discounts* or any other discounts. This price is denominated in the Billing Currency. The Contracted Unit Price is commonly used for calculating savings based on negotiation activities. If negotiated discounts are not applicable, the Contracted Unit Price defaults to the List Unit Price.

3.1.34.1. Requirements

ContractedUnitPrice MUST adhere to the following requirements:

- ContractedUnitPrice MUST be of type Decimal.
- ContractedUnitPrice MUST conform to NumericFormat requirements.
- ContractedUnitPrice MUST adhere to the following nullability requirements:
    - ContractedUnitPrice MUST be null when SkuPriceId is null.
    - ContractedUnitPrice MUST be null when ChargeCategory is "Tax".
    - ContractedUnitPrice MUST NOT be null when SkuPriceId is not null.
    - ContractedUnitPrice MUST NOT be null when ChargeCategory is "Usage" or "Purchase" andChargeClass is not "Correction".
    - ContractedUnitPrice MAY be null in all other cases.
- When ContractedUnitPrice is not null, ContractedUnitPrice MUST adhere to the following requirements:
    - ContractedUnitPrice MUST be a non-negative decimal value.
    - ContractedUnitPrice MUST be denominated in the BillingCurrency.

3.1.34.2. Usability Constraints

Aggregation: Column values should only be viewed in the context of their row and not aggregated to produce a total.

3.1.34.3. Column ID

ContractedUnitPrice

3.1.34.4. Display Name

Contracted Unit Price

3.1.34.5. Description

The agreed-upon unit price for a single Pricing Unit of the associated SKU, inclusive of negotiated discounts, if present, while excluding negotiated commitment discounts or any other discounts.

3.1.34.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid non-negative decimal value

3.1.34.7. Version Introduced

1.0

3.1.35. Effective Cost

Effective Cost represents the cost of a *charge* based on the *resources* used, *services* used, or *contract commitments* recognized in a given *charge period*. Effective Cost differs from Billed Cost when *covering charges* (e.g., prepaid or postpaid commitment purchases) are recorded separately from the *covered charges* to which they are applied.

For all *charges*, Effective Cost reflects all applicable pricing adjustments (e.g., reduced pricing from *negotiated discounts* or *commitment discounts*). For usage *charges*, Effective Cost includes the recognized portion of *Billed Cost* from related purchase *charges* (e.g., amortized portions of prepayments, drawdowns). For purchase *charges*, Effective Cost excludes any amounts recognized in related usage *charges* (e.g., usage *covered* by *covering charges* such as *commitments*, prepayments, or marketplace purchases which draw down based on usage), regardless of when those related *charges* are invoiced.

Effective Cost is denominated in the Billing Currency. Effective Cost is commonly used to support FinOps activities, including *accrual-based* reporting, forecasting, and cost allocation.

3.1.35.1. Requirements

EffectiveCost MUST adhere to the following requirements:

- EffectiveCost MUST be of type Decimal.
- EffectiveCost MUST conform to NumericFormat requirements.
- EffectiveCost MUST NOT be null.
- EffectiveCost MUST be denominated in the BillingCurrency.
- EffectiveCost MUST reflect all applicable pricing adjustments, including but not limited to *negotiated discounts*, *commitment discounts*, and other applicable discount programs.
- EffectiveCost MUST equal BilledCost when ChargeCategory is "Usage" and the *charge* is not *covered* by other eligible *charges*.
- EffectiveCost MUST equal BilledCost when ChargeCategory is "Purchase" and the *charge* is neither intended to cover other eligible *charges* nor *covered* by other eligible *charges*.
- EffectiveCost MUST equal BilledCost when ChargeCategory is "Tax" or "Credit".
- EffectiveCost MAY differ from BilledCost when ChargeCategory is "Adjustment".
- EffectiveCost MUST include any portion of the BilledCost of *covering* purchase *charges* (i.e., ChargeCategory is "Purchase") that is applied to this *charge*.
- EffectiveCost MUST be 0 when ChargeCategory is "Purchase" and the purchase is intended to cover related eligible *charges*. This requirement applies even when the *covered charges* originate from different CostAndUsage *dataset instances*, possibly from a different ServiceProviderName.
- EffectiveCost MUST be 0 for *charges* generated by entities that do not originate the cost and usage data, to avoid double-counting when merging multiple *dataset instances*.
- The sum of EffectiveCost across all related *covering* and *covered charges* MUST equal the sum of BilledCost across the same set of *charges*, within the *charge period* of the *covering charges*, when both the *covering* and *covered charges* are present in the *dataset instance*.
- The sum of EffectiveCost for a given *billing period* MAY differ from the sum of BilledCost when *covered* and *covering charges* span multiple *billing periods* or *billing accounts*, or when only one side of a covering relationship is present in the *dataset instance*.

3.1.35.2. Column ID

EffectiveCost

3.1.35.3. Display Name

Effective Cost

3.1.35.4. Description

Cost of a *charge* based on the *resources* used, *services* used, or *contract commitments* recognized in a given *charge period*.

3.1.35.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Mandatory
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.35.6. Version Introduced

0.5

3.1.36. Host Provider Name

Host Provider Name is the name of the entity that provides the underlying infrastructure on which the *resources* or *services* of the Service Provider are deployed.

In some instances, the host provider and the service provider are the same entity: the provider hosts their own services. In other instances, the host provider and the service provider are separate entities, though the service provider may or may not expose the host provider and/or allow the customer to select the host provider.

3.1.36.1. Requirements

HostProviderName MUST adhere to the following requirements:

- HostProviderName MUST be of type String.
- HostProviderName MUST conform to StringHandling requirements.
- HostProviderName MUST adhere to the following nullability requirements:
  - HostProviderName MAY be NULL when the associated ServiceName does not involve deployment on any underlying infrastructure (e.g., professional services, software licenses).
  - HostProviderName MAY be NULL when the information about the entity providing the underlying infrastructure cannot be uniquely determined (e.g., when the ChargeCategory is "Tax" or "Adjustment").
  - HostProviderName MUST NOT be null in all other cases.
- When HostProviderName is not null, HostProviderName values MUST adhere to the following requirements:
  - HostProviderName MUST reflect the name of the host provider when explicitly selected by the customer.
  - HostProviderName MUST reflect the name of the host provider when the service provider exposes the underlying hosting provider.
  - HostProviderName MUST match ServiceProviderName in all other cases.

See Appendix: Participating Entity Identification Examples section for examples of Host Provider values across various use case scenarios.

3.1.36.2. Column ID

HostProviderName

3.1.36.3. Display Name

Host Provider Name

3.1.36.4. Description

The name of the entity whose *resources* are used by the Service Provider to make their *resources* or *services* available.

3.1.36.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.36.6. Version Introduced

1.3

3.1.37. Invoice Detail ID

Invoice Detail ID is the invoice-issuer-assigned identifier for an Invoice Detail record encapsulating charges in the corresponding billing period for a given billing account. This identifier allows FinOps practitioners to map specific line items from an invoice to the granular charge data, facilitating detailed reconciliation and auditability.

3.1.37.1. Requirements

InvoiceDetailId MUST adhere to the following requirements:

- InvoiceDetailId MUST be of type String.
- InvoiceDetailId MUST conform to StringHandling requirements.
- InvoiceDetailId MUST adhere to the following nullability requirements:
  - InvoiceDetailId MUST be null when the charge is not associated either with an invoice or with a pre-generated provisional invoice.
  - InvoiceDetailId MUST NOT be null when the charge is associated with either an issued invoice or a pre-generated provisional invoice.
- InvoiceDetailId MAY be generated prior to an invoice being issued.
- InvoiceDetailId MUST uniquely identify a specific record within a given InvoiceId.

3.1.37.2. Column ID

InvoiceDetailId

3.1.37.3. Display Name

Invoice Detail ID

3.1.37.4. Description

The invoice-issuer-assigned identifier for an Invoice Detail record encapsulating charges in the corresponding billing period for a given billing account.

3.1.37.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.37.6. Version Introduced

1.4

3.1.38. Invoice ID

An Invoice ID is an invoice-issuer-assigned identifier for an invoice encapsulating *charges* in the corresponding *billing period*

for a given *billing account*. Invoices are commonly used for scenarios like tracking billing transactions, facilitating payment processes and for performing invoice reconciliation between *charges* and billing periods.

3.1.38.1. Requirements

InvoiceId MUST adhere to the following requirements:

- InvoiceId MUST be of type String.
- InvoiceId MUST conform to StringHandling requirements.
- InvoiceId MUST adhere to the following nullability requirements:
  - InvoiceId MUST be null when the *charge* is not associated either with an invoice or with a pre-generated provisional invoice.
  - InvoiceId MUST NOT be null when the *charge* is associated with either an issued invoice or a pre-generated provisional invoice.
- InvoiceId MAY be generated prior to an invoice being issued.
- InvoiceId MUST be associated with the related *charge* and BillingAccountId when a pre-generated invoice or provisional invoice exists.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.1.38.2. Column ID

InvoiceId

3.1.38.3. Display Name

Invoice ID

3.1.38.4. Description

The invoice-issuer-assigned identifier for an invoice encapsulating *charges* in the corresponding billing period for a given billing account.

3.1.38.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.38.6. Version Introduced

1.2

3.1.39. Invoice Issuer Name

Invoice Issuer Name is the name of the entity responsible for issuing payable invoices for the *resources* or *services* consumed. It is commonly used for cost analysis and reporting scenarios.

3.1.39.1. Requirements

InvoiceIssuerName MUST adhere to the following requirements:

- InvoiceIssuerName MUST be of type String.
- InvoiceIssuerName MUST conform to StringHandling requirements.
- InvoiceIssuerName MUST NOT be null.
- InvoiceIssuerName MUST represent the entity that issues invoices.

See Appendix: Participating Entity Identification Examples section for examples of Invoice Issuer Name values across various use case scenarios.

3.1.39.2. Column ID

InvoiceIssuerName

3.1.39.3. Display Name

Invoice Issuer Name

3.1.39.4. Description

The name of the entity responsible for invoicing for the *resources* or *services* consumed.

3.1.39.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.1.39.6. Version Introduced

0.5

3.1.40. List Cost

List Cost represents the cost calculated by multiplying the *list unit price* and the corresponding Pricing Quantity. List Cost is denominated in the Billing Currency and is commonly used for calculating savings based on various rate optimization activities by comparing it with Contracted Cost, Billed Cost and Effective Cost.

3.1.40.1. Requirements

ListCost MUST adhere to the following requirements:

- ListCost MUST be of type Decimal.
- ListCost MUST conform to NumericFormat requirements.
- ListCost MUST NOT be null.
- ListCost MUST be denominated in the BillingCurrency.

- When ListUnitPrice is null, ListCost MUST adhere to the following requirements:
  - ListCost of a *charge* calculated based on other *charges* (e.g., when the ChargeCategory is "Tax") MUST be calculated based on the ListCost of those related *charges*.
  - ListCost of a *charge* unrelated to other *charges* (e.g., when the ChargeCategory is "Credit") MUST be equal to the BilledCost.
- ListCost MUST equal the product of ListUnitPrice and PricingQuantity when ListUnitPrice is not null and PricingQuantity is not null.

3.1.40.2. Usability Constraints

Aggregation: When aggregating List Cost for savings calculations, it is important to exclude either Charge Category "Purchase" *charges* (one-time and recurring) that are paid to cover future eligible *charges* (e.g., commitment discount) or the covered Charge Category "Usage" *charges* themselves. This exclusion helps prevent double counting of these *charges* in the aggregation. Which set of *charges* to exclude depends on whether costs are aggregated on a billed basis (exclude covered *charges*) or accrual basis (exclude Purchases for future *charges*). For instance, *charges* categorized as Charge Category "Purchase" and their related Charge Category "Tax" *charges* for a Commitment Discount might be excluded from an accrual basis cost aggregation of List Cost. This is because the "Usage" and "Tax" charge records provided during the term of the commitment discount already specify the List Cost. Purchase *charges* that cover future eligible *charges* can be identified by filtering for Charge Category "Purchase" records with a Billed Cost greater than 0 and an Effective Cost equal to 0.

3.1.40.3. Column ID

ListCost

3.1.40.4. Display Name

List Cost

3.1.40.5. Description

Cost calculated by multiplying List Unit Price and the corresponding Pricing Quantity.

3.1.40.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Mandatory
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.40.7. Version Introduced

1.0-preview

3.1.41. List Unit Price

The List Unit Price represents the suggested service-provider-published unit price for a single Pricing Unit of the associated SKU, exclusive of any discounts. This price is denominated in the Billing Currency. The List Unit Price is commonly used for

calculating savings based on various rate optimization activities.

3.1.41.1. Requirements

ListUnitPrice MUST adhere to the following requirements:

- ListUnitPrice MUST be of type Decimal.
- ListUnitPrice MUST conform to NumericFormat requirements.
- ListUnitPrice MUST adhere to the following nullability requirements:
  - ListUnitPrice MUST be null when SkuPriceId is null.
  - ListUnitPrice MUST be null when ChargeCategory is "Tax".
  - ListUnitPrice MUST NOT be null when SkuPriceId is not null.
  - ListUnitPrice MUST NOT be null when ChargeCategory is "Usage" or "Purchase" andChargeClass is not "Correction".
  - ListUnitPrice MAY be null in all other cases.
- When ListUnitPrice is not null, ListUnitPrice MUST adhere to the following requirements:
  - ListUnitPrice MUST be a non-negative decimal value.
  - ListUnitPrice MUST be denominated in the BillingCurrency.

3.1.41.2. Usability Constraints

Aggregation: Column values should only be viewed in the context of their row and not aggregated to produce a total.

3.1.41.3. Column ID

ListUnitPrice

3.1.41.4. Display Name

List Unit Price

3.1.41.5. Description

The suggested service-provider-published unit price for a single Pricing Unit of the associated SKU, exclusive of any discounts.

3.1.41.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid non-negative decimal value

3.1.41.7. Version Introduced

1.0-preview

3.1.42. Pricing Category

Pricing Category describes the pricing model used for a *charge* at the time of use or purchase. It can be useful for distinguishing between *charges* incurred at the *list unit price* or a reduced price and exposing optimization opportunities, like increasing *commitment discount* coverage.

3.1.42.1. Requirements

PricingCategory MUST adhere to the following requirements:

- PricingCategory MUST be of type String.
- PricingCategory MUST adhere to the following nullability requirements:
  - PricingCategory MUST be null when SkuPriceId is null.
  - PricingCategory MUST be null when ChargeCategory is "Tax".
  - PricingCategory MUST NOT be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
  - PricingCategory MAY be null in all other cases.
- When PricingCategory is not null, PricingCategory MUST adhere to the following requirements:
  - PricingCategory MUST be one of the allowed values.
  - PricingCategory MUST be "Standard" when pricing is predetermined at the agreed upon rate for the billing account.
  - PricingCategory MUST be "Committed" when the *charge* is subject to an existing *commitment discount* and is not the purchase of the *commitment discount*.
  - PricingCategory MUST be "Dynamic" when pricing is determined by the service provider and may change over time, regardless of predetermined agreement pricing.
  - PricingCategory MUST be "Other" when there is a pricing model but none of the allowed values apply.

3.1.42.2. Allowed Values

Value	Description
Standard	*Charges* priced at the agreed upon rate for the billing account, including *negotiated discounts*. This pricing includes any flat rate and volume/tiered pricing but does not include dynamic pricing or reduced pricing due to the application of a *commitment discount*. This does include the purchase of a commitment discount at agreed upon rates.
Dynamic	*Charges* priced at a variable rate determined by the service provider. This includes any product or service with a unit price the service provider can change without notice, like interruptible or low priority *resources*.
Committed	*Charges* with reduced pricing due to the application of the *commitment discount* specified by the Commitment Discount ID.
Other	*Charges* priced in a way not covered by another pricing category.

3.1.42.3. Column ID

PricingCategory

3.1.42.4. Display Name

Pricing Category

3.1.42.5. Description

Describes the pricing model used for a *charge* at the time of use or purchase.

3.1.42.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	Allowed values

3.1.42.7. Version Introduced

1.0-preview

3.1.43. Pricing Currency

Pricing Currency is the *national* or *virtual currency* denomination that a *resource* or *service* was priced in. Pricing Currency is commonly used in scenarios where different currencies are used for pricing and billing.

3.1.43.1. Requirements

PricingCurrency MUST adhere to the following requirements:

- PricingCurrency MUST be of type String.
- PricingCurrency MUST conform to StringHandling requirements.
- PricingCurrency MUST conform to CurrencyFormat requirements.
- PricingCurrency MUST NOT be null.

3.1.43.2. Column ID

PricingCurrency

3.1.43.3. Display Name

Pricing Currency

3.1.43.4. Description

The *national* or *virtual currency* denomination that a *resource* or *service* was priced in.

3.1.43.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	Currency Format

3.1.43.6. Version Introduced

1.2

3.1.44. Pricing Currency Contracted Unit Price

The Pricing Currency Contracted Unit Price represents the agreed-upon unit price for a singlePricing Unit of the associated *SKU*, inclusive of *negotiated discounts*, if present, while excluding negotiated *commitment discounts* or any other discounts. This price is denominated in the Pricing Currency. When negotiated discounts do not apply to unit prices and instead are applied to exchange rates, the Pricing Currency Contracted Unit Price defaults to the Pricing Currency List Unit Price. The Pricing Currency Contracted Unit Price is commonly used to calculate savings based on negotiation activities.

3.1.44.1. Requirements

PricingCurrencyContractedUnitPrice MUST adhere to the following requirements:

- PricingCurrencyContractedUnitPrice MUST be of type Decimal.
- PricingCurrencyContractedUnitPrice MUST conform to NumericFormat requirements.
- PricingCurrencyContractedUnitPrice MUST adhere to the following nullability requirements:
    - PricingCurrencyContractedUnitPrice MUST be null when SkuPriceId is null.
    - PricingCurrencyContractedUnitPrice MUST be null when ChargeCategory is "Tax".
    - PricingCurrencyContractedUnitPrice MUST NOT be null when SkuPriceId is not null.
    - PricingCurrencyContractedUnitPrice MUST NOT be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
    - PricingCurrencyContractedUnitPrice MAY be null in all other cases.
- When PricingCurrencyContractedUnitPrice is not null, PricingCurrencyContractedUnitPrice MUST adhere to the following requirements:
    - PricingCurrencyContractedUnitPrice MUST be a non-negative decimal value.
    - PricingCurrencyContractedUnitPrice MUST be denominated in the PricingCurrency.

3.1.44.2. Usability Constraints

Aggregation: Column values should only be viewed in the context of their row and not aggregated to produce a total.

3.1.44.3. Column ID

PricingCurrencyContractedUnitPrice

3.1.44.4. Display Name

Pricing Currency Contracted Unit Price

3.1.44.5. Description

The agreed-upon unit price for a single Pricing Unit of the associated SKU, inclusive of *negotiated discounts*, if present, while excluding negotiated *commitment discounts* or any other discounts, and expressed in Pricing Currency.

3.1.44.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True

Constraint	Value
Data type	Decimal
Value format	Numeric Format
Number range	Any valid non-negative decimal value

3.1.44.7. Version Introduced

1.2

3.1.45. Pricing Currency Effective Cost

Pricing Currency Effective Cost represents the Pricing Currency-denominated equivalent of Effective Cost. It reflects the cost of a *charge* based on the *resources* used, *services* used, or *contract commitments* recognized in a given *charge period*.

Because Pricing Currency Effective Cost differs from Effective Cost only in denomination, it follows the same pricing adjustments, amortizations, and exclusions. This column provides practitioners with a standardized baseline, allowing them to view costs in a uniform currency, whether converting from a *virtual currency* to a *national currency* (e.g., tokens to USD) or from one national currency to another (e.g., EUR to USD).

Pricing Currency Effective Cost is commonly used to support FinOps activities, including *accrual-based* reporting, forecasting, and cost allocation when pricing and billing use different currencies.

3.1.45.1. Requirements

PricingCurrencyEffectiveCost MUST adhere to the following requirements:

- PricingCurrencyEffectiveCost MUST be of type Decimal.
- PricingCurrencyEffectiveCost MUST conform to NumericFormat requirements.
- PricingCurrencyEffectiveCost MUST NOT be null.
- PricingCurrencyEffectiveCost MUST be denominated in the PricingCurrency.
- PricingCurrencyEffectiveCost MUST be the PricingCurrency-denominated equivalent of EffectiveCost.

3.1.45.2. Column ID

PricingCurrencyEffectiveCost

3.1.45.3. Display Name

Pricing Currency Effective Cost

3.1.45.4. Description

The PricingCurrency-denominated equivalent of Effective Cost, representing the cost of a *charge* based on the *resources* used, *services* used, or *contract commitments* recognized in a given *charge period*.

3.1.45.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional

Constraint	Value
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.45.6. Version Introduced

1.2

3.1.46. Pricing Currency List Unit Price

The Pricing Currency List Unit Price represents the suggested service-provider-published unit price for a single Pricing Unit of the associated *SKU*, exclusive of any discounts. This price is denominated in the Pricing Currency. The Pricing Currency List Unit Price is commonly used for calculating savings based on various rate optimization activities.

3.1.46.1. Requirements

PricingCurrencyListUnitPrice MUST adhere to the following requirements:

- PricingCurrencyListUnitPrice MUST be of type Decimal.
- PricingCurrencyListUnitPrice MUST conform to NumericFormat requirements.
- PricingCurrencyListUnitPrice MUST adhere to the following nullability requirements:
  - PricingCurrencyListUnitPrice MUST be null when SkuPriceId is null.
  - PricingCurrencyListUnitPrice MUST be null when ChargeCategory is "Tax".
  - PricingCurrencyListUnitPrice MUST NOT be null when SkuPriceId is not null.
  - PricingCurrencyListUnitPrice MUST NOT be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
  - PricingCurrencyListUnitPrice MAY be null in all other cases.
- When PricingCurrencyListUnitPrice is not null, PricingCurrencyListUnitPrice MUST adhere to the following requirements:
  - PricingCurrencyListUnitPrice MUST be a non-negative decimal value.
  - PricingCurrencyListUnitPrice MUST be denominated in the PricingCurrency.

3.1.46.2. Usability Constraints

Aggregation: Column values should only be viewed in the context of their row and not aggregated to produce a total.

3.1.46.3. Column ID

PricingCurrencyListUnitPrice

3.1.46.4. Display Name

Pricing Currency List Unit Price

3.1.46.5. Description

The suggested service-provider-published unit price for a single Pricing Unit of the associated *SKU*, exclusive of any discounts and expressed in Pricing Currency.

3.1.46.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid non-negative decimal value

3.1.46.7. Version Introduced

1.2

3.1.47. Pricing Quantity

The Pricing Quantity represents the volume of a given *SKU* associated with a *resource* or *service* used or purchased, based on the Pricing Unit. Distinct from Consumed Quantity (complementary to Consumed Unit), it focuses on pricing and cost, not *resource* and *service* consumption.

3.1.47.1. Requirements

PricingQuantity MUST adhere to the following requirements:

- PricingQuantity MUST be of type Decimal.
- PricingQuantity MUST conform to NumericFormat requirements.
- PricingQuantity MUST adhere to the following nullability requirements:
  - PricingQuantity MUST be null when SkuPriceId is null.
  - PricingQuantity MUST be null when ChargeCategory is "Tax".
  - PricingQuantity MUST NOT be null when ChargeCategory is "Usage" or "Purchase" andChargeClass is not "Correction".
  - PricingQuantity MAY be null in all other cases.
- Cost metric (e.g., ContractedCost) MUST equal the product of the corresponding unit price (e.g.,ContractedUnitPrice) and PricingQuantity when the unit price is not null and PricingQuantity is not null.

3.1.47.2. Usability Constraints

Aggregation: When aggregating Pricing Quantity for commitment utilization calculations, it's important to exclude *commitment discount* purchases (i.e., when Charge Category is "Purchase") that are paid to cover future eligible*charges* (e.g., *commitment discount*). Otherwise, when accounting for all upfront or accrued purchases, it's important to exclude *commitment discount* usage (i.e., when Charge Category is "Usage"). This exclusion helps prevent double counting of these quantities in the aggregation.

3.1.47.3. Column ID

PricingQuantity

3.1.47.4. Display Name

Pricing Quantity

3.1.47.5. Description

The volume of a given *SKU* associated with a *resource* or *service* used or purchased, based on the Pricing Unit.

3.1.47.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Metric
Feature level	Mandatory
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.1.47.7. Version Introduced

1.0-preview

3.1.48. Pricing Unit

The Pricing Unit represents a service-provider-specified measurement unit for determining unit prices, indicating how the service provider rates measured usage and purchase quantities after applying pricing rules like *block pricing*. Common examples include the number of hours for compute appliance runtime (e.g., `Hours`), gigabyte-hours for a storage appliance (e.g., `GB-Hours`), or an accumulated count of requests for a network appliance or API service (e.g., `1000 Requests`). Pricing Unit complements the Pricing Quantity metric. Distinct from the Consumed Unit, it focuses on pricing and cost, not *resource* and *service* consumption, often at a coarser granularity.

3.1.48.1. Requirements

PricingUnit MUST adhere to the following requirements:

- PricingUnit MUST be of type String.
- PricingUnit MUST conform to StringHandling requirements.
- PricingUnit SHOULD conform to UnitFormat requirements.
- PricingUnit MUST adhere to the following nullability requirements:
  - PricingUnit MUST be null when PricingQuantity is null.
  - PricingUnit MUST NOT be null when PricingQuantity is not null.
- When PricingUnit is not null, PricingUnit MUST adhere to the following requirements:
  - PricingUnit MUST be semantically equal to the corresponding pricing measurement unit provided in service-provider-published *price list*.
  - PricingUnit MUST be semantically equal to the corresponding pricing measurement unit provided in invoice, when the invoice includes a pricing measurement unit.

3.1.48.2. Column ID

PricingUnit

3.1.48.3. Display Name

Pricing Unit

3.1.48.4. Description

Service-provider-specified measurement unit for determining unit prices, indicating how the service provider rates measured usage and purchase quantities after applying pricing rules like *block pricing*.

3.1.48.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	Unit Format

3.1.48.6. Version Introduced

1.0-preview

3.1.49. Region ID

A Region ID is a host-provider-assigned identifier for an isolated geographic area where a *resource* is provisioned or a *service* is provided. The region is commonly used for scenarios like analyzing cost and unit prices based on where *resources* are deployed.

3.1.49.1. Requirements

RegionId MUST adhere to the following requirements:

- RegionId MUST be of type String.
- RegionId MUST conform to StringHandling requirements.
- RegionId MUST adhere to the following nullability requirements:
  - RegionId MUST NOT be null when a *resource* or *service* is operated in or managed from a distinct region.
  - RegionId MAY be null when a *resource* or *service* is not operated in or managed from a distinct region.

3.1.49.2. Column ID

RegionId

3.1.49.3. Display Name

Region ID

3.1.49.4. Description

Host-provider-assigned identifier for an isolated geographic area where a *resource* is provisioned or a *service* is provided.

3.1.49.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.49.6. Version Introduced

1.0

3.1.50. Region Name

Region Name is a host-provider-assigned display name for an isolated geographic area where a *resource* is provisioned or a *service* is provided. Region Name is commonly used for scenarios like analyzing cost and unit prices based on where *resources* are deployed.

3.1.50.1. Requirements

RegionName MUST adhere to the following requirements:

- RegionName MUST be of type String.
- RegionName MUST conform to StringHandling requirements.
- RegionName MUST adhere to the following nullability requirements:
  - RegionName MUST be null when RegionId is null.
  - RegionName MUST NOT be null when RegionId is not null.

3.1.50.2. Column ID

RegionName

3.1.50.3. Display Name

Region Name

3.1.50.4. Description

The name of an isolated geographic area where a *resource* is provisioned or a *service* is provided.

3.1.50.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.50.6. Version Introduced

1.0

3.1.51. Resource ID

A Resource ID is an identifier assigned to a *resource* by the service provider. The Resource ID is commonly used for cost reporting, analysis, and allocation scenarios.

3.1.51.1. Requirements

ResourceId MUST adhere to the following requirements:

- ResourceId MUST be of type String.
- ResourceId MUST conform to StringHandling requirements.
- ResourceId MUST adhere to the following nullability requirements:
  - ResourceId MUST be null when a *charge* is not related to a *resource*.
  - ResourceId MUST NOT be null when a *charge* is related to a *resource*.
- When ResourceId is not null, ResourceId MUST adhere to the following requirements:
  - ResourceId MUST be a unique identifier within the service provider.
  - ResourceId SHOULD be a fully-qualified identifier.
  - ResourceId MUST be the identifier of the *resource* that received the *commitment discount* when CommitmentDiscountStatus is "Used".

3.1.51.2. Column ID

ResourceId

3.1.51.3. Display Name

Resource ID

3.1.51.4. Description

Identifier assigned to a *resource* by the service provider.

3.1.51.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.51.6. Version Introduced

0.5

3.1.52. Resource Name

The Resource Name is a display name assigned to a *resource*. It is commonly used for cost analysis, reporting, and allocation scenarios.

3.1.52.1. Requirements

ResourceName MUST adhere to the following requirements:

- ResourceName MUST be of type String.
- ResourceName MUST conform to StringHandling requirements.
- ResourceName MUST adhere to the following nullability requirements:
  - ResourceName MUST be null when ResourceId is null or when the *resource* does not have an assigned display name.
  - ResourceName MUST NOT be null when ResourceId is not null and the *resource* has an assigned display name.
- ResourceName MUST NOT duplicate ResourceId when the *resource* is not provisioned interactively or only has a system-generated ResourceId.

3.1.52.2. Column ID

ResourceName

3.1.52.3. Display Name

Resource Name

3.1.52.4. Description

Display name assigned to a *resource*.

3.1.52.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.52.6. Version Introduced

0.5

3.1.53. Resource Type

Resource Type describes the kind of *resource* the *charge* applies to. A Resource Type is commonly used for scenarios like identifying cost changes in groups of similar *resources* and may include values like Virtual Machine, Data Warehouse, and Load Balancer.

3.1.53.1. Requirements

ResourceType MUST adhere to the following requirements:

- ResourceType MUST be of type String.
- ResourceType MUST conform to StringHandling requirements.
- ResourceType MUST adhere to the following nullability requirements:
  - ResourceType MUST be null when ResourceId is null.
  - ResourceType MUST NOT be null when ResourceId is not null.

3.1.53.2. Column ID

ResourceType

3.1.53.3. Display Name

Resource Type

3.1.53.4. Description

The kind of *resource* the *charge* applies to.

3.1.53.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.53.6. Version Introduced

1.0-preview

3.1.54. Service Provider Name

Service Provider Name is the name of the entity that provides the *resources* or *services* available for usage or purchase. These services can be built on top of infrastructure provided by a Host Provider; offered as fully integrated solutions; or include complementary offerings such as support, licensing, or consulting. It is commonly used for cost analysis and reporting scenarios.

Notes:

- In marketplace scenarios, the Service Provider represents the seller rather than the marketplace operator, as the marketplace operator merely provides a purchasing mechanism and does not itself provide the *resources* or *services* available for usage or purchase.
- In reseller scenarios, if the reseller is selling resource or services that are white-labeled from another provider, the Service Provider is the reseller.
- In all other scenarios, the Service Provider is the entity that produced the resources or services.

3.1.54.1. Requirements

ServiceProviderName MUST adhere to the following requirements:

- ServiceProviderName MUST be of type String.
- ServiceProviderName MUST conform to StringHandling requirements.
- ServiceProviderName MUST NOT be null.

See Appendix: Participating Entity Identification Examples section for examples of Service Provider Name values across various use case scenarios.

3.1.54.2. Column ID

ServiceProviderName

3.1.54.3. Display Name

Service Provider Name

3.1.54.4. Description

The name of the entity that made the *resources* or *services* available for purchase or consumption.

3.1.54.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.1.54.6. Version Introduced

1.3 Introduced as a replacement for the removed ProviderName column

3.1.55. Service Category

The Service Category is the highest-level classification of a *service* based on the core function of the *service*. Each *service* should have one and only one category that best aligns with its primary purpose. The Service Category is commonly used for scenarios like analyzing costs across service providers and tracking the migration of workloads across fundamentally different architectures.

3.1.55.1. Requirements

ServiceCategory MUST adhere to the following requirements:

- ServiceCategory MUST be of type String.
- ServiceCategory MUST NOT be null.
- ServiceCategory MUST be one of the allowed values.

3.1.55.2. Allowed Values

Service Category	Description

Service Category	Description
AI and Machine Learning	Artificial Intelligence and Machine Learning related technologies.
Analytics	Data processing, analytics, and visualization capabilities.
Business Applications	Business and productivity applications and services.
Compute	Virtual, containerized, serverless, or high-performance computing infrastructure and services.
Databases	Database platforms and services that allow for storage and querying of data.
Developer Tools	Software development and delivery tools and services.
Multicloud	Support for interworking of multiple cloud and/or on-premises environments.
Identity	Identity and access management services.
Integration	Services that allow applications to interact with one another.
Internet of Things	Development and management of IoT devices and networks.
Management and Governance	Management, logging, and observability of a customer's use of cloud.
Media	Media and entertainment streaming and processing services.
Migration	Moving applications and data to the cloud.
Mobile	Services enabling cloud applications to interact via mobile technologies.
Networking	Network connectivity and management.
Security	Security monitoring and compliance services.
Storage	Storage services for structured or unstructured data.
Web	Services enabling cloud applications to interact via the Internet.
Other	New or emerging services that do not align with an existing category.

3.1.55.3. Column ID

ServiceCategory

3.1.55.4. Display Name

Service Category

3.1.55.5. Description

Highest-level classification of a *service* based on the core function of the *service*.

3.1.55.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.1.55.7. Version Introduced

0.5

3.1.56. Service Name

A *service* represents an offering that can be purchased from a service provider (e.g., cloud virtual machine, SaaS database, professional services from a systems integrator). A *service* offering can include various types of usage or other *charges*. For example, a cloud database *service* may include compute, storage, and networking *charges*.

The Service Name is a display name for the offering that was purchased. The Service Name is commonly used for scenarios like analyzing aggregate cost trends over time and filtering data to investigate anomalies.

3.1.56.1. Requirements

ServiceName MUST adhere to the following requirements:

- ServiceName MUST be of type String.
- ServiceName MUST conform to StringHandling requirements.
- ServiceName MUST NOT be null.
- The relationship between ServiceName and ServiceCategory MUST adhere to the following requirements:
  - ServiceName MUST have one and only one ServiceCategory that best aligns with its primary purpose, except when no suitable ServiceCategory is available.
  - ServiceName MUST be associated with the ServiceCategory "Other" when no suitable ServiceCategory is available.
- The relationship between ServiceName and ServiceSubcategory MUST adhere to the following requirements:
  - ServiceName SHOULD have one and only one ServiceSubcategory that best aligns with its primary purpose, except when no suitable ServiceSubcategory is available.
  - ServiceName SHOULD be associated with the ServiceSubcategory "Other" when no suitable ServiceSubcategory is available.

3.1.56.2. Column ID

ServiceName

3.1.56.3. Display Name

Service Name

3.1.56.4. Description

An offering that can be purchased from a service provider (e.g., cloud virtual machine, SaaS database, professional *services* from a systems integrator).

3.1.56.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.1.56.6. Version Introduced

0.5

3.1.57. Service Subcategory

The Service Subcategory is a secondary classification of the Service Category for a *service* based on its core function. The Service Subcategory (in conjunction with the Service Category) is commonly used for scenarios like analyzing spend and usage for specific workload types across service providers and tracking the migration of workloads across fundamentally different architectures.

3.1.57.1. Requirements

ServiceSubcategory MUST adhere to the following requirements:

- ServiceSubcategory MUST be of type String.
- ServiceSubcategory MUST NOT be null.
- ServiceSubcategory MUST be one of the allowed values.
- ServiceSubcategory MUST have one and only one parent ServiceCategory as specified in the allowed values below.

3.1.57.2. Allowed Values

Service Category	Service Subcategory	Service Subcategory Description
AI and Machine Learning	AI Platforms	Unified solution that combines artificial intelligence and machine learning technologies.
AI and Machine Learning	Bots	Automated performance of tasks such as customer service, data collection, and content moderation.
AI and Machine Learning	Generative AI	Creation of content like text, images, and music by learning patterns from existing data.
AI and Machine Learning	Machine Learning	Creation, training, and deployment of statistical algorithms that learn from and perform tasks based on data.
AI and Machine Learning	Natural Language Processing	Generation of human language, handling tasks like translation, sentiment analysis, and text summarization.
AI and Machine Learning	Other (AI and Machine Learning)	AI and Machine Learning services that do not fall into one of the defined subcategories.
Analytics	Analytics Platforms	Unified solution that combines technologies across the entire analytics lifecycle.
Analytics	Business Intelligence	Semantic models, dashboards, reports, and data visualizations to track performance and identify trends.
Analytics	Data Processing	Integration and transformation tasks to prepare data for analysis.
Analytics	Search	Discovery of information by indexing and retrieving data from various sources.
Analytics	Streaming Analytics	Real-time data stream processes to detect patterns, trends, and anomalies as they occur.
Analytics	Other (Analytics)	Analytics services that do not fall into one of the defined subcategories.
Business Applications	Productivity and Collaboration	Tools that facilitate individuals managing tasks and working together.
Business Applications	Other (Business Applications)	Business Applications services that do not fall into one of the defined subcategories.
Compute	Containers	Management and orchestration of containerized compute platforms.
Compute	End User Computing	Virtualized desktop infrastructure and device / endpoint management.
Compute	Quantum Compute	Resources and simulators that leverage the principles of quantum mechanics.
Compute	Serverless Compute	Enablement of compute capabilities without provisioning or managing servers.
Compute	Virtual Machines	Computing environments ranging from hosts with abstracted operating systems to bare-metal servers.
Compute	Other (Compute)	Compute services that do not fall into one of the defined subcategories.
Databases	Caching	Low-latency and high-throughput access to frequently accessed data.

Service Category	Service Subcategory	Service Subcategory Description
Databases	Data Warehouses	Big data storage and querying capabilities.
Databases	Ledger Databases	Immutable and transparent databases to record tamper-proof and cryptographically secure transactions.
Databases	NoSQL Databases	Unstructured or semi-structured data storage and querying capabilities.
Databases	Relational Databases	Structured data storage and querying capabilities.
Databases	Time Series Databases	Time-stamped data storage and querying capabilities.
Databases	Other (Databases)	Database services that do not fall into one of the defined subcategories.
Developer Tools	Developer Platforms	Unified solution that combines technologies across multiple areas of the software development lifecycle.
Developer Tools	Continuous Integration and Deployment	CI/CD tools and services that support building and deploying code for software and systems.
Developer Tools	Development Environments	Tools and services that support authoring code for software and systems.
Developer Tools	Source Code Management	Tools and services that support version control of code for software and systems.
Developer Tools	Quality Assurance	Tools and services that support testing code for software and systems.
Developer Tools	Other (Developer Tools)	Developer Tools services that do not fall into one of the defined subcategories.
Identity	Identity and Access Management	Technologies that ensure users have appropriate access to resources.
Identity	Other (Identity)	Identity services that do not fall into one of the defined subcategories.
Integration	API Management	Creation, publishing, and management of application programming interfaces.
Integration	Messaging	Asynchronous communication between distributed applications.
Integration	Workflow Orchestration	Design, execution, and management of business processes and workflows.
Integration	Other (Integration)	Integration services that do not fall into one of the defined subcategories.
Internet of Things	IoT Analytics	Examination of data collected from IoT devices.
Internet of Things	IoT Platforms	Unified solution that combines IoT data collection, processing, visualization, and device management.
Internet of Things	Other (Internet of Things)	Internet of Things (IoT) services that do not fall into one of the defined subcategories.
Management and Governance	Architecture	Planning, design, and construction of software systems.
Management and Governance	Compliance	Adherence to regulatory standards and industry best practices.
Management and Governance	Cost Management	Monitoring and controlling expenses of systems and services.
Management and Governance	Data Governance	Management of the availability, usability, integrity, and security of data.
Management and Governance	Disaster Recovery	Plans and procedures that ensure systems and services can recover from disruptions.
Management and Governance	Endpoint Management	Tools that configure and secure access to devices.
Management and Governance	Observability	Monitoring, logging, and tracing of data to track the performance and health of systems.
Management and Governance	Support	Assistance and expertise supplied by service providers.
Management and Governance	Other (Management and Governance)	Management and governance services that do not fall into one of the defined subcategories.
Media	Content Creation	Production of media content.
Media	Gaming	Development and delivery of gaming services.
Media	Media Streaming	Multimedia delivered and rendered in real-time on devices.

Service Category	Service Subcategory	Service Subcategory Description
Media	Mixed Reality	Technologies that blend real-world and computer-generated environments.
Media	Other (Media)	Media services that do not fall into one of the defined subcategories.
Migration	Data Migration	Movement of stored data from one location to another.
Migration	Resource Migration	Movement of resources from one location to another.
Migration	Other (Migration)	Migration services that do not fall into one of the defined subcategories.
Mobile	Other (Mobile)	All Mobile services.
Multicloud	Multicloud Integration	Environments that facilitate consumption of services from multiple cloud service providers.
Multicloud	Other (Multicloud)	Multicloud services that do not fall into one of the defined subcategories.
Networking	Application Networking	Distribution of incoming network traffic across application-based workloads.
Networking	Content Delivery	Distribution of digital content using a network of servers (CDNs).
Networking	Network Connectivity	Facilitates communication between networks or network segments.
Networking	Network Infrastructure	Configuration, monitoring, and troubleshooting of network devices.
Networking	Network Routing	Services that select paths for traffic within or across networks.
Networking	Network Security	Protection from unauthorized network access and cyber threats using firewalls and anti-malware tools.
Networking	Other (Networking)	Networking services that do not fall into one of the defined subcategories.
Security	Secret Management	Information used to authenticate users and systems, including secrets, certificates, tokens, and other keys.
Security	Security Posture Management	Tools that help organizations configure, monitor, and improve system security.
Security	Threat Detection and Response	Collect and analyze security data to identify and respond to potential security threats and vulnerabilities.
Security	Other (Security)	Security services that do not fall into one of the defined subcategories.
Storage	Backup Storage	Secondary storage to protect against data loss.
Storage	Block Storage	High performance, low latency storage that provides random access.
Storage	File Storage	Scalable, sharable storage for file-based data.
Storage	Object Storage	Highly available, durable storage for unstructured data.
Storage	Storage Platforms	Unified solution that supports multiple storage types.
Storage	Other (Storage)	Storage services that do not fall into one of the defined subcategories.
Web	Application Platforms	Integrated environments that run web applications.
Web	Other (Web)	Web services that do not fall into one of the defined subcategories.
Other	Other (Other)	Services that do not fall into one of the defined categories.

3.1.57.3. Column ID

ServiceSubcategory

3.1.57.4. Display Name

Service Subcategory

3.1.57.5. Description

Secondary classification of the Service Category for a *service* based on its core function.

3.1.57.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Recommended
Allows nulls	False
Data type	String
Value format	Allowed values

3.1.57.7. Version Introduced

1.1

3.1.58. SKU ID

A SKU ID is a service-provider-specified unique identifier that represents a specific *SKU*. *SKUs* are quantifiable goods or service offerings in a Cost and Usage *FOCUS dataset* that represent specific functionality and technical specifications. Examples of *SKUs* include but are not limited to:

- A product license that is purchased or subscribed to.
- Usage of a deployed resource from direct user interaction (e.g., request count).
- Usage by a deployed resource based on the resource's configuration (e.g., running hours, storage space).

Each SKU ID represents a unique set of features that can be sold at different price points or *SKU Prices*. SKU ID is consistent across all pricing variations, which may differ based on multiple factors beyond the common functionality and technical specifications. Examples include but are not limited to:

- Date the *charge* was incurred.
- Pricing tiers (e.g., free tier or volume-based tiers).
- Commitment discount pricing *period* (e.g., 1 year, 3 years).
- Negotiated discounts or other contractual terms or conditions.

SKU ID should be consistent across pricing variations of a good or service to facilitate price comparisons for the same functionality, like where the functionality is provided or how it's paid for. SKU ID can be referenced on a catalog or *price list* published by a service provider to look up detailed information about the *SKU*. The composition of the properties associated with the SKU ID may differ across service providers. SKU ID is commonly used for analyzing and comparing costs for the same SKU across different price details (e.g., *period*, tier, location).

3.1.58.1. Requirements

SkuId MUST adhere to the following requirements:

- SkuId MUST be of type String.
- SkuId MUST conform to StringHandling requirements.
- SkuId MUST adhere to the following nullability requirements:
  - SkuId MUST be null when ChargeCategory is "Tax".
  - SkuId MUST NOT be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
  - SkuId MAY be null in all other cases.
- SkuId for a given *SKU* MUST adhere to the following requirements:
  - SkuId MUST remain consistent across *billing accounts* or contracts.
  - SkuId MUST remain consistent across PricingCategory values.
  - SkuId MUST remain consistent regardless of any other factors that might impact the price but do not affect the functionality of the *SKU*.
- SkuId MUST be associated with a given *resource* or *service* when ChargeCategory is "Usage" or "Purchase".
- SkuId MAY match SkuPriceId.

3.1.58.2. Column ID

SkuId

3.1.58.3. Display Name

SKU ID

3.1.58.4. Description

Service-provider-specified unique identifier that represents a specific *SKU* (e.g., a quantifiable good or service offering).

3.1.58.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.58.6. Version Introduced

1.0-preview

3.1.59. SKU Meter

SKU Meter describes the functionality being metered or measured by a particular SKU in a *charge*.

Service providers often have billing models in which multiple SKUs exist for a given service to describe and bill for different functionalities for that service. For example, an object storage service may have separate SKUs for functionalities such as object storage, API requests, data transfer, encryption, and object management. This field helps practitioners understand which functionalities are being metered by the different SKUs that appear in a Cost and Usage *FOCUS dataset*.

3.1.59.1. Requirements

SkuMeter MUST adhere to the following requirements:

- SkuMeter MUST be of type String.
- SkuMeter MUST conform to StringHandling requirements.
- SkuMeter MUST adhere to the following nullability requirements:
  - SkuMeter MUST be null when SkuId is null.
  - SkuMeter SHOULD NOT be null when SkuId is not null.
- SkuMeter SHOULD remain consistent over time for a given SkuId.

3.1.59.2. Examples

Compute Usage, Block Volume Usage, Data Transfer, API Requests

3.1.59.3. Column ID

SkuMeter

3.1.59.4. Display Name

SKU Meter

3.1.59.5. Description

Describes the functionality being metered or measured by a particular SKU in a *charge*.

3.1.59.6. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.59.7. Version Introduced

1.1

3.1.60. SKU Price Details

SKU Price Details represent a list of *SKU Price* properties (key-value pairs) associated with a specific SKU Price ID. These properties include qualitative and quantitative properties of a *SKUs* (e.g., functionality and technical specifications), along with core stable pricing properties (e.g., pricing *periods*, tiers, etc.), excluding dynamic or negotiable pricing elements such as unit price amounts; currency (and related exchange rates); temporal validity (e.g., effective dates); and contract- or negotiation-specific factors (e.g., contract or account identifiers, and negotiable discounts).

The composition of properties associated with a specific *SKU Price* may differ across service providers and across *SKUs* within the same service provider. However, the exclusion of dynamic or negotiable pricing properties should ensure that all *charges* with the same SKU Price ID share the same SKU Price Details, i.e., that SKU Price Details remains consistent across different *billing periods* and *billing accounts* within a service provider.

SKU Price Details helps practitioners understand and distinguish *SKU Prices*, each identified by a SKU Price ID and associated with a used or purchased *resource* or *service*. It can also help determine the quantity of units for a property when it holds a numeric value (e.g., CoreCount), even when its unit differs from the one in which the *SKU* is priced and charged, thus supporting FinOps capabilities such as unit economics. Additionally, the SKU Price Details may be used to analyze costs based on pricing properties such as *periods* and tiers.

3.1.60.1. Requirements

SkuPriceDetails MUST adhere to the following requirements:

- SkuPriceDetails MUST be of type JSON Object (serialized as a String where necessary).
- SkuPriceDetails MUST conform to StringHandling requirements.
- SkuPriceDetails MUST conform to KeyValueFormat requirements.
- SkuPriceDetails property keys SHOULD conform to PascalCase format.
- SkuPriceDetails MUST adhere to the following nullability requirements:
  - SkuPriceDetails MUST be null when SkuPriceId is null.
  - SkuPriceDetails MAY be null when SkuPriceId is not null.
- When SkuPriceDetails is not null, SkuPriceDetails MUST adhere to the following requirements:
  - SkuPriceDetails MUST be associated with a given SkuPriceId.
  - SkuPriceDetails MUST include the FOCUS-defined SKU Price property when an equivalent property is included as a

custom property.
    - SkuPriceDetails MUST NOT include properties that are not applicable to the corresponding SkuPriceId.
    - SkuPriceDetails SHOULD include all FOCUS-defined SKU Price properties listed below that are applicable to the corresponding SkuPriceId.
    - SkuPriceDetails SHOULD include all custom SKU Price properties that are applicable to the corresponding SkuPriceId when there is no equivalent FOCUS-defined property.
    - SkuPriceDetails MAY include properties that are already captured in other dedicated columns.
    - SkuPriceDetails properties for a given SkuPriceId MUST adhere to the following requirements:
        - Existing SkuPriceDetails properties SHOULD remain consistent over time.
        - Existing SkuPriceDetails properties SHOULD NOT be removed.
        - Additional SkuPriceDetails properties MAY be added over time.
    - Property key SHOULD remain consistent across comparable *SKUs* having that property, and the values for this key SHOULD remain in a consistent format.
    - Property key MUST begin with the string "x_" unless it is a FOCUS-defined property.
    - Property value MUST represent the value for a singlePricingUnit when the property holds a numeric value.
- FOCUS-defined SKU Price properties MUST adhere to the following requirements:
    - Property key MUST match the spelling and casing specified for the FOCUS-defined property.
    - Property value MUST be of the type specified for that property.
    - Property value MUST represent the value for a single PricingUnit, denominated in the unit of measure specified for that property when the property holds a numeric value.

3.1.60.2. FOCUS-Defined Properties

The following keys should be used when applicable to facilitate cross-SKU and cross-service-provider queries for the same conceptual property. FOCUS-defined keys will appear in the list below and custom (e.g., service-provider-defined) keys will be prefixed with "x_" to make them easy to identify as well as prevent collisions.

Key	Description	Data Type	Unit of Measure (numeric) or example values (string)
CoreCount	Number of physical or virtual CPUs available[1]	Numeric	Measure: Quantity of Cores
DiskMaxIops	Storage maximum sustained input/output operations per second[1]	Numeric	Measure: Input/Output Operations per Second (IOPS)
DiskSpace	Storage capacity available	Numeric	Measure: Gibibytes (GiB)
DiskType	Kind of disk used	String	Examples: "SSD", "HDD", "NVMe"
GpuCount	Number of GPUs available	Numeric	Measure: Quantity of GPUs
InstanceType	Common name of the instance including size, shape, series, etc.	String	Examples: "m5d.2xlarge", "NC24rs_v3", "P50"
InstanceSeries	Common name for the series and/or generation of the instance	String	Examples: "M5", "Dadv5", "N2D"
MemorySize	RAM allocated for processing	Numeric	Measure: Gibibytes (GiB[2])
NetworkMaxIops	Network maximum sustained input/output operations per second[1]	Numeric	Measure: Input/Output Operations per Second (IOPS)
NetworkMaxThroughput	Network maximum sustained throughput for data transfer[1]	Numeric	Measure: Megabits per second (Mbps)
OperatingSystem	Operating system family[3]	String	Examples: "Linux", "MacOS", "Windows"
Redundancy	Level of redundancy offered by the SKU	String	Examples: "Local", "Zonal", "Global"
StorageClass	Class or tier of storage provided	String	Examples: "Hot", "Archive", "Nearline"

Notes

[1] In the case of "burstable" SKUs offering variable levels of performance, the baseline or guaranteed value should be used.

[2] Memory manufacturers still commonly uses "GB" to refer to 2^{30} bytes, which is known as GiB in other contexts.

[3] This is the operating system family of the SKU, if it's included with the SKU or the SKU only supports one type of operating system.

3.1.60.3. Examples

```
{
  "StorageClass": "Archive",
  "CoreCount": 4,
  "x_PremiumProcessing": true
}
```

3.1.60.4. Column ID

SkuPriceDetails

3.1.60.5. Display Name

SKU Price Details

3.1.60.6. Description

A set of properties of a SKU Price ID which are meaningful and common to all instances of that SKU Price ID.

3.1.60.7. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	JSON
Value format	Key-Value Format

3.1.60.8. Version Introduced

1.1

3.1.61. SKU Price ID

SKU Price ID is a service-provider-specified unique identifier that represents a specific *SKU Price* associated with a *resource* or *service* used or purchased. It serves as a key reference for a *SKU Price* in a *price list* published by a service provider, allowing practitioners to look up detailed information about the *SKU Price*.

The composition of properties associated with the SKU Price ID may differ across service providers and across *SKUs* within the same service provider. However, the exclusion of dynamic or negotiable pricing properties - such as unit price amount; currency (and related exchange rates); temporal validity (e.g., effective dates); and contract- or negotiation-specific elements (e.g., contract or account identifiers, and negotiable discounts) - ensures that the SKU Price ID remains consistent across different billing periods and billing accounts within a service provider. This consistency enables efficient filtering of *charges* to track price fluctuations (e.g., changes in unit price amounts) over time and across billing accounts, for both list and contracted unit prices. Additionally, the SKU Price ID is commonly used to analyze costs based on pricing properties such as *periods* and tiers.

3.1.61.1. Requirements

SkuPriceId MUST adhere to the following requirements:

- SkuPriceId MUST be of type String.
- SkuPriceId MUST conform to StringHandling requirements.
- SkuPriceId MUST adhere to the following nullability requirements:
  - SkuPriceId MUST be null when ChargeCategory is "Tax".
  - SkuPriceId MUST NOT be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
  - SkuPriceId MAY be null in all other cases.
- When SkuPriceId is not null, SkuPriceId MUST adhere to the following requirements:
  - SkuPriceId MUST have one and only one parent SkuId.
  - SkuPriceId MUST remain consistent over time.
  - SkuPriceId MUST remain consistent across *billing accounts* or contracts.
  - SkuPriceId MAY match SkuId.
  - SkuPriceId MUST be associated with a given *resource* or *service* when ChargeCategory is "Usage" or "Purchase".
  - SkuPriceId MUST reference a *SKU Price* in a service-provider-supplied *price list*, enabling the lookup of detailed information about the *SKU Price*.
  - SkuPriceId MUST be a valid reference to the ListUnitPrice when the service provider publishes unit prices exclusive of discounts.
  - SkuPriceId MUST be a valid reference to the ContractedUnitPrice when the service provider supports negotiated pricing concepts.

See Examples: Commitment Discount Flexibility for more details around *commitment discount flexibility*.

3.1.61.2. Column ID

SkuPriceId

3.1.61.3. Display Name

SKU Price ID

3.1.61.4. Description

A service-provider-specified unique identifier that represents a specific *SKU Price* associated with a *resource* or *service* used or purchased.

3.1.61.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.61.6. Version Introduced

1.0-preview

3.1.62. Sub Account ID

A Sub Account ID is a service-provider-assigned identifier assigned to a *sub account*. Sub Account ID is commonly used for scenarios like grouping based on organizational constructs, access management needs, and cost allocation strategies.

3.1.62.1. Requirements

SubAccountId MUST adhere to the following requirements:

- SubAccountId MUST be of type String.
- SubAccountId MUST conform to StringHandling requirements.
- SubAccountId MUST adhere to the following nullability requirements:
    - SubAccountId MUST be null when a *charge* is not related to a *sub account*.
    - SubAccountId MUST NOT be null when a *charge* is related to a *sub account*.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.1.62.2. Column ID

SubAccountId

3.1.62.3. Display Name

Sub Account ID

3.1.62.4. Description

An ID assigned to a grouping of *resources* or *services*, often used to manage access and/or cost.

3.1.62.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.62.6. Version Introduced

0.5

3.1.63. Sub Account Name

A Sub Account Name is a display name assigned to a *sub account*. Sub account Name is commonly used for scenarios like grouping based on organizational constructs, access management needs, and cost allocation strategies.

3.1.63.1. Requirements

SubAccountName MUST adhere to the following requirements:

- SubAccountName MUST be of type String.
- SubAccountName MUST conform to StringHandling requirements.
- SubAccountName MUST adhere to the following nullability requirements:
    - SubAccountName MUST be null when SubAccountId is null.
    - SubAccountName MUST NOT be null when SubAccountId is not null.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.1.63.2. Column ID

SubAccountName

3.1.63.3. Display Name

Sub Account Name

3.1.63.4. Description

A name assigned to a grouping of *resources* or *services*, often used to manage access and/or cost.

3.1.63.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.63.6. Version Introduced

0.5

3.1.64. Sub Account Type

Sub Account Type is a service-provider-assigned name to identify the type of *sub account*. Sub Account Type is a readable display name and not a code. Sub Account Type is commonly used for scenarios like mapping FOCUS and service provider constructs, summarizing costs across service providers, or invoicing and chargeback.

3.1.64.1. Requirements

SubAccountType MUST adhere to the following requirements:

- SubAccountType MUST be of type String.
- SubAccountType MUST conform to StringHandling requirements.
- SubAccountType MUST adhere to the following nullability requirements:
  - SubAccountType MUST be null when SubAccountId is null.
  - SubAccountType MUST NOT be null when SubAccountId is not null.
- SubAccountType MUST be a consistent, readable display value.

3.1.64.2. Column ID

SubAccountType

3.1.64.3. Display Name

Sub Account Type

3.1.64.4. Description

A service-provider-assigned name to identify the type of *sub account*.

3.1.64.5. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.1.64.6. Version Introduced

1.2

3.1.65. Tags

The Tags column represents the set of *tags* assigned to *tag sources* that also account for potential provider-defined or user-defined tag evaluations. Tags are commonly used for scenarios like adding business context to cost and usage data to identify and accurately allocate *charges*. Tags may also be referred to by data generators using other terms such as labels.

A tag becomes *finalized* when a single value is selected from a set of possible tag values assigned to the tag key. When supported by a data generator, this can occur when a tag value is set by provider-defined or user-defined rules.

3.1.65.1. Requirements

Tags MUST adhere to the following requirements:

- Tags MUST be of type JSON Object (serialized as a String where necessary).
- Tags MUST conform to StringHandling requirements.
- Tags MUST conform to KeyValueFormat requirements.
- Tags MAY be null.
- When Tags is not null, Tags MUST adhere to the following requirements:
  - Tags MUST include all user-defined and provider-defined tags.
  - Tags MUST only include finalized tags.
  - Tags SHOULD include tag keys with corresponding non-null values for a given *resource*.
  - Tags MAY include tag keys with a null value for a given *resource* depending on the data generator's tag finalization process.
  - Tag keys that do not support corresponding values, MUST have a corresponding true (boolean) value set.
  - Tag values MUST match the provided values unless true (boolean) is applied to valueless tags.
- Provider-defined tags MUST adhere to the following requirements:
  - Provider-defined tag keys MUST be prefixed with a predetermined, provider-specified tag key prefix that is unique to each corresponding provider-specified *tag scheme*.
  - Provider-specified tag key prefixes SHOULD be publicly documented.
- User-defined tags MUST adhere to the following requirements:
  - User-defined tag keys in all but one user-defined *tag scheme* MUST include a predetermined, provider-specified tag key prefix that is unique to each corresponding user-defined *tag scheme* when the data generator has more than one user-defined *tag scheme*.
  - User-defined tag keys MUST NOT include a *tag scheme*-specific prefix when the data generator has only one user-defined *tag scheme*.

    - Reserved tag key prefixes MUST be prevented from being used as prefixes for any user-defined tag keys within a prefixless user-defined *tag scheme*.
- Tag finalization documentation MUST adhere to the following requirements:
    - Tag finalization documentation SHOULD include tag finalization methods and semantics.
    - Tag finalization documentation SHOULD be accessible to practitioners.

3.1.65.2. Provider-Defined vs. User-Defined Tags

This example illustrates various tags produced from multiple user-defined and provider-defined *tag schemes*. The first three tags illustrate examples from three different, user-defined *tag schemes*. The data generator predetermined that one user-defined *tag scheme* (i.e., `"foo": "bar"`) does not have a prepended prefix, but the remaining two user-defined *tag schemes* (i.e., `"userDefinedTagScheme2/foo": "bar"`, `"userDefinedTagScheme3/foo": true`) do have provider-defined and reserved prefixes. Additionally, the third tag is produced from a valueless, user-defined *tag scheme*, so the data generator also applies `true` as its default value.

The last two tags illustrate examples from two different, provider-defined *tag schemes*. Since all provider-defined *tag schemes* require a prefix, the data generator has prepended predefined and reserved prefixes (`providerDefinedTagScheme1/`, `providerDefinedTagScheme2/`) to each tag.


```
{
   "foo": "bar",
   "userDefinedTagScheme2/foo": "bar",
   "userDefinedTagScheme3/foo": true,
   "providerDefinedTagScheme1/foo": "bar",
   "providerDefinedTagScheme2/foo": "bar"
}
```


3.1.65.3. Finalized Tags

Within a data generator, tag keys may be associated with multiple values, and potentially defined at different levels within the data generator, such as accounts, folders, *resource* and other *resource* grouping constructs. When finalizing, the *data generator* reduces these multiple levels of definition to a single value where each key is associated with exactly one value. The method by which this is done and the semantics are up to each data generator and are documented within their respective documentation.

As an example, assume one *sub account* exists with one virtual machine with the following details, and tag inheritance favors Resources over *Sub Accounts*.

- Sub Account
    - id: *my-sub-account*
    - user-defined tags: *team:ops*, *env:prod*
- Virtual Machine
    - id: *my-vm*
    - user-defined tags: *team:web*

The table below represents a finalized dataset with these *resources*. It also shows the finalized state after all resource-oriented, tag inheritance rules are processed.

ResourceType	ResourceId	Tags
Sub Account	my-sub-account	{ "team": "ops", "env": "prod" }
Virtual Machine	my-vm	{ "team": "web", *"env": "prod"* }

Because the Virtual Machine Resource did not have an `env` tag, it inherited tag, `env:prod` (italicized), from its parent *sub account*. Conversely, because the Virtual Machine Resource already has a `team` tag (`team:web`), it did not inherit `team:ops` from its parent *sub account*.

3.1.65.4. Column ID

Tags

3.1.65.5. Display Name

Tags

3.1.65.6. Description

The set of tags assigned to *tag sources* that account for potential provider-defined or user-defined tag evaluations.

3.1.65.7. Content Constraints

Constraint	Value
Dataset	Cost and Usage
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	JSON
Value format	Key-Value Format

3.1.65.8. Version Introduced

1.0-preview

3.2. Billing Period

The Billing Period dataset is a supporting dataset that defines the time intervals and statuses associated with an *invoice issuer's* billing cycles for grouping and presenting *charges* on invoices. This dataset helps FinOps practitioners better understand how and when they can leverage Cost and Usage and Invoice Detail data for formal financial reporting and showback/chargeback processes.

Columns

Column	Column Type	Feature Level	Allows Nulls	Data Type
Billing Period Created	Dimension	Mandatory	False	Date/Time
Billing Period End	Dimension	Mandatory	False	Date/Time
Billing Period Last Updated	Dimension	Mandatory	False	Date/Time
Billing Period Start	Dimension	Mandatory	False	Date/Time
Billing Period Status	Dimension	Mandatory	False	String
Invoice Issuer Name	Dimension	Mandatory	False	String

Relationships

The Billing Period dataset is primarily used to provide context for the Cost and Usage and Invoice Detail datasets. It is joined using the Billing Period Start and Invoice Issuer Name columns available in those datasets.

Dataset A	Dataset A Column	Dataset B	Dataset B Column
Billing Period	Billing Period Start and Invoice Issuer Name	Cost and Usage	Billing Period Start and Invoice Issuer Name
Billing Period	Billing Period Start and Invoice Issuer Name	Invoice Detail	Billing Period Start and Invoice Issuer Name

Requirements

BillingPeriod MUST adhere to the following requirements:

- BillingPeriod MUST be present when the invoice issuer supports payable invoices.
- The presence of columns in BillingPeriod MUST adhere to the following requirements:
    - BillingPeriod MUST include BillingPeriodCreated.
    - BillingPeriod MUST include BillingPeriodEnd.
    - BillingPeriod MUST include BillingPeriodLastUpdated.
    - BillingPeriod MUST include BillingPeriodStart.
    - BillingPeriod MUST include BillingPeriodStatus.
    - BillingPeriod MUST include InvoiceIssuerName.
- BillingPeriod MUST conform to CorrectionHandling requirements.
- BillingPeriod MUST conform to DatasetCompleteness requirements.
- BillingPeriod MUST conform to DatasetConfiguration requirements.
- BillingPeriod MUST conform to DeliveryHandling requirements.
- BillingPeriod *FOCUS columns* MUST conform to FocusColumnHandling requirements.
- BillingPeriod *FOCUS columns* MUST conform to NullHandling requirements.
- BillingPeriod *custom columns* MUST conform to CustomColumnHandling requirements.

Dataset ID

BillingPeriod

Display Name

Billing Period

Description

Describes the time intervals and statuses associated with an invoice issuer's billing cycles.

Version Introduced

1.4

3.2.1. Billing Period Created

Billing Period Created is the timestamp when the Billing Period record was first created. This timestamp facilitates auditability of the charge and invoice lifecycle, allowing the FinOps practitioner to distinguish between the time of service consumption and the time of financial record generation.

3.2.1.1. Requirements

BillingPeriodCreated MUST adhere to the following requirements:

- BillingPeriodCreated MUST be of type Date/Time.
- BillingPeriodCreated MUST conform to DateTimeFormat requirements.
- BillingPeriodCreated MUST NOT be null.
- BillingPeriodCreated MUST represent the moment in time the Billing Period record was instantiated.

3.2.1.2. Column ID

BillingPeriodCreated

3.2.1.3. Display Name

Billing Period Created

3.2.1.4. Description

The timestamp when the *Billing Period* record was first created.

3.2.1.5. Content Constraints

Constraint	Value
Dataset	Billing Period
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.2.1.6. Version Introduced

1.4

3.2.2. Billing Period End

Billing Period End represents the *exclusive end bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes January since Billing Period Start represents the *inclusive start bound*, but does not include February since Billing Period End represents the *exclusive end bound*.

3.2.2.1. Requirements

BillingPeriodEnd MUST adhere to the following requirements:

- BillingPeriodEnd MUST be of type Date/Time.
- BillingPeriodEnd MUST conform to DateTimeFormat requirements.
- BillingPeriodEnd MUST NOT be null.
- BillingPeriodEnd MUST be the *exclusive end bound* of the *billing period*.

3.2.2.2. Column ID

BillingPeriodEnd

3.2.2.3. Display Name

Billing Period End

3.2.2.4. Description

The *exclusive end bound* of a *billing period*.

3.2.2.5. Content Constraints

Constraint	Value
Dataset	Billing Period
Column type	Dimension

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.2.2.6. Version Introduced

1.4

3.2.3. Billing Period Last Updated

Billing Period Last Updated is the timestamp when the Billing Period record was last updated. This timestamp helps FinOps practitioners ensure that they are working with the most current version of a record, particularly if corrections or status changes have been applied to the record after its initial creation.

3.2.3.1. Requirements

BillingPeriodLastUpdated MUST adhere to the following requirements:

- BillingPeriodLastUpdated MUST be of type Date/Time.
- BillingPeriodLastUpdated MUST conform to DateTimeFormat requirements.
- BillingPeriodLastUpdated MUST NOT be null.
- BillingPeriodLastUpdated MUST represent the most recent moment in time when any column value of the Billing Period record was created or modified.
- BillingPeriodLastUpdated MUST be greater than or equal to BillingPeriodCreated.

3.2.3.2. Column ID

BillingPeriodLastUpdated

3.2.3.3. Display Name

Billing Period Last Updated

3.2.3.4. Description

The timestamp when the Billing Period record was last updated.

3.2.3.5. Content Constraints

Constraint	Value
Dataset	Billing Period
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.2.3.6. Version Introduced

1.4

3.2.4. Billing Period Start

Billing Period Start represents the *inclusive start bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes January since Billing Period Start represents the *inclusive start bound*, but does not include February since BillingPeriodEnd represents the *exclusive end bound*.

3.2.4.1. Requirements

BillingPeriodStart MUST adhere to the following requirements:

- BillingPeriodStart MUST be of type Date/Time.
- BillingPeriodStart MUST conform to DateTimeFormat requirements.
- BillingPeriodStart MUST NOT be null.
- BillingPeriodStart MUST be the *inclusive start bound* of the *billing period*.

3.2.4.2. Column ID

BillingPeriodStart

3.2.4.3. Display Name

Billing Period Start

3.2.4.4. Description

The *inclusive start bound* of a *billing period*.

3.2.4.5. Content Constraints

Constraint	Value
Dataset	Billing Period
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.2.4.6. Version Introduced

1.4

3.2.5. Billing Period Status

Billing Period Status represents the state of the billing period (i.e., "Open" or "Closed"). This status helps FinOps practitioners

determine if the Cost and Usage and Invoice Detail data for a given period is preliminary and subject to change, or if all anticipated invoices have been issued and the delivered data is finalized and ready for formal financial reporting and showback/chargeback processes.

3.2.5.1. Requirements

BillingPeriodStatus MUST adhere to the following requirements:

- BillingPeriodStatus MUST be of type String.
- BillingPeriodStatus MUST NOT be null.
- BillingPeriodStatus MUST be one of the allowed values.
- BillingPeriodStatus MUST represent the state of the billing period identified by BillingPeriodStart and BillingPeriodEnd.
- BillingPeriodStatus MUST NOT be "Open" following a previous status of "Closed", except when explicitly requested or approved by the customer.

3.2.5.2. Allowed Values

Value	Description
Open	The billing period is currently active or still being processed. Records may continue to be added or revised.
Closed	The billing period has ended, all anticipated invoices have been issued, and the delivered data is finalized.

3.2.5.3. Implementation Guidance

While the transition from "Open" to "Closed" typically signifies the end of a billing cycle, in scenarios such as the following, it may be necessary to provide corrections to closed billing periods:

- Retroactive adjustments: an *invoice issuer* generates credits or corrections for a period previously marked as finalized.
- Audit corrections: discrepancies are discovered during financial reconciliation that require the data to be re-processed.
- Late-arriving usage: occasional delays in usage reporting necessitate a revision of the final invoice.

Corrections to closed billing periods are generally represented in the context of a subsequent open billing period to preserve historical financial accuracy and ensure transparent tracking. Exceptionally, a previously closed billing period may be reopened to apply such corrections, but this transition from "Closed" to "Open" must be explicitly requested or approved by the customer to maintain auditability and the integrity of financial reporting.

Corrections that do not impact the integrity of the closed billing period, such as informational or metadata updates, are allowed regardless of Billing Period Status.

FinOps tools and reporting engines should be designed to detect Billing Period Status transitions and corrections to closed billing periods, and trigger updates to downstream processes (e.g., cost allocation, chargeback, reporting) to ensure financial accuracy.

For more information, please see the Invoice and Billing Period Handling appendix and the Correction Handling attribute.

3.2.5.4. Column ID

BillingPeriodStatus

3.2.5.5. Display Name

Billing Period Status

3.2.5.6. Description

The state of the billing period (i.e., "Open" or "Closed"), indicating whether the delivered data for the period is preliminary, or if all anticipated invoices have been issued and the delivered data is finalized.

3.2.5.7. Content Constraints

Constraint	Value
Dataset	Billing Period
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.2.5.8. Version Introduced

1.4

3.2.6. Invoice Issuer Name

Invoice Issuer Name is the name of the entity responsible for issuing payable invoices for the *resources* or *services* consumed. It is commonly used for cost analysis and reporting scenarios.

3.2.6.1. Requirements

InvoiceIssuerName MUST adhere to the following requirements:

- InvoiceIssuerName MUST be of type String.
- InvoiceIssuerName MUST conform to StringHandling requirements.
- InvoiceIssuerName MUST NOT be null.
- InvoiceIssuerName MUST represent the entity that issues invoices.

See Appendix: Participating Entity Identification Examples section for examples of Invoice Issuer Name values across various use case scenarios.

3.2.6.2. Column ID

InvoiceIssuerName

3.2.6.3. Display Name

Invoice Issuer Name

3.2.6.4. Description

The name of the entity responsible for invoicing for the *resources* or *services* consumed.

3.2.6.5. Content Constraints

Constraint	Value
Dataset	Billing Period

Constraint	Value
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.2.6.6. Version Introduced

1.4

3.3. Contract Commitment

The Contract Commitment dataset is a supporting dataset that describes the terms of contracts agreed between a service provider and a customer.

Columns

Column	Column Type	Feature Level	Allows Nulls	Data Type
Billing Currency	Dimension	Mandatory	True	String
Contract Commitment Applicability	Dimension / Metric	Mandatory	False	JSON
Contract Commitment Benefit Category	Dimension	Mandatory	False	String
Contract Commitment Category	Dimension	Mandatory	False	String
Contract Commitment Cost	Metric	Mandatory	True	Decimal
Contract Commitment Created	Dimension	Mandatory	False	Date/Time
Contract Commitment Description	Dimension	Mandatory	True	String
Contract Commitment Discount Percentage	Metric	Mandatory	True	Decimal
Contract Commitment Duration Type	Dimension	Mandatory	False	String
Contract Commitment Fulfillment Interval	Dimension	Mandatory	False	String
Contract Commitment ID	Dimension	Mandatory	False	String
Contract Commitment Last Updated	Dimension	Mandatory	False	Date/Time
Contract Commitment Lifecycle Status	Dimension	Mandatory	False	String
Contract Commitment Model	Dimension	Mandatory	False	String
Contract Commitment Offer Category	Dimension	Mandatory	False	String
Contract Commitment Payment Interval	Dimension	Mandatory	False	String
Contract Commitment Payment Model	Dimension	Mandatory	False	String
Contract Commitment Payment Upfront Percentage	Metric	Conditional	False	Decimal
Contract Commitment Period End	Dimension	Mandatory	False	Date/Time
Contract Commitment Period Start	Dimension	Mandatory	False	Date/Time
Contract Commitment Quantity	Metric	Mandatory	True	Decimal
Contract Commitment Type	Dimension	Mandatory	False	String
Contract Commitment Unit	Dimension	Mandatory	True	String
Contract ID	Dimension	Mandatory	False	String
Contract Period End	Dimension	Mandatory	False	Date/Time
Contract Period Start	Dimension	Mandatory	False	Date/Time
Invoice Issuer Name	Dimension	Mandatory	False	String
Pricing Currency	Dimension	Conditional	False	String
Pricing Currency Contract Commitment Cost	Metric	Conditional	True	Decimal
Service Provider Name	Dimension	Mandatory	False	String

Relationships

The Contract Commitment dataset can be joined to the Cost and Usage dataset through the use of Contract Commitment ID.

- In the Contract Commitment dataset, Contract Commitment ID is a column.
- In the Cost and Usage dataset, Contract Commitment ID is a property within a JSON object array provided in Contract Applied column.

Dataset A	Dataset A Column	Dataset B	Dataset B Column
Contract Commitment	Contract Commitment ID	Cost and Usage	Contract Applied

Requirements

ContractCommitment MUST adhere to the following requirements:

- ContractCommitment MUST be present when the service provider supports *contract commitments*.
- ContractCommitment column presence MUST adhere to the following requirements:
  - ContractCommitment MUST include BillingCurrency.
  - ContractCommitment MUST include ContractCommitmentApplicability.
  - ContractCommitment MUST include ContractCommitmentBenefitCategory.
  - ContractCommitment MUST include ContractCommitmentCategory.
  - ContractCommitment MUST include ContractCommitmentCost.
  - ContractCommitment MUST include ContractCommitmentCreated.
  - ContractCommitment MUST include ContractCommitmentDescription.
  - ContractCommitment MUST include ContractCommitmentDiscountPercentage.
  - ContractCommitment MUST include ContractCommitmentDurationType.
  - ContractCommitment MUST include ContractCommitmentFulfillmentInterval.
  - ContractCommitment MUST include ContractCommitmentId.
  - ContractCommitment MUST include ContractCommitmentLastUpdated.
  - ContractCommitment MUST include ContractCommitmentLifecycleStatus.
  - ContractCommitment MUST include ContractCommitmentModel.
  - ContractCommitment MUST include ContractCommitmentOfferCategory.
  - ContractCommitment MUST include ContractCommitmentPaymentInterval.
  - ContractCommitment MUST include ContractCommitmentPaymentModel.
  - ContractCommitment MUST include ContractCommitmentPaymentUpfrontPercentage when the service provider offers "Partial Upfront" payment models.
  - ContractCommitment MUST include ContractCommitmentPeriodEnd.
  - ContractCommitment MUST include ContractCommitmentPeriodStart.
  - ContractCommitment MUST include ContractCommitmentQuantity.
  - ContractCommitment MUST include ContractCommitmentType.
  - ContractCommitment MUST include ContractCommitmentUnit.
  - ContractCommitment MUST include ContractId.
  - ContractCommitment MUST include ContractPeriodEnd.
  - ContractCommitment MUST include ContractPeriodStart.
  - ContractCommitment MUST include InvoiceIssuerName.
  - ContractCommitment MUST include PricingCurrency when the service provider supports pricing and billing in different currencies.
  - ContractCommitment MUST include PricingCurrencyContractCommitmentCost when the service provider supports pricing and billing in different currencies.
  - ContractCommitment MUST include ServiceProviderName.
- ContractCommitment MUST conform to CorrectionHandling requirements.
- ContractCommitment MUST conform to DatasetCompleteness requirements.
- ContractCommitment MUST conform to DatasetConfiguration requirements.
- ContractCommitment MUST conform to DeliveryHandling requirements.
- ContractCommitment *FOCUS columns* MUST conform to FocusColumnHandling requirements.
- ContractCommitment *FOCUS columns* MUST conform to NullHandling requirements.
- ContractCommitment *custom columns* MUST conform to CustomColumnHandling requirements.

Dataset ID

ContractCommitment

Display Name

Contract Commitment

Description

Describes the terms of contracts agreed between a service provider and a customer.

Version Introduced

1.3

3.3.1. Billing Currency

Billing currency is an identifier that represents the currency of a *contract commitment*.

3.3.1.1. Requirements

BillingCurrency MUST adhere to the following requirements:

- BillingCurrency MUST be of type String.
- BillingCurrency MUST conform to StringHandling requirements.
- BillingCurrency MUST conform to CurrencyFormat requirements.
- BillingCurrency MUST NOT be null when ContractCommitmentCategory is "Spend".
- BillingCurrency MUST match the currency used in the invoice generated by the *invoice issuer*.
- BillingCurrency MUST be expressed in *national currency* (e.g., USD, EUR).

3.3.1.2. Column ID

BillingCurrency

3.3.1.3. Display Name

Billing Currency

3.3.1.4. Description

Represents the currency of a *contract commitment*.

3.3.1.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	Currency Format

3.3.1.6. Version Introduced

1.3

3.3.2. Contract Commitment Applicability

Contract Commitment Applicability is a structured definition of the specific entities eligible for coverage under a *contract commitment*. This column details inclusionary and exclusionary logic, as well as the specific portion of eligible cost or usage that is applicable.

3.3.2.1. Requirements

3.3.2.1.1. Column Requirements

ContractCommitmentApplicability MUST adhere to the following requirements:

- ContractCommitmentApplicability MUST be of type JSON Object (serialized as a String where necessary).
- ContractCommitmentApplicability MUST conform to StringHandling requirements.
- ContractCommitmentApplicability MUST conform to JsonObjectFormat requirements.
- ContractCommitmentApplicability MUST conform to ContractCommitmentApplicabilityObject requirements.
- ContractCommitmentApplicability MUST NOT be null.

3.3.2.2. Contract Commitment Applicability Object

Contract Commitment Applicability consists of a valid JSON object which contains a set of top-level property keys. These keys define entity-based inclusionary and exclusionary logic, as well as the portion of relevant cost and/or usage that is applicable to the *contract commitment*.

The following section details the normative requirements for the ContractCommitmentApplicabilityObject and its nested properties. For a logical overview of the expected content, see the Schema Structure and Object Example sections.

3.3.2.2.1. Object Requirements

ContractCommitmentApplicabilityObject MUST adhere to the following requirements:

- ContractCommitmentApplicabilityObject MUST conform to the ContractCommitmentApplicabilityObjectSchema JSON Schema.
- ContractCommitmentApplicabilityObject.IsGlobalScope MUST be `true` when the *contract commitment* applies to all entities.
- ContractCommitmentApplicabilityObject.IsComplexScope MUST be `true` when the *contract commitment's* applicability logic exceeds schema capabilities.
- ContractCommitmentApplicabilityObject.Applicability.Cost MUST represent the fraction of an eligible charge's cost that is applicable to the commitment (0.0 to 1.0).
- ContractCommitmentApplicabilityObject.Applicability.Usage MUST represent the fraction of an eligible charge's usage that is applicable to the commitment (0.0 to 1.0).
- ContractCommitmentApplicabilityObject.Inclusions[*].Applicability.Cost MUST represent the fraction of an eligible charge's cost that is applicable to the commitment (0.0 to 1.0).
- ContractCommitmentApplicabilityObject.Inclusions[*].Applicability.Usage MUST represent the fraction of an eligible charge's usage that is applicable to the commitment (0.0 to 1.0).
- ContractCommitmentApplicabilityObject.Inclusions[*].Dimension SHOULD represent a column in Cost and Usage.
- ContractCommitmentApplicabilityObject.Exclusions[*].Dimension SHOULD represent a column in Cost and Usage.
- ContractCommitmentApplicabilityObject.Inclusions[*].Values MUST contain only the single string "*" when the wildcard is present.
- ContractCommitmentApplicabilityObject.Exclusions[*].Values MUST contain only the single string "*" when the wildcard is present.

3.3.2.2.2. Object Schema Structure

ContractCommitmentApplicability contains a structured JSON object defining the logical boundaries and the applicability percentage of a commitment.

Top-Level Properties

Property	Type	Required	Description
`IsGlobalScope`	Boolean	No	If `true`, the commitment applies to all entities. Defaults to `false`.
`IsComplexScope`	Boolean	No	If `true`, indicates logic exceeds schema capabilities. Defaults to `false`.

Property	Type	Required	Description
`Applicability`	Object	No	The fractional mapping for metrics. If omitted, both `Cost` and `Usage` keys default to `1.0`.
`InclusionOperator`	String	Conditional	Required only if `IsGlobalScope` and `IsComplexScope` are both `false` or null. Valid values: `And`, `Or`. Must be omitted if Global or Complex scope is true.
`Inclusions`	Array	Conditional	Required only if `IsGlobalScope` and `IsComplexScope` are both `false` or null. List of `Rule` objects defining the boundary. Must be omitted if Global or Complex scope is true.
`ExclusionOperator`	String	Conditional	Required only if `Exclusions` are present. Defines the relationship for `Exclusions`. Valid values: `And`, `Or`.
`Exclusions`	Array	No	List of `Rule` objects defining entities to be removed from the boundary.

Rule Object

Key	Type	Description
`Dimension`	String	A valid FOCUS Column Name (e.g., `SkuId`, `RegionId`).
`Operator`	String	The comparison logic to apply. Must be one of the Supported Operators.
`Values`	Array	A list of strings to compare. A value of `["*"]` acts as a global wildcard.
`Applicability`	Object	Optional. The specific fraction of applicability for entities matching this rule. Overrides the top-level `Applicability`.

Applicability Object

Key	Type	Default	Description
`Cost`	Decimal	1.0	Fraction of an eligible charge's cost applicable to the *contract commitment*.
`Usage`	Decimal	1.0	Fraction of an eligible charge's usage applicable to the *contract commitment*.

Supported Operators

Operator	Logic	Usage Example
`In`	Exact match against any item in the list.	`["us-east-1", "us-west-2"]`
`NotIn`	Does not match any item in the list.	`["123456789"]`
`StartsWith`	String prefix match.	`["prod-"]`
`NotStartsWith`	Does not begin with the specified prefix.	`["test-"]`
`Contains`	Substring match anywhere in the value.	`["database"]`
`NotContains`	Substring is not present in the value.	`["sandbox"]`
`EndsWith`	String suffix match.	`["-temp"]`
`Exists`	Checks if the dimension is present and not null.	`Values` must be `["*"]`
`DoesNotExist`	Checks if the dimension is missing or null.	`Values` must be `["*"]`

Wildcard Handling

ContractCommitmentApplicability uses a reserved string to represent global or unrestricted boundaries within a specific Dimension.

Reserved Value	Description	Supported Operators
`"*"`	Represents all possible values for the specified Dimension.	`In`, `Contains`, `Exists`, `DoesNotExist`

Wildcard Behavior Rules

1. **Inclusion Logic:** When `["*"]` is used in an Inclusion rule, the rule evaluates to `True` for every entity, effectively making the commitment "Organization-wide" for that specific Dimension.
2. **Exclusion Logic:** When `["*"]` is used in an Exclusion rule, the rule evaluates to `True` for every entity, effectively excluding all entities (this is typically used only in combination with `ExclusionOperator: "And"` for surgical filtering).
3. **Implicit Wildcards:** If a Dimension (e.g., `RegionId`) is omitted entirely from the `Inclusions` array, it is treated as an implicit wildcard (unrestricted).

3.3.2.2.3. Object Implementation Guidance

Processing Workflow

The evaluation of an entity against a commitment applicability must follow a strict linear progression:

1. **Normalization:** Convert the entity attribute and the Scope `Values` to a consistent case (default: lowercase) for comparison.
2. **Inclusion Evaluation:** Iterate through `Inclusions`. If a match is found, record the rule-level `Applicability` if present. Apply `InclusionOperator`. If result is `False`, terminate.
3. **Exclusion Evaluation:** Iterate through `Exclusions`. If `True`, terminate evaluation.
4. **Applicability Resolution:**
    - **Inheritance:** A matching rule's `Applicability` object takes precedence over the top-level object.
    - **Defaulting:** If a metric key (`Cost` or `Usage`) is missing within a provided `Applicability` object, the engine must default that specific value to `1.0`.
    - **Rule-level Priority:** Use the `Applicability` from the matching inclusion rule. If multiple rules match under `Or`, the engine must use the highest percentage for each respective metric.
    - **Fallback:** Use the top-level `Applicability` if no rule-level value is provided.

Integration with Commitment Logic

The evaluation of **Applicability** percentages must be contextually aligned with the Contract Commitment Model and Contract Commitment Fulfillment Interval:

- **Continuous Models:** Applicability percentages must be applied to each discrete unit of activity (e.g., every hour) within the **Fulfillment Interval**. If the commitment is not fully utilized by eligible entities within that hour, the remaining capacity expires.
- **Discontinuous Models:** Applicability percentages determine the portion of aggregate activity that counts toward fulfillment over the entire **Interval** (e.g., a full year).

Dependency Logic

1. **Consistency:** Engines should expect a JSON Object and should not support scalar (Decimal/Float) values for this field to ensure compatibility with typed database schemas.
2. **Conflict Resolution:** If `IsGlobalScope` or `IsComplexScope` is `true`, the `Inclusions` array must be empty or omitted. Additionally, `IsGlobalScope` and `IsComplexScope` must both not be `true` at the same time. Engines should validate these structural constraints before processing.

3.3.2.2.4. Object Example

Here is a basic example of the object format, describing organization-wide coverage **except** for Database services running in BillingAccountId 123456789012.

- For more detailed examples, please see this column's entry in the JSON Object Examples appendix entry here.
- For the JSON schema, please see Contract Commitment Applicability Object Schema.

```
{
 "IsGlobalScope": true,
 "ExclusionOperator": "And",
 "Exclusions": [
  {
   "Dimension": "BillingAccountId",
   "Operator": "In",
   "Values": ["123456789012"]
  },
  {
   "Dimension": "ServiceCategory",
   "Operator": "In",
   "Values": ["Database"]
  }
 ]
}
```

3.3.2.2.5. Object ID

ContractCommitmentApplicabilityObject

3.3.2.2.6. Object Display Name

Contract Commitment Applicability Object

3.3.2.3. Column ID

ContractCommitmentApplicability

3.3.2.4. Display Name

Contract Commitment Applicability

3.3.2.5. Description

A structured definition of the specific entities to which a contract commitment applies, including inclusion/exclusion logic and applicability percentages.

3.3.2.6. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension / Metric
Feature level	Mandatory
Allows nulls	False
Data type	JSON
Value format	JSON Object Format
Object	ContractCommitmentApplicabilityObject

3.3.2.7. Version Introduced

1.4

3.3.3. Contract Commitment Benefit Category

Contract Commitment Benefit Category defines the primary value or advantage received for a *contract commitment*. It identifies whether the benefit is realized as a price reduction, a feature entitlement, a guarantee of service availability, or some other commercial outcome.

3.3.3.1. Requirements

ContractCommitmentBenefitCategory MUST adhere to the following requirements:

- ContractCommitmentBenefitCategory MUST be of type String.
- ContractCommitmentBenefitCategory MUST NOT be null.
- ContractCommitmentBenefitCategory MUST be one of the allowed values.

3.3.3.2. Allowed Values

Value	Sort Order	Description	Typical Use Case

Value	Sort Order	Description	Typical Use Case
Discount	10	A financial reduction in the unit price or list rate, whether applied immediately or conditionally upon meeting usage or spend thresholds.	Flat rate negotiated reductions, Savings Plans, growth rebates, or volume-tier discounts.
Entitlement	20	The contractual right to access and consume specific products, features, or software tiers that would otherwise be unavailable.	Marketplace SaaS purchases, Enterprise Agreements (e.g., Snowflake), or paid Proof of Concepts.
Availability	30	A contractual assurance of resource access and physical capacity.	Capacity reservations or dedicated host guarantees.
Other	40	Benefits not captured by standard categories.	Support access, training, or professional services.

3.3.3.3. Implementation Guidance

3.3.3.3.1. Distinguishing Outcomes from Mechanisms

When categorizing a commitment, the value reflects the actual commercial benefit received, rather than the funding or consumption mechanism used to acquire it. For example, a prepaid "monetary pool" or a drawdown fund is a mechanism; the *benefit* of that pool is typically the right to use the feature (**Entitlement**) or the reduced rate unlocked by the commitment (**Discount**).

3.3.3.3.2. Distinguishing from Technical IDs

Availability represents the contractual right to access resources. It must not be confused with technical fields like `CapacityReservationId`. A single **Availability** within a contract may encompass multiple technical reservations across various regions or accounts.

3.3.3.3.3. Primary Benefit Logic

In cases where a commitment provides multiple benefits (e.g., an Entitlement that also includes a Discount), the value should reflect the **primary commercial driver** of the agreement as defined by the procurement or FinOps team.

3.3.3.4. Column ID

ContractCommitmentBenefitCategory

3.3.3.5. Display Name

Contract Commitment Benefit Category

3.3.3.6. Description

Defines the primary value or advantage received for a *contract commitment*.

3.3.3.7. Content Constraints

Constraint	Value

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.3.8. Version Introduced

1.4

3.3.4. Contract Commitment Category

Contract Commitment Category represents the highest-level classification of a *contract commitment* based on the nature of how it is applied to a charge. Contract Commitment Category is commonly used to identify and distinguish between categories of contract commitments that may require different handling.

3.3.4.1. Requirements

ContractCommitmentCategory MUST adhere to the following requirements:

- ContractCommitmentCategory MUST be of type String.
- ContractCommitmentCategory MUST NOT be null.
- ContractCommitmentCategory MUST be one of the allowed values.

3.3.4.2. Allowed Values

Value	Description
Spend	Contract commitments that require a predetermined amount of spend.
Usage	Contract commitments that require a predetermined amount of usage.

3.3.4.3. Column ID

ContractCommitmentCategory

3.3.4.4. Display Name

Contract Commitment Category

3.3.4.5. Description

Represents the highest-level classification of a *contract commitment* based on the nature of how it is applied to a charge.

3.3.4.6. Content Constraints

Constraint	Value
Dataset	Contract Commitment

Constraint	Value
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.4.7. Version Introduced

1.3

3.3.5. Contract Commitment Cost

Contract Commitment Cost represents the monetary value of the *contract commitment*. Contract Commitment Cost is commonly used for monitoring the progress towards fulfilling contractual commitments that may facilitate discounts for *resources* or *services* as agreed between a service provider and a customer.

3.3.5.1. Requirements

ContractCommitmentCost MUST adhere to the following requirements:

- ContractCommitmentCost MUST be of type Decimal.
- ContractCommitmentCost MUST conform to NumericFormat requirements.
- ContractCommitmentCost MUST adhere to the following nullability requirements:
  - ContractCommitmentCost MUST NOT be null when ContractCommitmentCategory is "Spend".
  - ContractCommitmentCost MAY be null when ContractCommitmentCategory is "Usage".
- ContractCommitmentCost MUST be denominated in the BillingCurrency.

3.3.5.2. Column ID

ContractCommitmentCost

3.3.5.3. Display Name

Contract Commitment Cost

3.3.5.4. Description

The monetary value of the *contract commitment*.

3.3.5.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Metric
Feature level	Mandatory
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.3.5.6. Version Introduced

1.3

3.3.6. Contract Commitment Created

Contract Commitment Created is the timestamp when the Contract Commitment record was first created. This timestamp facilitates auditability of the contract commitment lifecycle.

3.3.6.1. Requirements

ContractCommitmentCreated MUST adhere to the following requirements:

- ContractCommitmentCreated MUST be of type Date/Time.
- ContractCommitmentCreated MUST conform to DateTimeFormat requirements.
- ContractCommitmentCreated MUST NOT be null.
- ContractCommitmentCreated MUST represent the moment in time the Contract Commitment record was instantiated.

3.3.6.2. Column ID

ContractCommitmentCreated

3.3.6.3. Display Name

Contract Commitment Created

3.3.6.4. Description

The timestamp when the contract commitment record was first created.

3.3.6.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.6.6. Version Introduced

1.4

3.3.7. Contract Commitment Description

Contract Commitment Description provides a high-level context of a *contract commitment* without requiring additional discovery. Contract Commitment Description is a self-contained summary of the contract commitment's terms, which may not be sufficiently described by the other columns of the Contract Commitment dataset.

3.3.7.1. Requirements

ContractCommitmentDescription MUST adhere to the following requirements:

- ContractCommitmentDescription MUST be of type String.
- ContractCommitmentDescription MUST conform to StringHandling requirements.
- ContractCommitmentDescription SHOULD NOT be null.
- ContractCommitmentDescription maximum length SHOULD be provided in the corresponding FOCUS Metadata Schema.

3.3.7.2. Column ID

ContractCommitmentDescription

3.3.7.3. Display Name

Contract Commitment Description

3.3.7.4. Description

The self-contained summary of the *contract commitment's* terms.

3.3.7.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	<not specified>

3.3.7.6. Version Introduced

1.3

3.3.8. Contract Commitment Discount Percentage

Contract Commitment Discount Percentage represents the effective percentage reduction applied to the list price for resources or services covered by a *contract commitment*.

3.3.8.1. Requirements

ContractCommitmentDiscountPercentage MUST adhere to the following requirements:

- ContractCommitmentDiscountPercentage MUST be of type Decimal.
- ContractCommitmentDiscountPercentage MUST conform to NumericFormat requirements.
- ContractCommitmentDiscountPercentage MUST adhere to the following nullability requirements:
    - ContractCommitmentDiscountPercentage MUST NOT be null when ContractCommitmentBenefitCategory is "Discount".
    - ContractCommitmentDiscountPercentage MUST be null when ContractCommitmentBenefitCategory is "Availability".

- ContractCommitmentDiscountPercentage MUST be a value between 0.0 and 1.0, inclusive.
- For contracts with multiple tiers (e.g., 5% discount up to 1M, 10% above 1M), ContractCommitmentDiscountPercentage MUST adhere to the following additional requirements:
  - ContractCommitmentDiscountPercentage MUST reflect the discount percentage defined for the specific pricing tier represented by the Contract Commitment row.
  - ContractCommitmentDiscountPercentage MUST correspond to only one pricing tier per Contract Commitment row.
- ContractCommitmentDiscountPercentage SHOULD represent the net effective discount when multiple contractual layers are applicable (e.g., a negotiated discount on top of a standard commitment).

3.3.8.2. Implementation Guidance

3.3.8.2.1. Calculating the Effective Percentage

In scenarios where a commitment has "stacked" or "nested" discounts, this field should reflect the total reduction from the list price.

Discount Percentage = 1 - (Contracted Unit Price / List Unit Price)

For example, consider usage that yields a cost for which the following discounts are applicable:

- Commitment discount: 20% applied to the on-demand cost
- Negotiated discount: 10% applied to the post-commitment discount cost

The effective discount is not 30%. It is: 1 - (0.8 × 0.9) = 28%

In this scenario, ContractCommitmentDiscountPercentage should be reported as 28%.

3.3.8.2.2. Relationship to Benefit Category

This field provides the magnitude for the **Discount** category. While `ContractCommitmentCost` tracks the financial obligation (what you pay), `ContractCommitmentDiscountPercentage` tracks the benefit rate (what you save).

3.3.8.2.3. Tiered Incentives

For commitments with multiple tiers (e.g., 5% discount up to 1M, 10% above 1M), this column should represent the **active** or **base** discount percentage applicable to the current contract row.

3.3.8.3. Column ID

ContractCommitmentDiscountPercentage

3.3.8.4. Display Name

Contract Commitment Discount Percentage

3.3.8.5. Description

The effective percentage reduction applied to the list price of resources or services covered by a *contract commitment*.

3.3.8.6. Content Constraints

Constraint	Value

Constraint	Value
Dataset	Contract Commitment
Column type	Metric
Feature level	Mandatory
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	0.0 to 1.0 (inclusive)

3.3.8.7. Version Introduced

1.4

3.3.9. Contract Commitment Duration Type

Contract Commitment Duration Type represents the categorical length of a *contract commitment* (e.g., "1 Year", "3 Years") based on the commercial agreement or pricing model.

This column serves as a stable classifier for the commitment's duration, distinct from the actual lifespan of the specific record. For example, a 3-year commitment that is exchanged or modified may have a calculated duration of only a few months, but its Contract Commitment Duration Type remains "3 Years". This allows for consistent grouping and reporting on commitment durations, regardless of lifecycle events.

3.3.9.1. Requirements

ContractCommitmentDurationType MUST adhere to the following requirements:

- ContractCommitmentDurationType MUST be of type String.
- ContractCommitmentDurationType MUST conform to StringHandling requirements.
- ContractCommitmentDurationType MUST NOT be null.
- ContractCommitmentDurationType SHOULD be expressed with a quantity and time unit, where quantity is a positive integer, and time-unit is a standardized unit of time, either singular or plural (e.g., "1 Day", "1 Year", "3 Months", "3 Years").
- ContractCommitmentDurationType SHOULD present the unit of time as one of the allowed values.
- ContractCommitmentDurationType SHOULD correspond to the standard duration of the purchased offering (e.g., "1 Year", "3 Years") rather than a precise calculation of days or hours.
- ContractCommitmentDurationType MAY differ from the actual duration calculated between ContractCommitmentPeriodStart and ContractCommitmentPeriodEnd (e.g., if a 3-year commitment is exchanged in its final month, the resulting record may have a short lifespan but retains a value of "3 Years").

3.3.9.2. Allowed Values

The following units should be used for the representation of time:

Time Unit
Minute
Minutes
Hour
Hours
Day
Days
Week
Weeks
Month
Months
Quarter

Time Unit
Quarters
Year
Years

3.3.9.3. Expected Format

A given Contract Commitment Duration Type value follows a structured format of "[Numeric Value] [Unit]".

- [Numeric Value]: A positive integer.
- [Unit]: A standardized unit of time, singular or plural (e.g., Hour, Year, Years).

3.3.9.4. Column ID

ContractCommitmentDurationType

3.3.9.5. Display Name

Contract Commitment Duration Type

3.3.9.6. Description

Represents the categorical length of the *contract commitment* offering.

3.3.9.7. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Expected format

3.3.9.8. Version Introduced

1.4

3.3.10. Contract Commitment Fulfillment Interval

Contract Commitment Fulfillment Interval represents the specific *period* used to measure and reset the fulfillment of a *contract commitment*. It establishes the window during which usage is aggregated to determine if commitment obligations have been met. At the end of each fulfillment interval, the contract commitment model logic is applied, either resulting in the expiration of unused capacity (Continuous) or the calculation of a balance or true-up (Discontinuous).

Contract Commitment Fulfillment Interval has a series of possible values that represent a length of time, typically recurring over the contract commitment duration type. Continuous models are typically Hourly, whereas Discontinuous models are typically Daily or greater.

3.3.10.1. Requirements

ContractCommitmentFulfillmentInterval MUST adhere to the following requirements:

- ContractCommitmentFulfillmentInterval MUST be of type String.
- ContractCommitmentFulfillmentInterval MUST NOT be null.
- ContractCommitmentFulfillmentInterval MUST be one of the allowed values.
- ContractCommitmentFulfillmentInterval MUST NOT be "Full Period" when ContractCommitmentModel is "Continuous".

3.3.10.2. Allowed Values

Value	Sort Order	Description	Typical Use Case
Hourly	10	Measured over a 60-minute period.	Continuous Model: Cloud-native RIs and Savings Plans.
Daily	20	Measured over a calendar day.	Daily active user (DAU) caps or daily license minimums.
Weekly	30	Measured over a rolling or fixed 7-day period.	Burstable bandwidth or weekly sprint-based SaaS usage.
Monthly	40	Measured over a calendar month.	Discontinuous Model: SaaS MRR minimums or tiered discounts.
Quarterly	50	Measured over a 3-month fiscal period.	Enterprise true-ups or volume-based rebate targets.
Semi-Annual	60	Measured over a 6-month period.	Mid-year budget alignments or review cycles.
Annual	70	Measured over a 12-month period.	Discontinuous Model: Cloud EAs (Enterprise Agreements).
Full Period	80	The full duration of the contract commitment, with no internal resets.	"Pool-of-funds commitments" spanning the full commitment duration.
Transactional	90	No time-based reset; based purely on event volume or credit consumption.	API call bundles or "Credit Packs" with no expiration date.
Custom	100	A bespoke interval that does not fit standard calendar units.	Bridge contracts, unique POCs, or non-standard durations (e.g., 100 days).

Note: the sort orders and use cases presented above are included for convenience and are not defined as separate FOCUS columns.

3.3.10.3. Column ID

ContractCommitmentFulfillmentInterval

3.3.10.4. Display Name

Contract Commitment Fulfillment Interval

3.3.10.5. Description

Represents the specific *period* used to measure and reset the fulfillment of a *contract commitment*.

3.3.10.6. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.10.7. Version Introduced

1.4

3.3.11. Contract Commitment ID

Contract Commitment ID is a service-provider-assigned identifier describing a single contract term agreed between a provider and a customer. Contracts can include commitments to a certain amount of spend or usage over an agreed period of time.

3.3.11.1. Requirements

ContractCommitmentId MUST adhere to the following requirements:

- ContractCommitmentId MUST be of type String.
- ContractCommitmentId MUST conform to StringHandling requirements.
- ContractCommitmentId MUST NOT be null.
- ContractCommitmentId MUST be a unique identifier within the service provider.
- ContractCommitmentId SHOULD be a fully-qualified identifier.
- ContractCommitmentId MUST have one and only one parent ContractId.
- ContractCommitmentId MAY match ContractId.

3.3.11.2. Column ID

ContractCommitmentId

3.3.11.3. Display Name

Contract Commitment ID

3.3.11.4. Description

A service-provider-assigned identifier describing a single contract term agreed between a service provider and a customer.

3.3.11.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.3.11.6. Version Introduced

1.3

3.3.12. Contract Commitment Last Updated

Contract Commitment Last Updated is the timestamp when the Contract Commitment record was last updated. This timestamp helps FinOps practitioners ensure that they are working with the most current version of a Contract Commitment record, particularly if corrections or status changes have been applied to the record after its initial creation.

3.3.12.1. Requirements

ContractCommitmentLastUpdated MUST adhere to the following requirements:

- ContractCommitmentLastUpdated MUST be of type Date/Time.
- ContractCommitmentLastUpdated MUST conform to DateTimeFormat requirements.
- ContractCommitmentLastUpdated MUST NOT be null.
- ContractCommitmentLastUpdated MUST represent the most recent moment in time when any column value of the Contract Commitment record was created or modified.
- ContractCommitmentLastUpdated MUST be greater than or equal to ContractCommitmentCreated.

3.3.12.2. Column ID

ContractCommitmentLastUpdated

3.3.12.3. Display Name

Contract Commitment Last Updated

3.3.12.4. Description

The timestamp when the contract commitment record was last updated.

3.3.12.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.12.6. Version Introduced

1.4

3.3.13. Contract Commitment Lifecycle Status

Contract Commitment Lifecycle Status represents the current lifecycle state of a *contract commitment*. The Status determines the applicability of the commitment to a specific period of Cost and Usage data.

3.3.13.1. Requirements

ContractCommitmentLifecycleStatus MUST adhere to the following requirements:

- ContractCommitmentLifecycleStatus MUST be of type String.
- ContractCommitmentLifecycleStatus MUST NOT be null.
- ContractCommitmentLifecycleStatus MUST be one of the allowed values.
- When a contract commitment record is modified in a way that requires a new ContractCommitmentID, ContractCommitmentLifecycleStatus for the previous record MUST be "Superseded".

3.3.13.2. Allowed Values

Value	Sort Order	Description
Proposed	10	The commitment is being negotiated or modeled; it has no legal or financial impact on current data.
Pending	20	The commitment is finalized or signed, but the effective start date is in the future.
Active	30	The commitment is currently in effect, and it has remaining value.
Exhausted	40	The commitment is currently in effect, but its value has been fully consumed.
Expired	50	The commitment is no longer active because it reached its scheduled end date.
Canceled	60	The commitment is no longer active because it was terminated by either party prior to its scheduled end date.
Superseded	70	The commitment is no longer active because it was replaced by a newer version prior to its scheduled end date.

3.3.13.3. Column ID

ContractCommitmentLifecycleStatus

3.3.13.4. Display Name

Contract Commitment Lifecycle Status

3.3.13.5. Description

The current lifecycle state of a *contract commitment*.

3.3.13.6. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.13.7. Version Introduced

3.3.14. Contract Commitment Model

Contract Commitment Model represents the operational behavior and consumption flexibility of a *contract commitment*. This field distinguishes between rigid, "use-it-or-lose-it" obligations (typically hourly) and flexible, aggregate-based agreements that accommodate variable usage patterns.

Contract Commitment Model has two possible values: **Continuous** and **Discontinuous**. Continuous models (e.g., reserved instances, savings plans) represent a flat, constant "floor" of commitment where any dip in usage results in immediate, unrecoverable waste. Discontinuous models (e.g., enterprise agreements, SaaS minimum spend agreements) represent a broader, more flexible bucket where the commitment is "spikier": the usage can fluctuate wildly, but as long as the aggregate hits the target (or the true-up handles the variance), the commitment is satisfied. In either case, the interval of the commitment is represented by the Contract Commitment Fulfillment Interval.

3.3.14.1. Requirements

ContractCommitmentModel MUST adhere to the following requirements:

- ContractCommitmentModel MUST be of type String.
- ContractCommitmentModel MUST NOT be null.
- ContractCommitmentModel MUST be one of the allowed values.
- ContractCommitmentModel MUST be "Discontinuous" when ContractCommitmentFulfillmentInterval is "Full Period".

3.3.14.2. Allowed Values

Value	Description
Continuous	A flat, constant "floor" of commitment (e.g., RIs, Savings Plans). Coverage is applied at a fixed rate per Contract Commitment Fulfillment Interval (usually hourly), and benefits are not carried over to subsequent intervals.
Discontinuous	A flexible, aggregate commitment (e.g., Enterprise Agreements, SaaS Minimum Spend). Coverage is measured over a broad window or against a total monetary value.

3.3.14.3. Implementation Guidance

3.3.14.3.1. Reporting and Analysis

- For *continuous* models: report on Utilization % to identify immediate waste.
- For *discontinuous* models: report on Burn Rate and Remaining Balance to ensure the "spikes" are not trending toward an early exhaustion of the fund or a massive year-end true-up bill.

3.3.14.3.2. Relationship with Fulfillment Interval

Because a `Continuous` model dictates a recurring, "use-it-or-lose-it" evaluation window, it cannot logically span an entire, cumulative contract term without a reset. Therefore, if the associated Contract Commitment Fulfillment Interval is `Full Period`, the Contract Commitment Model must be categorized as `Discontinuous`.

3.3.14.4. Column ID

ContractCommitmentModel

3.3.14.5. Display Name

Contract Commitment Model

3.3.14.6. Description

Represents the operational behavior and consumption flexibility of a *contract commitment*.

3.3.14.7. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.14.8. Version Introduced

1.4

3.3.15. Contract Commitment Offer Category

Contract Commitment Offer Category indicates whether the pricing and terms of a *contract commitment* are based on a standard, publicly accessible offering or have been specifically brokered through private negotiation.

Contract Commitment Offer Category has two possible values: **Public** and **Negotiated**. *Public* denotes terms and pricing that are generally available to all customers via a service provider's standard rate card or portal. *Negotiated* denotes terms and pricing that have been specifically modified through an agreement between the customer and the service provider.

3.3.15.1. Requirements

ContractCommitmentOfferCategory MUST adhere to the following requirements:

- ContractCommitmentOfferCategory MUST be of type String.
- ContractCommitmentOfferCategory MUST NOT be null.
- ContractCommitmentOfferCategory MUST be one of the allowed values.

3.3.15.2. Allowed Values

Value	Description	Typical Use Case
Public	Terms that are generally available to all customers via a service provider's standard rate card or portal.	Standard Savings Plans or Reserved Instances purchased via the cloud console without a custom discount.
Negotiated	Terms and pricing that have been specifically modified through an agreement between the customer and the service provider.	Enterprise Agreements (EA), private marketplace offers, or custom SaaS contracts with volume-based discounting.

3.3.15.3. Implementation Guidance

- Use Public as your baseline for market comparison.

- Use Negotiated to track the efficacy of your procurement team's discount efforts.

Sensitivity Note: Records marked as Negotiated often fall under non-disclosure agreements (NDAs). This field can serve as a metadata tag for data masking or access control when sharing reports with third parties.

3.3.15.4. Column ID

ContractCommitmentOfferCategory

3.3.15.5. Display Name

Contract Commitment Offer Category

3.3.15.6. Description

Indicates whether the pricing and terms of a *contract commitment* are based on a standard, publicly accessible offering or have been specifically brokered through private negotiation.

3.3.15.7. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.15.8. Version Introduced

1.4

3.3.16. Contract Commitment Payment Interval

Contract Commitment Payment Interval represents the frequency by which a *contract commitment* is invoiced. For payment models involving deferred financial obligations, the Payment Interval denotes the ongoing billing cycle. For models paid upfront, the Payment Interval denotes the single settlement event.

Note: Do not confuse the Contract Commitment Payment Interval with the Contract Commitment Fulfillment Interval. For example, a spend-based commitment discount may have an Hourly Fulfillment Interval (usage reset) but a Monthly Payment Interval (billing cycle).

3.3.16.1. Requirements

ContractCommitmentPaymentInterval MUST adhere to the following requirements:

- ContractCommitmentPaymentInterval MUST be of type String.
- ContractCommitmentPaymentInterval MUST NOT be null.
- ContractCommitmentPaymentInterval MUST be one of the allowed values.
- ContractCommitmentPaymentInterval MUST be "One-Time" when ContractCommitmentPaymentModel is "All Upfront".
- ContractCommitmentPaymentInterval SHOULD represent a time granularity equal to or lesser than the time granularity represented by ContractCommitmentDurationType.

3.3.16.2. Allowed Values

Value	Sort Order	Description	Typical Use Case
One-Time	10	A single invoice is generated for the entire obligation.	All Upfront models (e.g., 3yr All-Upfront RI) or single-invoice arrears paid at the end of a term.
Monthly	20	Invoices for the deferred balance are generated once per month.	No Upfront Savings Plans or Monthly SaaS.
Quarterly	30	Invoices for the deferred balance are generated every three months.	Partial Upfront deals with 90-day true-ups.
Semi-Annual	40	Invoices for the deferred balance are generated every six months.	Split-payment agreements.
Annual	50	Invoices for the deferred balance are generated once per year.	Partial Upfront EAs billed yearly.
Custom	60	Hourly/Daily or other irregular cycles.	Irregular bridge contracts or non-standard terms.

3.3.16.3. Column ID

ContractCommitmentPaymentInterval

3.3.16.4. Display Name

Contract Commitment Payment Interval

3.3.16.5. Description

Represents the frequency by which a *contract commitment* is invoiced.

3.3.16.6. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.16.7. Version Introduced

1.4

3.3.17. Contract Commitment Payment Model

Contract Commitment Payment Model defines the financial settlement structure of a *contract commitment*. It identifies whether the financial obligation is settled via a single upfront payment, distributed recurring charges, or a combination of both over the contract commitment duration type.

Contract Commitment Payment Model has three possible values: **No Upfront**, **Partial Upfront**, and **All Upfront**.

- No Upfront denotes that the obligation is settled entirely through recurring charges with no initial payment.
- Partial Upfront denotes that the obligation is settled through a combination of an initial payment and recurring charges.
- All Upfront denotes that the obligation is settled via a single payment at the start of the duration.

3.3.17.1. Requirements

ContractCommitmentPaymentModel MUST adhere to the following requirements:

- ContractCommitmentPaymentModel MUST be of type String.
- ContractCommitmentPaymentModel MUST NOT be null.
- ContractCommitmentPaymentModel MUST be one of the allowed values.

3.3.17.2. Allowed Values

Value	Sort Order	Description	Typical Use Case
No Upfront	10	The obligation is settled entirely through deferred payment(s) (typically multiple recurring charges) with no initial payment.	Pay-as-you-go Savings Plans or monthly-billed SaaS.
Partial Upfront	20	The obligation is settled through a combination of an initial payment and deferred payment(s) (typically multiple recurring charges).	Hybrid RIs or EAs with a "Year 1" deposit plus installments.
All Upfront	30	The total obligation is settled via a single payment at the start of the duration.	High-discount RIs or multi-year contracts paid in full Day 1.

3.3.17.3. Implementation Guidance

- Settled via Upfront: Refer to the contract commitment period start for the cash event.
- Settled via Recurring Charges: Refer to the contract commitment payment interval to understand the frequency of those subsequent cash events.

3.3.17.4. Column ID

ContractCommitmentPaymentModel

3.3.17.5. Display Name

Contract Commitment Payment Model

3.3.17.6. Description

Defines the financial settlement structure of a *contract commitment*.

3.3.17.7. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.3.17.8. Version Introduced

1.4

3.3.18. Contract Commitment Payment Upfront Percentage

Contract Commitment Payment Upfront Percentage represents the portion of the total Contract Commitment Cost paid at the start of the duration of a *contract commitment*.

This column allows for precise financial modeling of "Partial Upfront" payment models, enabling FinOps practitioners and accounting professionals to distinguish between immediate cash outlays and deferred liabilities.

3.3.18.1. Requirements

ContractCommitmentPaymentUpfrontPercentage MUST adhere to the following requirements:

- ContractCommitmentPaymentUpfrontPercentage MUST be of type Decimal.
- ContractCommitmentPaymentUpfrontPercentage MUST conform to NumericFormat requirements.
- ContractCommitmentPaymentUpfrontPercentage MUST NOT be null.
- ContractCommitmentPaymentUpfrontPercentage MUST be a value between 0.0 and 1.0, inclusive.
- ContractCommitmentPaymentUpfrontPercentage MUST be 1.0 when ContractCommitmentPaymentModel is "All Upfront".
- ContractCommitmentPaymentUpfrontPercentage MUST be 0.0 when ContractCommitmentPaymentModel is "No Upfront".
- ContractCommitmentPaymentUpfrontPercentage MUST be greater than 0.0 and less than 1.0 when ContractCommitmentPaymentModel is "Partial Upfront".

3.3.18.2. Column ID

ContractCommitmentPaymentUpfrontPercentage

3.3.18.3. Display Name

Contract Commitment Payment Upfront Percentage

3.3.18.4. Description

Represents the portion of the total Contract Commitment Cost paid at the start of the duration of a *contract commitment*.

3.3.18.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Metric
Feature level	Conditional
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	0.0 to 1.0 (inclusive)

3.3.18.6. Version Introduced

1.4

3.3.19. Contract Commitment Period End

Contract Commitment Period End represents the *exclusive end bound* of a contract commitment period. For example, a time period where Contract Commitment Period Start is '2024-01-01T00:00:00Z' and Contract Commitment Period End is '2024-01-02T00:00:00Z' includes January 1 2024 since Contract Commitment Period Start represents the *inclusive start bound*, but does not include January 1 2025 since Contract Commitment Period End represents the *exclusive end bound*.

3.3.19.1. Requirements

ContractCommitmentPeriodEnd MUST adhere to the following requirements:

- ContractCommitmentPeriodEnd MUST be of type Date/Time.
- ContractCommitmentPeriodEnd MUST conform to DateTimeFormat requirements.
- ContractCommitmentPeriodEnd MUST NOT be null.
- ContractCommitmentPeriodEnd MUST be the *exclusive end bound* of the effective period of the *contract commitment*.

3.3.19.2. Column ID

ContractCommitmentPeriodEnd

3.3.19.3. Display Name

Contract Commitment Period End

3.3.19.4. Description

The *exclusive end bound* of a contract commitment period.

3.3.19.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.19.6. Version Introduced

1.3

3.3.20. Contract Commitment Period Start

Contract Commitment Period Start represents the *inclusive start bound* of a contract commitment period. For example, a time period where Contract Commitment Period Start is '2024-01-01T00:00:00Z' and Contract Commitment End is '2025-01-01T00:00:00Z' includes January 1 2024 since Contract Commitment Period Start represents the *inclusive start bound*, but does not include *charges* for January 2 2025 since Contract Commitment Period End represents the *exclusive end bound*.

3.3.20.1. Requirements

ContractCommitmentPeriodStart MUST adhere to the following requirements:

- ContractCommitmentPeriodStart MUST be of type Date/Time.
- ContractCommitmentPeriodStart MUST conform to DateTimeFormat requirements.
- ContractCommitmentPeriodStart MUST NOT be null.
- ContractCommitmentPeriodStart MUST be the *inclusive start bound* of the effective period of the *contract commitment*.

3.3.20.2. Column ID

ContractCommitmentPeriodStart

3.3.20.3. Display Name

Contract Commitment Period Start

3.3.20.4. Description

The *inclusive start bound* of a contract commitment period.

3.3.20.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.20.6. Version Introduced

1.3

3.3.21. Contract Commitment Quantity

Contract Commitment Quantity represents the amount associated with the *contract commitment*, denominated in a service-provider-defined Contract Commitment Unit. Contract Commitment Quantity is commonly used for monitoring the progress towards fulfilling contractual commitments that may facilitate discounts for *resources* or *services* as agreed between a provider and a customer.

3.3.21.1. Requirements

ContractCommitmentQuantity MUST adhere to the following requirements:

- ContractCommitmentQuantity MUST be of type Decimal.
- ContractCommitmentQuantity MUST conform to NumericFormat requirements.
- ContractCommitmentQuantity MUST adhere to the following nullability requirements:
    - ContractCommitmentQuantity MUST NOT be null when ContractCommitmentCategory is "Usage".

- ContractCommitmentQuantity MAY be null when ContractCommitmentCategory is "Spend".

3.3.21.2. Column ID

ContractCommitmentQuantity

3.3.21.3. Display Name

Contract Commitment Quantity

3.3.21.4. Description

The amount associated with the *contract commitment*.

3.3.21.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Metric
Feature level	Mandatory
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.3.21.6. Version Introduced

1.3

3.3.22. Contract Commitment Type

Contract Commitment Type is a service-provider-assigned name to identify the type of *contract commitment*. Contract Commitment Type is a readable display name and not a code. Contract Commitment Type is commonly used for displaying and aggregating the types of commitments the practitioner has made, stated in service-provider-specific terms.

3.3.22.1. Requirements

ContractCommitmentType MUST adhere to the following requirements:

- ContractCommitmentType MUST be of type String.
- ContractCommitmentType MUST conform to StringHandling requirements.
- ContractCommitmentType MUST NOT be null.
- ContractCommitmentType MUST be a consistent, readable display value.

3.3.22.2. Column ID

ContractCommitmentType

3.3.22.3. Display Name

Contract Commitment Type

3.3.22.4. Description

A service-provider-assigned name to identify the type of *contract commitment*.

3.3.22.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.3.22.6. Version Introduced

1.3

3.3.23. Contract Commitment Unit

The Contract Commitment Unit represents a service-provider-specified measurement unit for the amount declared in Contract Commitment Quantity. Contract Commitment Unit complements the Contract Commitment Quantity metric.

3.3.23.1. Requirements

ContractCommitmentUnit MUST adhere to the following requirements:

- ContractCommitmentUnit MUST be of type String.
- ContractCommitmentUnit MUST conform to StringHandling requirements.
- ContractCommitmentUnit SHOULD conform to UnitFormat requirements.
- ContractCommitmentUnit MUST adhere to the following nullability requirements:
  - ContractCommitmentUnit MUST be null when ContractCommitmentQuantity is null.
  - ContractCommitmentUnit MUST NOT be null when ContractCommitmentQuantity is not null.

3.3.23.2. Column ID

ContractCommitmentUnit

3.3.23.3. Display Name

Contract Commitment Unit

3.3.23.4. Description

A service-provider-specified measurement unit for the amount declared in Contract Commitment Quantity.

3.3.23.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	Unit Format recommended

3.3.23.6. Version Introduced

1.3

3.3.24. Contract ID

Contract ID is a service-provider-assigned identifier for a contract describing the agreed terms between a service provider and a customer. Contracts can include commitment to a certain amount of spend or usage over an agreed period of time.

3.3.24.1. Requirements

ContractId MUST adhere to the following requirements:

- ContractId MUST be of type String.
- ContractId MUST conform to StringHandling requirements.
- ContractId MUST NOT be null.
- When ContractId is not null, ContractId MUST adhere to the following requirements:
    - ContractId MUST be a unique identifier within the service provider.
    - ContractId SHOULD be a fully-qualified identifier.

3.3.24.2. Column ID

ContractId

3.3.24.3. Display Name

Contract ID

3.3.24.4. Description

A service-provider-assigned identifier for a contract describing the agreed terms between a service provider and a customer.

3.3.24.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String

Constraint	Value
Value format	<not specified>

3.3.24.6. Version Introduced

1.3

3.3.25. Contract Period End

Contract Period End represents the *exclusive end bound* of a contract period. For example, a time period where Contract Period Start is '2024-01-01T00:00:00Z' and Contract Period End is '2024-01-02T00:00:00Z' includes January 1 2024 since Contract Period Start represents the *inclusive start bound*, but does not include January 1 2025 since Contract Period End represents the *exclusive end bound*.

3.3.25.1. Requirements

ContractPeriodEnd MUST adhere to the following requirements:

- ContractPeriodEnd MUST be of type Date/Time.
- ContractPeriodEnd MUST conform to DateTimeFormat requirements.
- ContractPeriodEnd MUST NOT be null.
- ContractPeriodEnd MUST be the *exclusive end bound* of the effective period of the *contract*.

3.3.25.2. Column ID

ContractPeriodEnd

3.3.25.3. Display Name

Contract Period End

3.3.25.4. Description

The *exclusive end bound* of a contract period.

3.3.25.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.25.6. Version Introduced

1.3

3.3.26. Contract Period Start

Contract Period Start represents the *inclusive start bound* of a contract period. For example, a time period where Contract Period Start is '2024-01-01T00:00:00Z' and Contract Period End is '2025-01-01T00:00:00Z' includes January 1 2024 since Contract Period Start represents the *inclusive start bound*, but does not include January 2 2025 since Contract Period End represents the *exclusive end bound*.

3.3.26.1. Requirements

ContractPeriodStart MUST adhere to the following requirements:

- ContractPeriodStart MUST be of type Date/Time.
- ContractPeriodStart MUST conform to DateTimeFormat requirements.
- ContractPeriodStart MUST NOT be null.
- ContractPeriodStart MUST be the *inclusive start bound* of the effective period of the *contract*.

3.3.26.2. Column ID

ContractPeriodStart

3.3.26.3. Display Name

Contract Period Start

3.3.26.4. Description

The *inclusive start bound* of a contract period.

3.3.26.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.3.26.6. Version Introduced

1.3

3.3.27. Invoice Issuer Name

Invoice Issuer Name is the name of the entity responsible for issuing payable invoices for the *contract commitment*. It is commonly used for cost analysis, reconciliation, and reporting scenarios.

3.3.27.1. Requirements

InvoiceIssuerName MUST adhere to the following requirements:

- InvoiceIssuerName MUST be of type String.
- InvoiceIssuerName MUST conform to StringHandling requirements.
- InvoiceIssuerName MUST NOT be null.
- InvoiceIssuerName MUST represent the entity that issues invoices.

3.3.27.2. Column ID

InvoiceIssuerName

3.3.27.3. Display Name

Invoice Issuer Name

3.3.27.4. Description

The name of the entity responsible for invoicing for the *contract commitment*.

3.3.27.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.3.27.6. Version Introduced

1.4

3.3.28. Pricing Currency

Pricing Currency is the national or virtual currency denomination that a *contract commitment* was priced in. This is commonly used in scenarios where a commitment is negotiated in one currency but billed in another.

3.3.28.1. Requirements

PricingCurrency MUST adhere to the following requirements:

- PricingCurrency MUST be of type String.
- PricingCurrency MUST conform to StringHandling requirements.
- PricingCurrency MUST conform to CurrencyFormat requirements.
- PricingCurrency MUST NOT be null.

3.3.28.2. Column ID

PricingCurrency

3.3.28.3. Display Name

Pricing Currency

3.3.28.4. Description

The national or virtual currency denomination that the Contract Commitment Cost was priced in.

3.3.28.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	Currency Format

3.3.28.6. Version Introduced

1.4

3.3.29. Pricing Currency Contract Commitment Cost

Pricing Currency Contract Commitment Cost represents the monetary value of the *contract commitment* denominated in the *Pricing Currency*. This metric is used to track progress towards fulfilling contractual milestones using the original negotiated value, independent of currency exchange rate fluctuations.

3.3.29.1. Requirements

PricingCurrencyContractCommitmentCost MUST adhere to the following requirements:

- PricingCurrencyContractCommitmentCost MUST be of type Decimal.
- PricingCurrencyContractCommitmentCost MUST conform to NumericFormat requirements.
- PricingCurrencyContractCommitmentCost MUST adhere to the following nullability requirements:
  - PricingCurrencyContractCommitmentCost MUST NOT be null when ContractCommitmentCategory is "Spend" and PricingCurrency is provided.
  - PricingCurrencyContractCommitmentCost MAY be null when ContractCommitmentCategory is "Usage".
- PricingCurrencyContractCommitmentCost MUST be denominated in the PricingCurrency.

3.3.29.2. Column ID

PricingCurrencyContractCommitmentCost

3.3.29.3. Display Name

Pricing Currency Contract Commitment Cost

3.3.29.4. Description

The monetary value of the *contract commitment* in the Pricing Currency.

3.3.29.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Metric
Feature level	Conditional
Allows nulls	True
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.3.29.6. Version Introduced

1.4

3.3.30. Service Provider Name

Service Provider Name is the name of the entity that provides the resources or services available for usage or purchase. This entity is responsible for fulfilling the terms of the *contract commitment*, such as applying discounts, managing credit pools, or guaranteeing resource availability.

Notes:

- In marketplace scenarios, the Service Provider represents the seller of the commitment (e.g., Datadog, MongoDB) rather than the marketplace operator (e.g., AWS, Azure), unless the marketplace operator is the entity providing the specific commitment benefit.
- In reseller scenarios, if the commitment is made directly with a reseller for white-labeled services, the Service Provider is the reseller. Otherwise, it is the entity that produced the underlying services tied to the commitment.

3.3.30.1. Requirements

ServiceProviderName MUST adhere to the following requirements:

- ServiceProviderName MUST be of type String.
- ServiceProviderName MUST conform to StringHandling requirements.
- ServiceProviderName MUST NOT be null.

3.3.30.2. Column ID

ServiceProviderName

3.3.30.3. Display Name

Service Provider Name

3.3.30.4. Description

The name of the entity that provides the *contract commitment*.

3.3.30.5. Content Constraints

Constraint	Value
Dataset	Contract Commitment
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.3.30.6. Version Introduced

1.4

3.4. Invoice Detail

The Invoice Detail dataset is a transactional dataset that represents the financial record of *charges* as they appear on invoices provided by an *invoice issuer*. This dataset enables FinOps practitioners to perform financial reconciliation, tax reporting, and payment processing tasks. While the Cost and Usage dataset provides granular visibility into consumption, the Invoice Detail dataset ensures alignment with the physical or electronic billing documents.

Columns

Column	Column Type	Feature Level	Allows Nulls	Data Type
Billed Cost	Metric	Mandatory	False	Decimal
Billing Account ID	Dimension	Mandatory	False	String
Billing Currency	Dimension	Mandatory	False	String
Charge Category	Dimension	Mandatory	False	String
Billing Period End	Dimension	Mandatory	False	Date/Time
Billing Period Start	Dimension	Mandatory	False	Date/Time
Invoice Detail Created	Dimension	Mandatory	False	Date/Time
Invoice Detail Description	Dimension	Mandatory	True	String
Invoice Detail Grain	Dimension	Mandatory	True	JSON
Invoice Detail ID	Dimension	Mandatory	False	String
Invoice Detail Last Updated	Dimension	Mandatory	False	Date/Time
Invoice ID	Dimension	Mandatory	False	String
Invoice Issue Date	Dimension	Mandatory	True	Date/Time
Invoice Issue Status	Dimension	Mandatory	False	String
Invoice Issuer Name	Dimension	Mandatory	False	String
Payment Currency	Dimension	Conditional	False	String
Payment Currency Billed Cost	Metric	Conditional	False	Decimal
Payment Currency Invoice Detail ID	Dimension	Conditional	False	String
Payment Due Date	Dimension	Mandatory	True	Date/Time
Payment Terms	Dimension	Mandatory	False	String
Purchase Order Number	Dimension	Conditional	True	String
Reference Invoice ID	Dimension	Mandatory	False	String

Relationships

The Invoice Detail dataset can be joined to the Cost and Usage dataset through Invoice Issuer Name, Invoice ID, and (optionally) Invoice Detail ID. Take note: one or both datasets will need to be aggregated in order to facilitate any comparison.

The timing of Invoice ID and Invoice Detail ID availability in Cost and Usage varies across data generators. Some data generators populate these values while the *billing period* is still open, while others do not populate them until after the *billing period* is closed and invoices have been issued.

For more information, see the Invoice Reconciliation supported feature.

Dataset A	Dataset A Column	Dataset B	Dataset B Column
Invoice Detail	Invoice Issuer Name and Invoice ID	Cost and Usage	Invoice Issuer Name and Invoice ID
Invoice Detail	Invoice Issuer Name, Invoice ID, and Invoice Detail ID	Cost and Usage	Invoice Issuer Name, Invoice ID, and Invoice Detail ID

Requirements

InvoiceDetail MUST adhere to the following requirements:

- InvoiceDetail MUST be present when the invoice issuer supports payable invoices.
- The presence of columns in InvoiceDetail MUST adhere to the following requirements:
  - InvoiceDetail MUST include BilledCost.
  - InvoiceDetail MUST include BillingAccountId.
  - InvoiceDetail MUST include BillingCurrency.
  - InvoiceDetail MUST include BillingPeriodEnd.
  - InvoiceDetail MUST include BillingPeriodStart.
  - InvoiceDetail MUST include ChargeCategory.
  - InvoiceDetail MUST include InvoiceDetailCreated.
  - InvoiceDetail MUST include InvoiceDetailDescription.
  - InvoiceDetail MUST include InvoiceDetailGrain.
  - InvoiceDetail MUST include InvoiceDetailId.
  - InvoiceDetail MUST include InvoiceDetailLastUpdated.
  - InvoiceDetail MUST include InvoiceId.
  - InvoiceDetail MUST include InvoiceIssueDate.
  - InvoiceDetail MUST include InvoiceIssueStatus.
  - InvoiceDetail MUST include InvoiceIssuerName.
  - InvoiceDetail MUST include PaymentCurrency when the invoice issuer supports billing and payment in different currencies.
  - InvoiceDetail MUST include PaymentCurrencyBilledCost when the invoice issuer supports billing and payment in different currencies.
  - InvoiceDetail MUST include PaymentCurrencyInvoiceDetailId when the invoice issuer represents billing currency and payment currency at different aggregation levels on payable invoices.
  - InvoiceDetail MUST include PaymentDueDate.
  - InvoiceDetail MUST include PaymentTerms.
  - InvoiceDetail MUST include PurchaseOrderNumber when the invoice issuer supports customer input of purchase order numbers.
  - InvoiceDetail MUST include ReferenceInvoiceId.
  - InvoiceDetail MUST include *custom columns* to represent any monetary metric that appears on an invoice issued to a BillingAccountId when there is no equivalent *FOCUS column*.
- InvoiceDetail MUST conform to CorrectionHandling requirements.
- InvoiceDetail MUST conform to DatasetCompleteness requirements.
- InvoiceDetail MUST conform to DatasetConfiguration requirements.
- InvoiceDetail MUST conform to DeliveryHandling requirements.
- InvoiceDetail MUST represent all invoice line items with a non-zero BilledCost on any invoice associated with a BillingAccountId.
- InvoiceDetail *FOCUS columns* MUST conform to FocusColumnHandling requirements.
- InvoiceDetail *FOCUS columns* MUST conform to NullHandling requirements.
- InvoiceDetail *custom columns* MUST conform to CustomColumnHandling requirements.
- InvoiceDetail documentation MUST adhere to the following requirements:
  - InvoiceDetail documentation MUST specify how InvoiceDetail records correspond to invoice line items.
  - InvoiceDetail documentation MUST specify whether invoice line items with BilledCost of 0 are excluded from InvoiceDetail.
  - InvoiceDetail documentation MUST describe how columns in the CostAndUsage and InvoiceDetail *dataset instances* represent the invoice issuer's *invoice reconciliation* process.
  - InvoiceDetail documentation MUST be freely accessible to FOCUS consumers.

Dataset ID

InvoiceDetail

Display Name

Description

The financial record of *charges* as they appear on invoices provided by an invoice issuer.

Version Introduced

1.4

3.4.1. Billed Cost

Billed Cost represents the cost of a *charge* as invoiced by the invoice issuer in a given *billing period*.

For all *charges*, Billed Cost reflects all applicable pricing adjustments (e.g., reduced pricing from *negotiated discounts* or *commitment discounts*). For purchase *charges*, Billed Cost includes any portion invoiced in the given *billing period*. For usage *charges*, Billed Cost excludes any portion *covered* by related purchase *charges* (e.g., *covering charges* such as *commitments*, prepayments, or marketplace purchases), regardless of when those related *charges* are invoiced.

Billed Cost is denominated in the Billing Currency. Billed Cost is commonly used to support FinOps activities, including invoice reconciliation, *cash-based* forecasting, budgeting, and cost allocation.

3.4.1.1. Requirements

BilledCost MUST adhere to the following requirements:

- BilledCost MUST be of type Decimal.
- BilledCost MUST conform to NumericFormat requirements.
- BilledCost MUST NOT be null.
- BilledCost MUST be denominated in the BillingCurrency.
- BilledCost MUST reflect all applicable pricing adjustments, including but not limited to *negotiated discounts*, *commitment discounts*, and other applicable discount programs.
- BilledCost MUST NOT include any portion of a *covered charge* that is offset by a *covering charge*.
- BilledCost MUST be 0 for *charges* that are fully *covered* by one or more *covering charges*.
- The sum of BilledCost for a given InvoiceDetailId, InvoiceId, and InvoiceIssuerName MUST be equal to the payable amount provided in the corresponding entries on the issued invoice when InvoiceIssueStatus is "Issued".
- When comparing BilledCost aggregated by InvoiceId and InvoiceIssuerName with CostAndUsage.BilledCost aggregated by CostAndUsage.InvoiceId and CostAndUsage.InvoiceIssuerName, BilledCost MUST adhere to the following requirements:
  - When ChargeCategory is not "Tax" and InvoiceIssueStatus is not "Open", the sum of BilledCost MUST NOT differ from the sum of CostAndUsage.BilledCost by more than `MAX(100 × Subunit, (SQRT(Rows) × 0.5) × Subunit)` as defined in Rounding Variance Tolerance.
  - When ChargeCategory is "Tax" or InvoiceIssueStatus is "Open", the sum of BilledCost MAY differ from the sum of CostAndUsage.BilledCost.
- When comparing BilledCost aggregated by InvoiceDetailId, InvoiceId, and InvoiceIssuerName with CostAndUsage.BilledCost aggregated by CostAndUsage.InvoiceDetailId, CostAndUsage.InvoiceId, and CostAndUsage.InvoiceIssuerName, BilledCost MUST adhere to the following requirements:
  - When InvoiceIssueStatus is not "Open", the sum of BilledCost MUST NOT differ from the sum of CostAndUsage.BilledCost by more than `MAX(100 × Subunit, (SQRT(Rows) × 0.5) × Subunit)` as defined in Rounding Variance Tolerance.
  - When InvoiceIssueStatus is "Open", the sum of BilledCost MAY differ from the sum of CostAndUsage.BilledCost.

3.4.1.2. Implementation Guidance

3.4.1.2.1. Handling Rounding Discrepancies

When validating InvoiceDetail.BilledCost against CostAndUsage.BilledCost, the totals may not be perfectly equal due to precision differences (e.g., 6 decimals in CostAndUsage vs. 2 decimals in InvoiceDetail). The requirement allows for a maximum rounding error based on the statistical probability of rounding variance, which grows with the square root of the row count. For more information, see the Rounding Variance Tolerance appendix entry.

3.4.1.3. Column ID

BilledCost

3.4.1.4. Display Name

Billed Cost

3.4.1.5. Description

Cost of a *charge* as invoiced by the *invoice issuer* in a given *billing period*.

3.4.1.6. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Metric
Feature level	Mandatory
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.4.1.7. Version Introduced

1.4

3.4.2. Billing Account ID

Billing Account ID is an invoice-issuer-assigned identifier for a *billing account*. *Billing accounts* are commonly used for scenarios like grouping based on organizational constructs, invoice reconciliation and cost allocation strategies.

3.4.2.1. Requirements

BillingAccountId MUST adhere to the following requirements:

- BillingAccountId MUST be of type String.
- BillingAccountId MUST conform to StringHandling requirements.
- BillingAccountId MUST NOT be null.
- BillingAccountId MUST be a unique identifier within an *invoice issuer*.
- BillingAccountId SHOULD be a fully-qualified identifier.

See Appendix: Grouping constructs for resources or services for details and examples of the different grouping constructs supported by FOCUS.

3.4.2.2. Column ID

BillingAccountId

3.4.2.3. Display Name

Billing Account ID

3.4.2.4. Description

The identifier assigned to a *billing account* by the invoice issuer.

3.4.2.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.2.6. Version Introduced

1.4

3.4.3. Billing Currency

Billing Currency is an identifier that represents the currency that a *charge* for *resources* or *services* was billed in. Billing Currency is commonly used in scenarios where costs need to be grouped or aggregated.

3.4.3.1. Requirements

BillingCurrency MUST adhere to the following requirements:

- BillingCurrency MUST be of type String.
- BillingCurrency MUST conform to StringHandling requirements.
- BillingCurrency MUST conform to CurrencyFormat requirements.
- BillingCurrency MUST NOT be null.
- BillingCurrency MUST match the currency used in the invoice generated by the *invoice issuer*.
- BillingCurrency MUST be expressed in *national currency* (e.g., USD, EUR).

3.4.3.2. Column ID

BillingCurrency

3.4.3.3. Display Name

Billing Currency

3.4.3.4. Description

Represents the currency that a *charge* was billed in.

3.4.3.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Currency Format

3.4.3.6. Version Introduced

1.4

3.4.4. Billing Period End

Billing Period End represents the *exclusive end bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes *charges* for January since Billing Period Start represents the *inclusive start bound*, but does not include *charges* for February since Billing Period End represents the *exclusive end bound*.

3.4.4.1. Requirements

BillingPeriodEnd MUST adhere to the following requirements:

- BillingPeriodEnd MUST be of type Date/Time.
- BillingPeriodEnd MUST conform to DateTimeFormat requirements.
- BillingPeriodEnd MUST NOT be null.
- BillingPeriodEnd MUST be the *exclusive end bound* of the *billing period*.
- BillingPeriodEnd for a given InvoiceId and InvoiceIssuerName MUST match CostAndUsage.BillingPeriodEnd for the same CostAndUsage.InvoiceId and CostAndUsage.InvoiceIssuerName.

3.4.4.2. Column ID

BillingPeriodEnd

3.4.4.3. Display Name

Billing Period End

3.4.4.4. Description

The *exclusive end bound* of a *billing period*.

3.4.4.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail

Constraint	Value
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.4.4.6. Version Introduced

1.4

3.4.5. Billing Period Start

Billing Period Start represents the *inclusive start bound* of a *billing period*. For example, a time period where Billing Period Start is '2024-01-01T00:00:00Z' and Billing Period End is '2024-02-01T00:00:00Z' includes *charges* for January since Billing Period Start represents the *inclusive start bound*, but does not include *charges* for February since BillingPeriodEnd represents the *exclusive end bound*.

3.4.5.1. Requirements

BillingPeriodStart MUST adhere to the following requirements:

- BillingPeriodStart MUST be of type Date/Time.
- BillingPeriodStart MUST conform to DateTimeFormat requirements.
- BillingPeriodStart MUST NOT be null.
- BillingPeriodStart MUST be the *inclusive start bound* of the *billing period*.
- BillingPeriodStart for a given InvoiceId and InvoiceIssuerName MUST match CostAndUsage.BillingPeriodStart for the same CostAndUsage.InvoiceId and CostAndUsage.InvoiceIssuerName.

3.4.5.2. Column ID

BillingPeriodStart

3.4.5.3. Display Name

Billing Period Start

3.4.5.4. Description

The *inclusive start bound* of a *billing period*.

3.4.5.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time

Constraint	Value
Value format	Date/Time Format

3.4.5.6. Version Introduced

1.4

3.4.6. Charge Category

Charge Category represents the highest-level classification of a *charge* based on the nature of how it is billed. Charge Category is commonly used to identify and distinguish between types of *charges* that may require different handling.

3.4.6.1. Requirements

ChargeCategory MUST adhere to the following requirements:

- ChargeCategory MUST be of type String.
- ChargeCategory MUST NOT be null.
- ChargeCategory MUST be one of the allowed values.
- When the *charge* does not aggregate multiple classifications, ChargeCategory MUST adhere to the following requirements:
  - ChargeCategory MUST be "Usage" when the *charge* represents consumption of a service or resource.
  - ChargeCategory MUST be "Purchase" when the *charge* represents acquisition of a service, resource, or *commitment*.
  - ChargeCategory MUST be "Tax" when the *charge* represents taxes levied by the relevant authorities.
  - ChargeCategory MUST be "Credit" when the *charge* represents a financial incentive or allowance unrelated to other *charges*.
  - ChargeCategory MUST be "Adjustment" when the *charge* represents a billing modification that does not fall into other ChargeCategories.
- When the *charge* aggregates multiple classifications, ChargeCategory MUST adhere to the following requirements:
  - ChargeCategory MAY be "Usage" when the record aggregates *charges* across multiple allowed values other than "Tax" (e.g., aggregation of "Usage" and "Credit" is allowed, but not "Usage" and "Tax").

3.4.6.2. Allowed Values

Value	Description
Usage	Positive or negative *charges* based on the quantity of a service or resource that was consumed over a given period of time including refunds.
Purchase	Positive or negative *charges* for the acquisition of a service or resource bought upfront or on a recurring basis including refunds.
Tax	Positive or negative applicable taxes that are levied by the relevant authorities including refunds. Tax *charges* may vary depending on factors such as the location, jurisdiction, and local or federal regulations.
Credit	Positive or negative *charges* granted by the service provider for various scenarios (e.g., promotional credits, corrections to promotional credits).
Adjustment	Positive or negative *charges* the service provider applies that do not fall into other category values.

3.4.6.3. Implementation Guidance

Unlike the CostAndUsage dataset, which requires strict categorization per row, InvoiceDetail records are designed to align with physical or electronic invoice line items. When the invoice does not provide an explicit charge category for a line item, a single line item may inherently represent a mix of charge categories (e.g., combining base usage and promotional credits). To accurately reflect the issued invoice without breaking schema validation, the `InvoiceDetail` dataset permits the "Usage" category to represent an aggregate of multiple charge categories, provided that "Tax" charges are kept distinctly separate.

3.4.6.4. Column ID

ChargeCategory

3.4.6.5. Display Name

Charge Category

3.4.6.6. Description

Represents the highest-level classification of a *charge* based on the nature of how it is billed.

3.4.6.7. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.4.6.8. Version Introduced

1.4

3.4.7. Invoice Detail Created

Invoice Detail Created is the timestamp when the Invoice Detail record was first created. This timestamp facilitates auditability of the charge and invoice lifecycle, allowing the FinOps practitioner to distinguish between the time of service consumption and the time of financial record generation.

3.4.7.1. Requirements

InvoiceDetailCreated MUST adhere to the following requirements:

- InvoiceDetailCreated MUST be of type Date/Time.
- InvoiceDetailCreated MUST conform to DateTimeFormat requirements.
- InvoiceDetailCreated MUST NOT be null.
- InvoiceDetailCreated MUST represent the moment in time the Invoice Detail record was instantiated.
- InvoiceDetailCreated for a given BillingPeriodStart and InvoiceIssuerName MUST be earlier than or equal to BillingPeriod.BillingPeriodLastUpdated for the same BillingPeriod.BillingPeriodStart and BillingPeriod.InvoiceIssuerName when BillingPeriod.BillingPeriodStatus is "Closed".

3.4.7.2. Column ID

InvoiceDetailCreated

3.4.7.3. Display Name

Invoice Detail Created

3.4.7.4. Description

The timestamp when the Invoice Detail record was first created.

3.4.7.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.4.7.6. Version Introduced

1.4

3.4.8. Invoice Detail Description

Invoice Detail Description is the invoice-issuer-provided description of an invoice line item. This description provides context for the charge as it appears on the invoice, often summarizing the service, resource, or period covered by that specific line item to assist in human-readable reconciliation.

3.4.8.1. Requirements

InvoiceDetailDescription MUST adhere to the following requirements:

- InvoiceDetailDescription MUST be of type String.
- InvoiceDetailDescription MUST conform to StringHandling requirements.
- InvoiceDetailDescription SHOULD NOT be null.
- InvoiceDetailDescription maximum length SHOULD be provided in the corresponding FOCUS Metadata Schema.
- InvoiceDetailDescription MUST describe the *charges* represented by the InvoiceDetailId.

3.4.8.2. Column ID

InvoiceDetailDescription

3.4.8.3. Display Name

Invoice Detail Description

3.4.8.4. Description

The invoice-issuer-provided description of an invoice line item.

3.4.8.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	String
Value format	<not specified>

3.4.8.6. Version Introduced

1.4

3.4.9. Invoice Detail Grain

Invoice Detail Grain represents the set of key-value pairs that defines the granularity of an invoice line item. The grain may vary from one record to the next, both within a single invoice and across *invoice issuers*, and key-value pairs are used instead of separate columns to accommodate this variability.

This gives FinOps practitioners a single point of reference for all possible Invoice Detail granularities (e.g., SKU, service, resource, custom dimension), supporting downstream data transformations such as reconciliation and cost allocation.

3.4.9.1. Requirements

InvoiceDetailGrain MUST adhere to the following requirements:

- InvoiceDetailGrain MUST be of type JSON Object (serialized as a String where necessary).
- InvoiceDetailGrain MUST conform to StringHandling requirements.
- InvoiceDetailGrain MUST conform to KeyValueFormat requirements.
- InvoiceDetailGrain MUST NOT be null when one or more properties uniquely define the granularity of the invoice line item.
- InvoiceDetailGrain MUST contain the set of all properties that uniquely define the granularity of the invoice line item.
- InvoiceDetailGrain SHOULD use the applicable FOCUS-defined Invoice Detail Grain properties listed below to represent the granularity of the invoice line item.
- InvoiceDetailGrain MUST include all custom Invoice Detail Grain properties that are applicable to the granularity of the invoice line item when there is no equivalent FOCUS-defined property.
- InvoiceDetailGrain property keys SHOULD conform to PascalCase format.
- InvoiceDetailGrain property keys MUST begin with the string "x_" unless it is a FOCUS-defined property.
- FOCUS-defined InvoiceDetailGrain properties MUST adhere to the following requirements:
    - Property key MUST match the spelling and casing specified for the FOCUS-defined property.
    - Property value MUST be of the type specified for that property.

3.4.9.2. FOCUS-Defined Properties

The following keys should be used when applicable when a relevant concept is represented on an invoice. For more information, see the relevant Cost and Usage column entry.

- Contract ID (element of ContractApplied)
- Region ID
- Resource ID
- Resource Type
- Service Name
- SKU ID
- SKU Meter
- SKU Price ID
- Sub Account ID

3.4.9.3. Examples

```
{
  "ServiceName": "Elastic Cloud Server",
  "ResourceType": "Cloud Host",
  "x_BillingMode": "Pay-per-Use"
}
```

3.4.9.4. Column ID

InvoiceDetailGrain

3.4.9.5. Display Name

Invoice Detail Grain

3.4.9.6. Description

The set of key-value pairs that defines the granularity of the invoice line item.

3.4.9.7. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	JSON
Value format	Key-Value Format

3.4.9.8. Version Introduced

1.4

3.4.10. Invoice Detail ID

Invoice Detail ID is the invoice-issuer-assigned identifier for an Invoice Detail record encapsulating charges in the corresponding billing period for a given billing account. This identifier allows FinOps practitioners to map specific line items from an invoice to the granular charge data, facilitating detailed reconciliation and auditability.

3.4.10.1. Requirements

InvoiceDetailId MUST adhere to the following requirements:

- InvoiceDetailId MUST be of type String.
- InvoiceDetailId MUST conform to StringHandling requirements.
- InvoiceDetailId MUST NOT be null.
- InvoiceDetailId MUST uniquely identify a record within a given InvoiceId.

3.4.10.2. Column ID

InvoiceDetailId

3.4.10.3. Display Name

Invoice Detail ID

3.4.10.4. Description

The invoice-issuer-assigned identifier for an Invoice Detail record encapsulating charges in the corresponding billing period for a given billing account.

3.4.10.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.10.6. Version Introduced

1.4

3.4.11. Invoice Detail Last Updated

Invoice Detail Last Updated is the timestamp when theInvoice Detail record was last updated. This timestamp helps FinOps practitioners ensure that they are working with the most current version of a record, particularly if corrections or status changes have been applied to the record after its initial creation.

3.4.11.1. Requirements

InvoiceDetailLastUpdated MUST adhere to the following requirements:

- InvoiceDetailLastUpdated MUST be of type Date/Time.
- InvoiceDetailLastUpdated MUST conform to DateTimeFormat requirements.
- InvoiceDetailLastUpdated MUST NOT be null.
- InvoiceDetailLastUpdated MUST represent the most recent moment in time when any column value of the record identified by InvoiceDetailId was created or modified.
- InvoiceDetailLastUpdated MUST be greater than or equal to InvoiceDetailCreated.

3.4.11.2. Column ID

InvoiceDetailLastUpdated

3.4.11.3. Display Name

3.4.11.4. Description

The timestamp when the Invoice Detail record was last updated.

3.4.11.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

3.4.11.6. Version Introduced

1.4

3.4.12. Invoice ID

Invoice ID is an invoice-issuer-assigned identifier for an invoice encapsulating *charges* in the corresponding *billing period* for a given *billing account*. Invoices are commonly used for scenarios like tracking billing transactions, facilitating payment processes and for performing invoice reconciliation between *charges* and billing periods.

3.4.12.1. Requirements

InvoiceId MUST adhere to the following requirements:

- InvoiceId MUST be of type String.
- InvoiceId MUST conform to StringHandling requirements.
- InvoiceId MUST NOT be null.
- InvoiceId MAY be generated prior to an invoice being issued.
- InvoiceId MUST uniquely identify the invoice as provided by the *invoice issuer*.

3.4.12.2. Column ID

InvoiceId

3.4.12.3. Display Name

Invoice ID

3.4.12.4. Description

The invoice-issuer-assigned identifier for an invoice encapsulating *charges* in the corresponding billing period for a given billing account.

3.4.12.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.12.6. Version Introduced

1.4

3.4.13. Invoice Issue Date

Invoice Issue Date is the date the invoice was issued by the *invoice issuer*. This date serves as the official point of record for the billing document, determining the beginning of payment terms and providing a key reference point for financial audits and period closing processes.

3.4.13.1. Requirements

InvoiceIssueDate MUST adhere to the following requirements:

- InvoiceIssueDate MUST be of type Date/Time.
- InvoiceIssueDate MUST conform to DateTimeFormat requirements.
- InvoiceIssueDate MAY be null.
- InvoiceIssueDate MUST represent the official date of issuance for the corresponding InvoiceId.

3.4.13.2. Column ID

InvoiceIssueDate

3.4.13.3. Display Name

Invoice Issue Date

3.4.13.4. Description

The date the invoice was issued by the invoice issuer.

3.4.13.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	Date/Time
Value format	Date/Time Format

3.4.13.6. Version Introduced

1.4

3.4.14. Invoice Issuer Name

Invoice Issuer Name is the name of the entity responsible for issuing payable invoices for the *resources* or *services* consumed. It is commonly used for cost analysis and reporting scenarios.

3.4.14.1. Requirements

InvoiceIssuerName MUST adhere to the following requirements:

- InvoiceIssuerName MUST be of type String.
- InvoiceIssuerName MUST conform to StringHandling requirements.
- InvoiceIssuerName MUST NOT be null.
- InvoiceIssuerName MUST represent the entity that issues invoices.

See Appendix: Participating Entity Identification Examples section for examples of Invoice Issuer Name values across various use case scenarios.

3.4.14.2. Column ID

InvoiceIssuerName

3.4.14.3. Display Name

Invoice Issuer Name

3.4.14.4. Description

The name of the entity responsible for invoicing for the *resources* or *services* consumed.

3.4.14.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.14.6. Version Introduced

1.4

3.4.15. Invoice Issue Status

Invoice Issue Status indicates the publication state of the invoice and the reliability of its associated delivered Cost and Usage

and Invoice Detail data. It distinguishes between provisional data that is subject to change, invoices that have been formally issued as valid financial obligations with finalized associated data, and invoices that have been explicitly retracted.

3.4.15.1. Requirements

InvoiceIssueStatus MUST adhere to the following requirements:

- InvoiceIssueStatus MUST be of type String.
- InvoiceIssueStatus MUST conform to StringHandling requirements.
- InvoiceIssueStatus MUST NOT be null.
- InvoiceIssueStatus MUST be one of the allowed values.
- InvoiceIssueStatus MUST represent the current publication state of the invoice.
- InvoiceIssueStatus MUST NOT be "Open" following a previous status of "Issued", except when explicitly requested or approved by the customer.

3.4.15.2. Allowed Values

Value	Description
Open	The invoice is provisional and subject to change. It is not a valid financial obligation and the associated delivered data is preliminary.
Issued	The invoice has been formally issued by the provider. It represents a valid financial obligation with finalized associated data.
Voided	The invoice was previously issued but has been retracted or nullified. It is not a valid financial obligation.

3.4.15.3. Implementation Guidance

The transition from "Open" to "Issued" typically signifies that an invoice has been finalized, invoice reconciliation has been performed, and the delivered data is accurate. However, when the delivered data is found to be inaccurate or incomplete, it may be necessary to apply corrections to records associated with the issued invoice.

If needed, a previously issued invoice may be reopened to apply such corrections, but this transition from "Issued" to "Open" must be explicitly requested or approved by the customer to maintain auditability.

Corrections to underlying records that do not impact invoice reconciliation are allowed regardless of Invoice Issue Status, but may reduce auditability and traceability or affect downstream processes (e.g., cost allocation, chargeback, reporting).

FinOps tools and reporting engines should be designed to detect Invoice Issue Status transitions and corrections to records associated with issued invoices, and trigger updates to downstream processes to ensure financial accuracy.

For more information, please see the Invoice and Billing Period Handling appendix and the Correction Handling attribute.

3.4.15.4. Column ID

InvoiceIssueStatus

3.4.15.5. Display Name

Invoice Issue Status

3.4.15.6. Description

The publication state of the invoice and the reliability of its associated delivered data, indicating if it is provisional ("Open"), issued ("Issued"), or voided ("Voided").

3.4.15.7. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	Allowed values

3.4.15.8. Version Introduced

1.4

3.4.16. Payment Currency

Payment Currency represents the currency in which the *invoice issuer* requires settlement. This is the currency of the financial obligation created by the invoice, which may differ from the Billing Currency and/or the source currency of the payer's funds or bank account. Payment Currency allows FinOps practitioners to track settlement obligations and foreign exchange impacts.

3.4.16.1. Requirements

PaymentCurrency MUST adhere to the following requirements:

- PaymentCurrency MUST be of type String.
- PaymentCurrency MUST conform to StringHandling requirements.
- PaymentCurrency MUST NOT be null.
- PaymentCurrency MUST represent the currency in which the invoice payment was made or expected to be made.
- PaymentCurrency MUST be expressed in *national currency* (e.g., USD, EUR).

3.4.16.2. Column ID

PaymentCurrency

3.4.16.3. Display Name

Payment Currency

3.4.16.4. Description

The currency in which the invoice is paid.

3.4.16.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Conditional
Allows nulls	False
Data type	String

Constraint	Value
Value format	Currency Format

3.4.16.6. Version Introduced

1.4

3.4.17. Payment Currency Billed Cost

Payment Currency Billed Cost represents the Billed Cost as expressed in Payment Currency. This metric is essential for organizations that need to reconcile their financial records in the currency used for actual settlement, especially when it differs from the currency used for initial billing.

3.4.17.1. Requirements

PaymentCurrencyBilledCost MUST adhere to the following requirements:

- PaymentCurrencyBilledCost MUST be of type Decimal.
- PaymentCurrencyBilledCost MUST conform to NumericFormat requirements.
- PaymentCurrencyBilledCost MUST NOT be null.
- PaymentCurrencyBilledCost MUST be denominated in the PaymentCurrency.
- PaymentCurrencyBilledCost MUST be the PaymentCurrency-denominated equivalent of BilledCost.
- PaymentCurrencyBilledCost MAY be non-zero while BilledCost is 0 when PaymentCurrencyBilledCost represents the aggregation of BilledCost amounts (denominated in PaymentCurrency) stated in other records.
- PaymentCurrencyBilledCost MAY be 0 while BilledCost is non-zero when BilledCost (denominated in PaymentCurrency) is represented in a separate aggregate record.

3.4.17.2. Examples

3.4.17.2.1. Example 1: Consistent Grain

In this scenario, the *invoice issuer* performs currency conversion at the individual line-item level. The grain of the payment currency matches the grain of the billing currency exactly.

- **Billing Currency:** USD
- **Payment Currency:** EUR
- **Exchange Rate:** 1.00 USD = 0.92 EUR

Column	Value
InvoiceDetailId	ID-001
ChargeCategory	Usage
BillingCurrency	USD
BilledCost	100.00
PaymentCurrency	EUR
PaymentCurrencyBilledCost	92.00
PaymentCurrencyInvoiceDetailId	ID-001

> **Note:** Because the conversion is 1:1, the `PaymentCurrencyInvoiceDetailId` points to the record's own `InvoiceDetailId`.

3.4.17.2.2. Example 2: Divergent Grain

In this scenario, the invoice issuer tracks usage in the billing currency at a granular level but represents that cost in the

payment currency as a separate aggregate record.

- **Billing Currency:** USD
- **Payment Currency:** EUR
- **Effective Exchange Rate:** 1.00 USD = 0.92 EUR

Column	A-101	A-102	Z-999
InvoiceDetailId	A-101	A-102	Z-999
InvoiceDetailDescription	Compute Instance A	Compute Instance B	Usage in Payment Currency
BilledCost	45.00	55.00	0.00
PaymentCurrencyBilledCost	0.00	0.00	92.00
PaymentCurrencyInvoiceDetailId	Z-999	Z-999	Z-999

Logic Breakdown:

- **Rows A-101 & A-102:** These are "child" records. Their `PaymentCurrencyBilledCost` is 0, so they provide a pointer in `PaymentCurrencyInvoiceDetailId` to Row **Z-999**, where the financial settlement value is stored.
- **Row Z-999:** This is the "parent" record. It aggregates the costs of the children. To identify itself as the root of this conversion, its `PaymentCurrencyInvoiceDetailId` matches its own `InvoiceDetailId`.
- **Reconciliation:** A practitioner can now sum all `BilledCost` values where `PaymentCurrencyInvoiceDetailId` is **Z-999** to verify that the $100.00 total is equal to the 92.00 EUR settlement using the expected exchange rate.

3.4.17.3. Column ID

PaymentCurrencyBilledCost

3.4.17.4. Display Name

Payment Currency Billed Cost

3.4.17.5. Description

The Billed Cost as expressed in Payment Currency.

3.4.17.6. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Metric
Feature level	Conditional
Allows nulls	False
Data type	Decimal
Value format	Numeric Format
Number range	Any valid decimal value

3.4.17.7. Version Introduced

1.4

3.4.18. Payment Currency Invoice Detail ID

Payment Currency Invoice Detail ID is a reference to the Invoice Detail ID of the record where the Payment Currency Billed

Cost for the current row is aggregated. This identifier enables practitioners to explicitly link granular usage records to their corresponding aggregate records stated in their chosen currency for settlement, ensuring accurate reconciliation across divergent grains.

3.4.18.1. Requirements

PaymentCurrencyInvoiceDetailId MUST adhere to the following requirements:

- PaymentCurrencyInvoiceDetailId MUST be of type String.
- PaymentCurrencyInvoiceDetailId MUST conform to StringHandling requirements.
- PaymentCurrencyInvoiceDetailId MUST NOT be null.
- PaymentCurrencyInvoiceDetailId MUST match the InvoiceDetailId of the record representing the PaymentCurrencyBilledCost aggregation for the current row.
- PaymentCurrencyInvoiceDetailId MUST match InvoiceDetailId of the current record when PaymentCurrencyBilledCost is non-zero.

3.4.18.2. Column ID

PaymentCurrencyInvoiceDetailId

3.4.18.3. Display Name

Payment Currency Invoice Detail ID

3.4.18.4. Description

The identifier linking a granular record to the specific Invoice Detail record where its Payment Currency Billed Cost is represented or aggregated.

3.4.18.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.18.6. Version Introduced

1.4

3.4.19. Payment Due Date

Payment Due Date is the date by which the payment for an invoice is expected to be received by the *invoice issuer*. This date is used by FinOps practitioners and finance teams to manage cash flow, prioritize payments, and avoid late fees or service interruptions.

3.4.19.1. Requirements

PaymentDueDate MUST adhere to the following requirements:

- PaymentDueDate MUST be of type Date/Time.
- PaymentDueDate MUST conform to DateTimeFormat requirements.
- PaymentDueDate MAY be null.
- PaymentDueDate MUST be the date specified by the invoice issuer as the deadline for payment for the corresponding InvoiceId.

3.4.19.2. Column ID

PaymentDueDate

3.4.19.3. Display Name

Payment Due Date

3.4.19.4. Description

The date by which the payment for an invoice is expected to be received by the invoice issuer.

3.4.19.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	True
Data type	Date/Time
Value format	Date/Time Format

3.4.19.6. Version Introduced

1.4

3.4.20. Payment Terms

Payment Terms represents the *terms* (typically focused on timeframe) by which the *invoice issuer* expects to receive payment for an invoice. These terms define the agreed-upon period for settling the invoice, helping both the provider and the customer manage financial expectations and payment schedules.

3.4.20.1. Requirements

PaymentTerms MUST adhere to the following requirements:

- PaymentTerms MUST be of type String.
- PaymentTerms MUST conform to StringHandling requirements.
- PaymentTerms MUST NOT be null.
- PaymentTerms MUST represent the payment terms (e.g., "Net 30") as defined on the corresponding invoice.

3.4.20.2. Column ID

PaymentTerms

3.4.20.3. Display Name

Payment Terms

3.4.20.4. Description

The terms (typically focused on timeframe) by which the invoice issuer expects to receive payment for an invoice.

3.4.20.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

3.4.20.6. Version Introduced

1.4

3.4.21. Purchase Order Number

Purchase Order Number is the unique customer-issued identifier for tracking the lifecycle of a purchase. This identifier is typically provided by the customer to the *invoice issuer* to ensure that charges are mapped to specific internal procurement records or purchase orders.

3.4.21.1. Requirements

PurchaseOrderNumber MUST adhere to the following requirements:

- PurchaseOrderNumber MUST be of type String.
- PurchaseOrderNumber MUST conform to StringHandling requirements.
- PurchaseOrderNumber MAY be null.
- PurchaseOrderNumber MUST represent the identifier used by the customer to uniquely identify the purchase order responsible for the charge.

3.4.21.2. Column ID

PurchaseOrderNumber

3.4.21.3. Display Name

Purchase Order Number

3.4.21.4. Description

The unique customer-issued identifier for tracking the lifecycle of a purchase.

3.4.21.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Conditional
Allows nulls	True
Data type	String
Value format	<not specified>

3.4.21.6. Version Introduced

1.4

3.4.22. Reference Invoice ID

Reference Invoice ID is an invoice-issuer-assigned identifier for an invoice that affects charges as stated on a previous invoice. This occurs typically in the context of credits, refunds, or corrections where an adjustment is applied to a specific previously-issued billing document. This ID allows for direct lineage between adjustments and the original billing records.

3.4.22.1. Requirements

ReferenceInvoiceId MUST adhere to the following requirements:

- ReferenceInvoiceId MUST be of type String.
- ReferenceInvoiceId MUST conform to StringHandling requirements.
- ReferenceInvoiceId MUST NOT be null.
- ReferenceInvoiceId MUST match the InvoiceId of the original *invoice* when it adjusts another invoice.
- ReferenceInvoiceId MUST match the InvoiceId of the current invoice when it does not adjust another invoice.

3.4.22.2. Column ID

ReferenceInvoiceId

3.4.22.3. Display Name

Reference Invoice ID

3.4.22.4. Description

The invoice-issuer-assigned identifier for an invoice that affects charges as stated on a previous invoice.

3.4.22.5. Content Constraints

Constraint	Value
Dataset	Invoice Detail
Column type	Dimension
Feature level	Mandatory

Constraint	Value

Allows nulls	False
Data type	String
Value format	<not specified>

3.4.22.6. Version Introduced

1.4

4. Attributes

Attributes serve as reusable containers for requirements that enforce consistency across FOCUS specification entities (e.g., datasets, columns, objects). Functioning as a governance layer, Attributes define the constraints - such as naming conventions, data types, granularity, or recency - that an entity must satisfy. By grouping these requirements, Attributes ensure that data from any origin can be processed via standard instructions, which is essential for accurately supporting FinOps capabilities.

Attribute List

Attribute	Description
Correction Handling	Defines how corrections to previously delivered dataset artifacts are represented in subsequent deliveries.
Currency Format	Specifies rules and formatting requirements for currency columns.
Custom Column Handling	Defines column ID naming, formatting, and value requirements for *custom columns* appearing in a *FOCUS dataset*.
Data Generator-Calculated Split Cost Allocation Handling	Allows data generators to provide granular cost information based on specific documented methods.
Dataset Completeness	Defines requirements for a *FOCUS dataset* to include *custom columns* not represented in *FOCUS columns*.
Dataset Configuration	Defines the rules for customizing a dataset's schema.
Date/Time Format	Outlines rules and ISO 8601 formatting requirements for date and time information.
Delivery Handling	Defines how a data generator delivers a FOCUS dataset to a customer.
FOCUS Column Handling	Defines naming conventions for *FOCUS columns* appearing in a *FOCUS dataset*.
JSON Object Format	Defines rules for columns that convey data as complex, hierarchical serialized JSON strings.
Key-Value Format	Provides formatting requirements for columns conveying data as simple key-value pairs.
Null Handling	Standardizes how to represent columns that do not have a value using NULL.
Numeric Format	Establishes rules for numeric values to ensure clarity, accuracy, and ease of interpretation.
String Handling	Sets requirements for string-capturing columns to foster data integrity and interoperability.
Unit Format	Standardizes the expression of measurement units for data size, count, time, and other dimensions.

4.1. Correction Handling

4.1.1. Overview

The Correction Handling attribute defines how *corrections* to previously delivered *FOCUS dataset artifacts* are represented in subsequent deliveries.

Corrections may consist of one or more simultaneous changes, including updates to or omission of previously delivered records, or the addition of new records that supplement previously delivered data within the affected *delivery scope* (e.g., temporal grouping such as a *billing period* or non-temporal, logical grouping such as a *contract*).

Corrections may address a variety of operational or technical causes, such as refunds, late-arriving data, rounding errors, delivery errors, and other post-processing adjustments.

Accurate correction handling is essential to ensure the consistency, integrity, and usability of *FOCUS datasets* over time. Depending on the dataset and delivery configuration, it supports a range of key outcomes, including but not limited to:

- Consistency of delivered data - ensuring that delivered data remains consistent and reliable over time, where applicable, including alignment between related *FOCUS datasets* (e.g., Invoice Detail records and the underlying Cost

and Usage records).

- Data integrity and *invoice reconciliation* - ensuring that corrections do not compromise records associated with *issued invoices* or *closed billing periods*, and that alignment is maintained in accordance with defined *invoice reconciliation* requirements.
- Auditability and traceability - enabling the tracking of delivered data and applied corrections over time, so that changes and their effects can be understood, verified, and correctly reflected in downstream processes (e.g., cost allocation, chargeback, reporting).

See Appendix: Invoice and Billing Period Handling for details on corrections to *issued invoices* and *closed billing periods*.

4.1.1.1. Correction Styles

FOCUS recognizes three styles for handling corrections within subsequent *dataset artifacts*:

Correction Style	Delivery Mechanism	Correction Style Description
Replacement	Overwrite	Previously delivered records are not corrected individually; each delivery provides a complete snapshot and supersedes any previously delivered data within the affected *delivery scope*.
Delta	Append	Previously delivered records are preserved; corrections are appended as additive records applied during aggregation and may include supplemental records as needed.
Ledger	Append	Previously delivered records are preserved; corrections are appended as additive records representing explicit reversals and re-entries, applied during aggregation, and may include supplemental records as needed.

For more information on delivery mechanisms for *dataset artifacts*, see the Delivery Handling attribute.

4.1.1.1.1. Replacement Corrections

In the Replacement correction style, a *dataset artifact* uses the Overwrite delivery mechanism to provide a complete snapshot of data for a given *delivery scope*, based on the data available at the time of delivery.

Any given *dataset artifact* completely replaces all previous *dataset artifacts* for the same *delivery scope* to reflect updates, additions, or omissions relative to the previous snapshot. The practitioner only needs to reference the most recent *dataset artifact* for a given *delivery scope* in order to see a complete view; all previously delivered *dataset artifacts* for that *delivery scope* are considered obsolete and can be safely ignored.

Given that changes are not presented as separate entries, this style lacks inherent auditability. Dataset artifact size typically increases within a delivery scope as underlying data accumulates, but net data volume is the lowest of the three correction styles as each dataset artifact supersedes previously delivered ones for the same delivery scope.

4.1.1.1.2. Delta Corrections

In the Delta correction style, a *dataset artifact* uses the Append delivery mechanism to provide additive records for a given *delivery scope*, based on the data available at the time of delivery.

All previously delivered *dataset artifacts* are preserved, and corrections are expressed as additive records that are applied during aggregation. These records effectively increase or decrease values in selected additive metrics (e.g., cost- and quantity-related columns) of previously delivered records, or supplement previously delivered records, all within the same *delivery scope*. The practitioner must reference all *dataset artifacts* delivered for a given *delivery scope* in order to see a complete and accurate view.

Given that only net changes are presented and previously delivered records are not explicitly reversed, the Delta correction style provides limited inherent auditability compared to Ledger corrections. Net data volume increases over time as all delivered dataset artifacts are preserved, and is typically higher compared to Replacement corrections but lower compared to Ledger corrections.

4.1.1.1.3. Ledger Corrections

In the Ledger correction style, a *dataset artifact* uses the Append delivery mechanism in combination with a double-entry

bookkeeping method to provide detailed updates for a given *delivery scope*, based on the data available at the time of delivery. Depending on the nature of the correction, either or both of the following steps may be required: (1) reversal of the original record using a record in which additive metrics (e.g., cost- and quantity-related columns) carry values with the opposite sign, while all other columns match the original; and (2) a new record with corrected values.

All previously delivered *dataset artifacts* are preserved, and corrections are expressed as additive records that reflect explicit reversals and re-entries, applied during aggregation. These records effectively increase or decrease values in selected additive metrics (e.g., cost- and quantity-related columns) of previously delivered records, or supplement previously delivered records, all within the same *delivery scope*. The practitioner must reference all *dataset artifacts* delivered for a given *delivery scope* in order to see a complete and accurate view.

Given that the entire change history is presented, the Ledger correction style provides full inherent auditability. Net data volume increases over time as all delivered dataset artifacts are preserved, and is typically the highest of the three correction styles as each correction requires explicit reversal and re-entry records.

4.1.2. Requirements

Dataset conforming to CorrectionHandling attribute MUST adhere to the following requirements:

- *FOCUS dataset* MUST have its styles for representing corrections in *dataset artifacts* documented and accessible to practitioners (including whether Replacement, Delta, or Ledger style is used and under which conditions each style applies).
- *FOCUS dataset* MUST represent a complete snapshot of data for the affected *delivery scope* when using Replacement correction style.
- *FOCUS dataset* MUST include additive records representing corrections within the same *delivery scope* when using Delta correction style.
- *FOCUS dataset* MUST include explicit reversal and re-entry additive records representing corrections within the same *delivery scope* when using Ledger correction style.

4.1.3. Attribute ID

CorrectionHandling

4.1.4. Attribute Name

Correction Handling

4.1.5. Description

Defines how *corrections* to previously delivered FOCUS *dataset artifacts* are represented in subsequent deliveries.

4.1.6. Version Introduced

1.4

4.2. Currency Format

Columns that contain currency information in cost data following a consistent format reduce friction for FinOps practitioners who consume the data for analysis, reporting, and other use cases.

A currency may be one of the following currency types:

- National currency (e.g., USD, EUR).
- Virtual currency (e.g., tokens, credits).

4.2.1. Requirements

Column conforming to CurrencyFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST conform to ISO 4217:2015 standard.
- *FOCUS dataset column* MUST use the three-letter alphabetic code defined in ISO 4217:2015 (e.g., USD, EUR).

4.2.2. Attribute ID

CurrencyFormat

4.2.3. Attribute Name

Currency Format

4.2.4. Description

Formatting for currency columns appearing in a *FOCUS dataset*.

4.2.5. Version Introduced

0.5

4.3. Custom Column Handling

A *FOCUS dataset* consists of a set of columns that convey information about the records provided by a *data generator*. While FOCUS establishes the core structure and defines standardized *FOCUS columns* for consistent reporting, the diverse and evolving landscape of service providers and service offerings may require *data generators* to include *custom columns* in a *FOCUS dataset*.

The Custom Column Handling attribute defines column ID naming, formatting, and value requirements for *custom columns* appearing in a *FOCUS dataset*.

Each column describes an aspect of the record, including but not limited to:

- Who is responsible for or associated with the activity.
- What the record represents.
- When the activity occurred.
- Where the activity took place.
- Why the record exists or has specific values.
- How values are calculated or determined.

These additional columns enable deeper analysis and provide more detailed information that may not be fully captured by standard *FOCUS columns*. See the Dataset Completeness attribute for requirements on what *custom columns* to include.

4.3.1. Requirements

Column conforming to CustomColumnHandling attribute MUST adhere to the following requirements:

- *Custom column* MUST adhere to the following Column ID naming requirements:
  - *Custom column* MUST include the `x_` prefix in the Column ID to identify it as an external *custom column* and to distinguish it from *FOCUS columns* to avoid conflicts in future releases.
  - *Custom column* SHOULD use *Pascal case* in the portion of the Column ID following the required `x_` prefix.
  - *Custom column* SHOULD use only alphanumeric characters in the portion of the Column ID following the required `x_` prefix.
  - *Custom column* SHOULD NOT include special characters other than the underscore in the required `x_` prefix.
  - *Custom column* SHOULD NOT use abbreviations other than `Id` in the Column ID.
  - *Custom column* SHOULD NOT use acronyms other than `Sku` in the Column ID.
  - *Custom column* SHOULD NOT exceed 50 characters in the Column ID to accommodate column length restrictions of various data repositories.
  - *Custom column* SHOULD include the `Id` suffix in the Column ID when the *custom column* represents an identifier.

    - *Custom column* SHOULD include the `Name` suffix in the Column ID when the *custom column* represents a name.
- *Custom column* SHOULD conform to DataGeneratorCalculatedSplitCostAllocationHandling requirements when the data generator supports data generator-calculated split cost allocation.
- *Custom column* SHOULD conform to NullHandling requirements.
- *Custom column* containing date/time values SHOULD conform to DateTimeFormat requirements.
- *Custom column* containing JSON objects MUST have its object schema documented by the data generator and accessible to practitioners.
- *Custom column* containing numeric values MUST contain a single numeric value.
- *Custom column* containing numeric values SHOULD conform to NumericFormat requirements.
- *Custom column* containing string values SHOULD conform to StringHandling requirements.
- *Custom column* representing a national currency SHOULD conform to CurrencyFormat requirements.
- *Custom column* representing a measurement unit SHOULD conform to UnitFormat requirements.

4.3.2. Attribute ID

CustomColumnHandling

4.3.3. Attribute Name

Custom Column Handling

4.3.4. Description

Column ID naming, formatting, and value requirements for *custom columns* appearing in a *FOCUS dataset*.

4.3.5. Version Introduced

1.4

4.4. Data Generator-Calculated Split Cost Allocation Handling

The data generator-calculated split cost allocation for data generator-defined services is a capability that can be offered by data generators which allocates a charge to a more granular level or, in some cases, surfaces detail not visible in the origin charge (such as unused capacity). This is accomplished by taking a charge record present in a FOCUS dataset (*origin charge*) and splitting it into multiple charge records (*allocated charges*) to reflect the more granular detail, while ensuring the origin charge can be derived from the combination of *allocated charges*. This feature is used by practitioners to conduct chargebacks and better understand the usage of resources.

4.4.1. Requirements

Column conforming to DataGeneratorCalculatedSplitCostAllocationHandling attribute MUST adhere to the following requirements:

- *FOCUS dataset column* representing a dimension MUST match the corresponding value in the *origin charge* when present in an *allocated charge*.
- *FOCUS dataset column* representing a non-summable *metric* (e.g., unit prices) MUST match the corresponding value in the *origin charge* when present in an *allocated charge*.
- The sum of *FOCUS dataset column* across *allocated charges* MUST match the *FOCUS dataset column* in the corresponding *origin charge* when the *FOCUS dataset column* represents a summable metric (e.g., costs and quantities).

4.4.2. Attribute ID

DataGeneratorCalculatedSplitCostAllocationHandling

4.4.3. Attribute Name

Data Generator-Calculated Split Cost Allocation Handling

4.4.4. Description

An attribute that allows data generators to offer more detailed cost and usage information based on a method defined and documented by the data generator, including support for allocating costs in cases where the usage of a resource might not match the units the resource is measured in.

4.4.5. Version Introduced

1.3

4.5. Dataset Completeness

FinOps practitioners need data beyond what *FOCUS columns* define to facilitate a variety of FinOps activities. When this data is only available in *native datasets*, practitioners cannot rely on *FOCUS datasets* as a primary data source, making FOCUS an added overhead rather than a data generator-agnostic alternative that supports essential FinOps activities.

The Dataset Completeness attribute requires *data generators* to include *custom columns* in a *FOCUS dataset* for all *native dataset* columns except those explicitly listed as exclusions with justification in publicly-available documentation. This allows practitioners to adopt *FOCUS datasets* without losing analytical capabilities.

4.5.1. Requirements

Dataset conforming to DatasetCompleteness attribute MUST adhere to the following requirements:

- *FOCUS dataset* MUST adhere to the following *custom column* presence requirements:
  - *FOCUS dataset* MUST include *custom columns* (e.g., `x_ChargeSubType`) needed to support *invoice reconciliation* when the *invoice issuer* supports payable invoices, and when *FOCUS columns* are not sufficient.
  - *FOCUS dataset* MUST include *custom columns* corresponding to *native dataset* columns, except those explicitly listed as exclusions with justification in publicly-available documentation, provided those excluded columns are unrelated to *invoice reconciliation*.
  - *FOCUS dataset* MUST have all included *custom columns* documented in publicly-available documentation, including description, purpose, and relationship to *native dataset* columns.
  - *FOCUS dataset* SHOULD include *custom columns* that enable correlation between *FOCUS dataset* records and *native dataset* records (e.g., native *charge* identifiers), even when they meet the criteria for exclusion.
  - *FOCUS dataset* SHOULD exclude *custom columns* that duplicate information already captured in *FOCUS columns*, except during a transitional period as defined in publicly-available documentation, to enable migration without breaking changes.
- *FOCUS dataset* MUST retain the fidelity of corresponding *native dataset* values within *custom columns* without lossy transformations (e.g., rounding or truncation).
- *FOCUS dataset* MUST NOT alter the aggregated values of summable *metrics* (e.g., costs and quantities) due to the inclusion of *custom columns*.
- *FOCUS dataset* SHOULD sort all *FOCUS columns* alphabetically first, then all *custom columns* alphabetically second.

4.5.2. Implementation Context

A service provider's native cost dataset includes a column `internal_project_id` used for organizational hierarchy attribution. Since no *FOCUS column* captures this data, the corresponding *FOCUS dataset* includes it as a *custom column* (`x_InternalProjectId`), enabling practitioners to perform the same project-level cost allocation without falling back to the *native dataset*.

The native dataset also includes a native billing event reference (`billing_event_id`) that does not directly support analysis but allows practitioners to trace *FOCUS dataset* records back to native records. The *FOCUS dataset* includes this as `x_BillingEventId` to enable correlation between the two datasets.

Even when a single native record results in multiple FOCUS records (e.g., separating discounted and non-discounted portions of a charge in a Cost and Usage *dataset artifact*), *custom columns* like `x_InternalProjectId` and `x_BillingEventId` are preserved in

all resulting records and cost metrics like `BilledCost` are split accurately, maintaining data integrity.

Custom columns that duplicate newly introduced *FOCUS columns* may be preserved during a documented transitional period to enable migration without breaking changes.

This attribute ensures *custom columns* are fully represented in the *FOCUS dataset* schema. Data generators may require FOCUS consumers to explicitly select these columns when generating a *dataset artifact* (see Dataset Configuration).

4.5.3. Attribute ID

DatasetCompleteness

4.5.4. Attribute Name

Dataset Completeness

4.5.5. Description

Defines requirements for a *FOCUS dataset* to include *custom columns* for *native dataset* columns not represented in *FOCUS columns*.

4.5.6. Version Introduced

1.4

4.6. Dataset Configuration

Dataset Configuration allows FinOps practitioners to tailor the structure and content of a *FOCUS dataset*. Datasets provided by data generators are often massive, and their ingestion can lead to excessive storage costs and slow processing times. By removing large, static, or irrelevant columns, FinOps practitioners can optimize the dataset for better performance and lower storage costs.

Common scenarios where dataset configuration is valuable include:

- **Managing Scale**: Trim large exports to reduce time and cost of data preparation
- **Reducing Noise**: Tailor datasets for specific workflows (e.g., cost allocation, commitment analysis)
- **Lowering Barriers**: Strip away technical complexity for spreadsheet users
- **Enabling Comparison**: Remove custom (`x_`) columns for standardized cross-provider reporting

4.6.1. Requirements

Dataset conforming to DatasetConfiguration attribute MUST adhere to the following requirements:

- *FOCUS dataset* MUST be configurable to include only a user-defined selection of columns.
- *FOCUS dataset* MUST adhere to all column-level specifications defined in the FOCUS schema, regardless of the user's chosen configuration (e.g., column selection).
- *FOCUS dataset* MAY offer a default column set.
- *FOCUS dataset* default column set MUST include all applicable *FOCUS columns* when a default column set is offered.

4.6.2. Example

A practitioner configures their FOCUS Cost and Usage dataset to include only these columns:

- BillingAccountId
- ServiceName
- BilledCost
- EffectiveCost

- Tags

Even though columns like `CommitmentDiscountId` and `ResourceId` are excluded, the included cost columns (`BilledCost`, `EffectiveCost`) still reflect commitment discounts correctly. The dataset remains conformant to the FOCUS specification because each included column follows all requirements for that column, including requirements that reference columns not in the dataset.

4.6.3. Attribute ID

DatasetConfiguration

4.6.4. Attribute Name

Dataset Configuration

4.6.5. Description

Defines configuration options for controlling the structure and content of a FOCUS dataset.

4.6.6. Version Introduced

1.4

4.7. Date/Time Format

Columns that provide date and time information conforming to specified rules and formatting requirements ensure clarity, accuracy, and ease of interpretation for both humans and systems.

4.7.1. Requirements

Column conforming to DateTimeFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST be expressed in UTC (Coordinated Universal Time) to avoid ambiguity and ensure consistency across different time zones.
- *FOCUS dataset column* MUST conform to the ISO 8601 standard, which provides a globally recognized format for representing dates and times (see ISO 8601-1:2019 governing document for details).
- When *FOCUS dataset column* represents a specific moment in time, it MUST adhere to the following requirements:
  - *FOCUS dataset column* MUST use the extended ISO 8601 format with UTC offset (`YYYY-MM-DDTHH:mm:ssZ`).
  - *FOCUS dataset column* MUST include both the date and time components, separated with the letter `T`.
  - *FOCUS dataset column* MUST use two-digit hours (`HH`), minutes (`mm`), and seconds (`ss`).
  - *FOCUS dataset column* MUST end with the ISO 8601 UTC designator `Z`.

4.7.2. Attribute ID

DateTimeFormat

4.7.3. Attribute Name

Date/Time Format

4.7.4. Description

Rules and formatting requirements for date/time-related columns appearing in a *FOCUS dataset*.

4.7.5. Version Introduced

0.5

4.8. Delivery Handling

4.8.1. Overview

The Delivery Handling attribute defines how a *data generator* delivers a *FOCUS dataset* to a customer.

A *dataset instance* represents a specific implementation of a *FOCUS dataset*. A *dataset instance artifact* is the physical delivery of that instance, representing one or more records, independent of storage or transport boundaries (e.g., files, batches, or responses).

4.8.1.1. Delivery Mechanisms

FOCUS recognizes two *FOCUS dataset* delivery mechanisms:

- Overwrite: Each delivery provides a complete snapshot, superseding any previously delivered *dataset artifact* for the same *delivery scope* (e.g., temporal grouping such as a *billing period* or non-temporal, logical grouping such as a *contract*).
- Append: Each delivery adds new data, while previously delivered *dataset artifacts* are preserved.

Overwrite and Append mechanisms are not mutually exclusive, and hybrid implementations are common in practice, allowing data generators to meet specific technical and auditability requirements.

For example, for Cost and Usage *FOCUS datasets*, a data generator may use Overwrite mechanism for *dataset artifacts* corresponding to an *open billing period*, ensuring the snapshot reflects the most recent state, while using Append mechanism for *closed billing periods* to preserve historical data and support auditing of corrections to previously *closed billing periods* (i.e., *charges* with Charge Class set to "Correction").

For more information on corrections, see the Correction Handling attribute.

4.8.1.1.1. Overwrite Delivery

In the Overwrite delivery mechanism, each *dataset artifact* provides a complete snapshot of data for a predefined scope (e.g., a *billing period* or a logical grouping), based on the data available at the time of delivery. Subsequent *dataset artifacts* for the same scope typically reflect updates, additions, or omissions relative to the previous snapshot. This mechanism provides delivery simplicity, but it lacks inherent auditability. Dataset artifact size typically increases within a delivery scope as underlying data accumulates, but total data volume remains the lowest compared to Append delivery as each dataset artifact supersedes previously delivered ones for the same delivery scope.

Subsequent *dataset artifacts* using the Overwrite mechanism may include the following:

- Unchanged records are carried over.
- Updated records overwrite previous values.
- Additional records supplement previously delivered data.
- Omitted records are removed if no longer applicable.

4.8.1.1.2. Append Delivery

In the Append delivery mechanism, a subsequent *dataset artifact* appends new records without modifying or removing previously delivered ones. This mechanism inherently supports auditability, as all original and correction records are retained. Total data volume increases over time as all delivered dataset artifacts are preserved, and is typically higher compared to Overwrite delivery.

Subsequent *dataset artifacts* using the Append mechanism may include the following:

- Unchanged records are not included.
- Updated records are recorded as new entries, representing the net effect on aggregated quantities or costs.
- Additional records supplement previously delivered data.
- Omitted records are recorded as new entries, representing the reversal.

4.8.2. Requirements

Dataset conforming to DeliveryHandling attribute MUST adhere to the following requirements:

- *FOCUS dataset* MUST NOT require practitioners to deduplicate records within or across delivered dataset artifacts.
- When using Overwrite delivery mechanism, *FOCUS dataset* MUST adhere to the following additional requirements:
  - *FOCUS dataset* MUST represent a complete snapshot for a given *delivery scope*.
  - *FOCUS dataset* MUST supersede all previously delivered *dataset artifacts* for the same *delivery scope*.
- *FOCUS dataset* MUST preserve all previously delivered *dataset artifacts* when using Append delivery mechanism.
- *FOCUS dataset* SHOULD have delivered *dataset artifacts* accompanied by corresponding FOCUS Metadata.
- *FOCUS dataset* delivery mechanism documentation MUST adhere to the following requirements:
  - *FOCUS dataset* delivery mechanism documentation MUST include the delivery mechanism used (Overwrite or Append).
  - *FOCUS dataset* delivery mechanism documentation MUST include the conditions under which each delivery mechanism applies when more than one delivery mechanism is used.
  - *FOCUS dataset* delivery mechanism documentation MUST include the mechanism for correlating *dataset artifacts* with the FOCUS Metadata Schema object when the Metadata is delivered.
  - *FOCUS dataset* delivery mechanism documentation MUST be accessible to practitioners.

4.8.3. Attribute ID

DeliveryHandling

4.8.4. Attribute Name

Delivery Handling

4.8.5. Description

Defines how a *data generator* delivers a *FOCUS dataset* to a customer.

4.8.6. Version Introduced

1.4

4.9. FOCUS Column Handling

Columns within FOCUS include an ID and a display name. Column IDs are used in *dataset artifacts* and display names can be used in report output and other descriptive content, like documentation. Column IDs provided in a *FOCUS dataset* follow consistent naming conventions for FinOps practitioners who consume the data for analysis, reporting, and other use cases.

4.9.1. Requirements

Column conforming to FocusColumnHandling attribute MUST adhere to the following requirements:

- *FOCUS column* MUST use a Display Name consistent with the Column ID, with spaces inserted between words (e.g., Column ID `BillingAccountName` and Display Name `Billing Account Name`, Column ID `BillingAccountId` and Display Name `Billing Account ID`).
- *FOCUS column* MUST use *Pascal case* in the Column ID.
- *FOCUS column* MUST use only alphanumeric characters in the Column ID.

- *FOCUS column* MUST NOT include special characters in the Column ID.
- *FOCUS column* MUST NOT use abbreviations other than `Id` in the Column ID.
- *FOCUS column* SHOULD NOT use acronyms other than `Sku` in the Column ID.
- *FOCUS column* SHOULD NOT exceed 50 characters in the Column ID to accommodate column length restrictions of various data repositories.
- *FOCUS column* representing an identifier MUST include the `Id` suffix in the Column ID.
- *FOCUS column* representing a name MUST include the `Name` suffix in the Column ID.
- *FOCUS column* representing a product offering that incurred a charge MUST include `Sku` in the Column ID.
- *FOCUS column* that includes the `Category` suffix in the Column ID and is not null MUST contain one of the FOCUS-defined allowed values.

4.9.2. Attribute ID

FocusColumnHandling

4.9.3. Attribute Name

FOCUS Column Handling

4.9.4. Description

Naming conventions for *FOCUS columns* appearing in a *FOCUS dataset*.

4.9.5. Version Introduced

0.5

4.10. JSON Object Format

JSON Objects extend the Key-Value Format to add support for complex data types like arrays and nested key-value pairs. This format is used when the Key-Value Format is insufficient to represent the complexity, such as when multiple sets of key-value pairs apply to the same charge record. JSON Objects are also referred to as maps, trees, or hashtables.

4.10.1. Requirements

Column conforming to JsonObjectFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST contain a serialized JSON string, consistent with the ECMA 404 definition of an object.
- *FOCUS dataset column* MUST conform to all requirements of the corresponding column definition, which may specify or restrict the shape or contents of the object.
- Object in *FOCUS dataset column* SHOULD NOT exceed 3 levels of nesting.
- Key in Object in *FOCUS dataset column* MUST be unique.
- Key value in Object in *FOCUS dataset column* MUST be of type number, string, boolean (`true` or `false`), array, object, or `null`.
- Object in array in *FOCUS dataset column* MUST adhere to the following requirements:
  - Object in array in *FOCUS dataset column* MUST be of a consistent type.
  - Object in array in *FOCUS dataset column* MUST NOT be repeated.
  - Object in array in *FOCUS dataset column* MUST NOT be null.

4.10.2. Attribute ID

JsonObjectFormat

4.10.3. Attribute Name

JSON Object Format

4.10.4. Description

Rules and formatting requirements for columns appearing in a *FOCUS dataset* that convey data as complex, hierarchical objects.

4.10.5. Version Introduced

1.3

4.11. Key-Value Format

Columns that provide Key-Value information are often used in place of separate columns for enumerating data which would be inherently sparse and/or without predetermined keys. This consolidates related information and provides more consistency in the schema. Key-value pairs are also referred to as name-value pairs, attribute-value pairs, or field-value pairs.

4.11.1. Requirements

Column conforming to KeyValueFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST be a serialized JSON string, consistent with the ECMA 404 definition of an object.
- Keys in *FOCUS dataset column* MUST be unique within the object.
- Key values in *FOCUS dataset column* MUST be of type number, string, boolean (`true` or `false`), or `null`.
- Key values in *FOCUS dataset column* MUST NOT be objects or arrays.

4.11.2. Attribute ID

KeyValueFormat

4.11.3. Attribute Name

Key-Value Format

4.11.4. Description

Rules and formatting requirements for columns appearing in a *FOCUS dataset* that convey data as key-value pairs.

4.11.5. Version Introduced

1.0-preview

4.12. Null Handling

FOCUS dataset records that don't have a value that can be presented for a column must be handled in a consistent way to reduce friction for FinOps practitioners who consume the data for analysis, reporting, and other use cases.

4.12.1. Requirements

Column conforming to NullHandling attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST use `null` for absent values when the *FOCUS dataset column* is defined as nullable.
- *FOCUS dataset column* MUST NOT contain empty strings or placeholder strings (e.g., `Not Applicable`) for absent values when the *FOCUS dataset column* contains string values.
- *FOCUS dataset column* MUST NOT contain placeholder numeric values (e.g., `0`) for absent values when the *FOCUS dataset column* contains numeric values.

4.12.2. Attribute ID

NullHandling

4.12.3. Attribute Name

Null Handling

4.12.4. Description

Indicates how to handle columns that don't have a value.

4.12.5. Version Introduced

0.5

4.13. Numeric Format

Columns that provide numeric values conforming to specified rules and formatting requirements ensure clarity, accuracy, and ease of interpretation for humans and systems. The FOCUS specification does not require a specific level of precision for numeric values. The level of precision required for a given column is determined by the provider and should be part of a data definition published by the provider.

4.13.1. Requirements

Column conforming to NumericFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST contain a single numeric value.
- *FOCUS dataset column* MUST have values of type integer, decimal, or scientific notation.
- *FOCUS dataset column* MUST contain values that, when not null, conform to one of the allowed Data Types defined in the table below.
- *FOCUS dataset column* MUST contain values that, when not null, conform to one of the allowed precision levels (and scale, where applicable) defined in the table below.
- *FOCUS dataset column* MUST NOT use mathematical symbols, functions, or operators, except for a negative sign (-) to indicate a negative value or a negative exponent in scientific notation.
- *FOCUS dataset column* MUST NOT include additional characters or qualifiers (e.g., currency symbols, units of measure).
- *FOCUS dataset column* MUST NOT contain commas or punctuation marks, except for a single decimal point when required for a decimal value.
- *FOCUS dataset column* MUST use a negative sign (-) to indicate a negative value.
- *FOCUS dataset column* MUST NOT include a positive sign (+) for a positive value.
- When *FOCUS dataset column* contains numeric values expressed in scientific notation, it MUST adhere to the following requirements:
  - *FOCUS dataset column* MUST use E notation "mEn", where m is a real number and n is an integer exponent.
  - *FOCUS dataset column* MUST use a negative sign (-) to indicate a negative exponent.
  - *FOCUS dataset column* MUST NOT include a positive sign (+) for a positive exponent.

4.13.1.1. Allowed Data Types

Data Type	Type Description
Integer	Specifies a numeric value represented by a whole number or by zero. Integer number formats correspond to standard data types defined by ISO/IEC 9899:2018
Decimal	Specifies a numeric value represented by a decimal number. Decimal formats correspond to ISO/IEC/IEEE 60559:2011 and IEEE 754-2008 definitions.

4.13.1.2. Allowed Precisions

Data Type	Precision	Definition	Range / Significant Digits
Integer	Short	16-bit signed short int ISO/IEC 9899:2018	-32,767 to +32,767
Integer	Long	32-bit signed long int ISO/IEC 9899:2018	-2,147,483,647 to +2,147,483,647
Integer	Extended	64-bit signed two's complement integer *or higher*	-(2^63 - 1) to (2^63 - 1)
Decimal	Single	32-bit binary format IEEE 754-2008 floating-point (decimal32)	9
Decimal	Double	64-bit binary format IEEE 754-2008 floating-point (decimal64)	16
Decimal	Extended	128-bit binary format IEEE 754-2008 floating-point (decimal128) or higher	36+

4.13.2. Examples

This format requires that single numeric values be represented using an integer or decimal format without additional characters or qualifiers. The following lists provide examples of values that meet the requirements and those that do not.

- Values Meeting Numeric Requirements:
    - -100.2
    - -3
    - 4
    - 35.2E-7
    - 1.234
- Values NOT Meeting Numeric Requirements
    - 1 1/2 - contains fractional notation
    - 35.2E+7 - contains a positive exponent with a sign
    - 35.24 x 10^7 - contains an invalid format for scientific notation
    - [3,5,8] - contains an array
    - [4:5] - contains a range
    - 5i + 4 - contains a complex number
    - sqrt(2) - contains a mathematical symbol or operation
    - 2.3^3 - contains an exponent
    - 32 GiB - contains a unit of measure
    - $32 - contains a currency symbol
    - 3,432,342 - contains a comma
    - +333 - contains a positive sign

4.13.3. Attribute ID

NumericFormat

4.13.4. Attribute Name

Numeric Format

4.13.5. Description

Rules and formatting requirements for numeric columns appearing in a *FOCUS dataset*.

4.13.6. Version Introduced

1.0-preview

4.14. String Handling

Columns that capture string values conforming to specified requirements foster data integrity, interoperability, and consistency; improve data analysis and reporting; and support reliable data-driven decision-making.

4.14.1. Requirements

Column conforming to StringHandling attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST preserve the original casing of string values.
- *FOCUS dataset column* MUST preserve the original spacing of string values.
- *FOCUS dataset column* MUST preserve other relevant consistency factors as specified by the data generator or end-user.
- *FOCUS dataset column* MUST remain consistent across all *billing periods* when the *FOCUS dataset column* contains immutable string values (e.g., resource identifier, region identifier).
- When *FOCUS dataset column* contains mutable string values (e.g., resource name, region name), it MUST adhere to the following requirements:
  - *FOCUS dataset column* MUST reflect the altered value in all records pertaining to a period after the change.
  - *FOCUS dataset column* MUST reflect the string value as it existed prior to the change in all records pertaining to a period prior to the change when the record does not represent a correction to a previously closed billing period.
  - *FOCUS dataset column* MAY reflect the altered value in records pertaining to a period prior to the change when the record represents a correction to a previously closed billing period.
- When *FOCUS dataset column* contains not-nullable string values, it MUST adhere to the following requirements:
  - *FOCUS dataset column* SHOULD NOT contain empty strings.
  - *FOCUS dataset column* SHOULD NOT contain strings consisting solely of whitespace characters.

4.14.2. Attribute ID

StringHandling

4.14.3. Attribute Name

String Handling

4.14.4. Description

Requirements for string-capturing columns appearing in a *FOCUS dataset*.

4.14.5. Version Introduced

1.0

4.15. Unit Format

Billing data frequently captures data measured in units related to data size, count, time, and other *dimensions*. The Unit Format attribute provides a standard for expressing units of measure in columns appearing in a *FOCUS dataset*.

4.15.1. Requirements

Column conforming to UnitFormat attribute MUST adhere to the following requirements:

- *FOCUS dataset column* MUST adhere to the following base unit requirements:
  - *FOCUS dataset column* MUST include at least one base unit.
  - *FOCUS dataset column* MUST use one of the allowed data size unit abbreviations listed below for data size base units.
  - *FOCUS dataset column* MUST use the allowed data size unit abbreviations in the same form for both singular and plural units.
  - *FOCUS dataset column* MUST use the allowed abbreviation for exabit, exabyte, exbibit, or exbibyte when representing values exceeding 10^18.
  - *FOCUS dataset column* MUST use the allowed abbreviation for bit or byte when representing values smaller than one byte.
  - *FOCUS dataset column* MUST use one of the allowed time-based unit names listed below for time-based base units.
  - *FOCUS dataset column* SHOULD use one of the recommended count-based unit names listed below for count-based base units.
  - *FOCUS dataset column* SHOULD use capitalized nouns for base units that do not correspond to any of the allowed base unit names listed below.
  - *FOCUS dataset column* MAY include a count-based base unit that is not listed as one of the allowed values.
- *FOCUS dataset column* MAY include a unit quantity expressed as a positive integer.
- *FOCUS dataset column* expressing a compound unit MUST use a hyphen (`-`) to separate base units (e.g., `GB-Hours`).
- *FOCUS dataset column* expressing a compound unit SHOULD use the `<singular-base-unit>-<plural-base-unit>` format (e.g., `GB-Hours`, `MB-Days`, `Request-Tokens`).
- *FOCUS dataset column* expressing a ratio unit MUST use a slash (`/`) to separate the numerator and denominator (e.g., `GB/Hour` to signify gigabytes per hour).
- *FOCUS dataset column* expressing a ratio unit MAY include a denominator quantity expressed as a positive integer.
- *FOCUS dataset column* expressing a ratio unit and including a denominator quantity SHOULD use the `<plural-units>/<denominator-quantity> <plural-time-units>` format (e.g., `Units/3 Months`).
- *FOCUS dataset column* expressing a ratio unit with a compound unit numerator SHOULD use the `<compound-unit>/<singular-time-unit>` format (e.g., `Core-Hours/Day`).
- *FOCUS dataset column* expressing a ratio unit with a time denominator SHOULD use the `<plural-units>/<singular-time-unit>` format (e.g., `GB/Hour`, `PB/Day`).
- *FOCUS dataset column* expressing a simple unit SHOULD use the `<plural-units>` format (e.g., `GB`, `Seconds`).
- *FOCUS dataset column* including a unit quantity SHOULD use the `<unit-quantity> <plural-units>` format (e.g., `1000 Tokens`, `1000 Characters`).

4.15.2. Definitions

The normative requirements above refer to the following definitions.

4.15.2.1. Measurement Unit

A standardized expression that describes how quantities in a *FOCUS dataset* are denominated (e.g., `GB`, `Seconds`, `GB-Hours`, `10 GB/Hour`, `Units/3 Months`).

4.15.2.2. Base Unit

An atomic unit that serves as a building block for all measurement units; can be a data size unit, time-based unit, or count-based unit (e.g., `GB`, `Hour`, `Token`).

4.15.2.3. Simple Unit

A measurement unit that contains exactly one base unit, optionally preceded by a unit quantity (e.g., `GB`, `Seconds`, `1000 Tokens`).

4.15.2.4. Compound Unit

A measurement unit that combines two base units using a hyphen (`-`), optionally preceded by a unit quantity (e.g., `GB-Hours`, `MB-Days`, `Request-Tokens`).

4.15.2.5. Ratio Unit

A measurement unit that expresses one base unit or compound unit per another using a slash (`/`), optionally including a denominator quantity (e.g., `GB/Hour`, `Units/3 Months`, `Core-Hours/Day`).

4.15.2.6. Unit Quantity

A positive integer included in a measurement unit, indicating the granularity (e.g., `1000` in `1000 Tokens`).

4.15.2.7. Denominator Quantity

A positive integer included in the denominator of a ratio unit, indicating the granularity of the denominator (e.g., `3` in `Units/3 Months`).

4.15.3. Base Unit Names

4.15.3.1. Allowed Data Size Unit Abbreviations

Data size units are nouns representing data size measured in bits or bytes, expressed using standard abbreviations. Each abbreviation can be used for both the singular and plural form of the unit.

For example:

- `GB` represents both the singular and plural form of a gigabyte.
- `TB` is a valid base unit name, while `TBs` and `terabyte` are considered invalid.

Values larger than 10^18 must use the abbreviation for exabit, exabyte, exbibit, or exbibyte. Values smaller than a byte must use the abbreviation for bit or byte.

The table below lists the valid abbreviations for data size units from a single bit or byte to 10^18 bits or bytes.

Data size in bits	Data size in bytes
b (bit = 10^0)	B (byte = 10^0)
Kb (kilobit = 10^3)	KB (kilobyte = 10^3)
Mb (megabit = 10^6)	MB (megabyte = 10^6)
Gb (gigabit = 10^9)	GB (gigabyte = 10^9)
Tb (terabit = 10^12)	TB (terabyte = 10^12)
Pb (petabit = 10^15)	PB (petabyte = 10^15)
Eb (exabit = 10^18)	EB (exabyte = 10^18)
Kib (kibibit = 2^10)	KiB (kibibyte = 2^10)
Mib (mebibit = 2^20)	MiB (mebibyte = 2^20)
Gib (gibibit = 2^30)	GiB (gibibyte = 2^30)
Tib (tebibit = 2^40)	TiB (tebibyte = 2^40)
Pib (pebibit = 2^50)	PiB (pebibyte = 2^50)
Eib (exbibit = 2^60)	EiB (exbibyte = 2^60)

4.15.3.2. Allowed Time-Based Unit Names

Time-based units are nouns representing a discrete time period. They can be used alone to indicate duration, combined with another unit to form a compound unit (e.g., `GB-Hours`), or a per-time ratio unit (e.g., `GB/Hour`).

The table below lists allowed time-based base units.

Time-based Unit (Singular)	Time-based Unit (Plural)
Year	Years
Month	Months
Day	Days
Hour	Hours
Minute	Minutes
Second	Seconds
Millisecond	Milliseconds
Microsecond	Microseconds

4.15.3.3. Recommended Count-Based Unit Names

A count-based unit is a noun representing a discrete number of items, events, or actions. For example, a count-based unit can represent the number of requests, instances, tokens, or connections.

The table below lists recommended names for count-based base units.

Count-based Unit (Singular)	Count-based Unit (Plural)
Count	Counts
Unit	Units
Request	Requests
Token	Tokens
Connection	Connections
Certificate	Certificates
Domain	Domains
Core	Cores

Note: When a count-based base unit is not covered by the recommended values, a new value can be used as long as it is capitalized.

4.15.4. Attribute ID

UnitFormat

4.15.5. Attribute Name

Unit Format

4.15.6. Description

Indicates standards for expressing measurement units in columns appearing in a *FOCUS dataset*.

4.15.7. Version Introduced

1.0-preview

5. Metadata

The FOCUS specification defines a metadata structure to be supplied by data providers to facilitate practitioners' use of FOCUS data. This metadata includes general information about the *dataset artifact*.

The metadata includes the following sections:

Metadata Section	Description
Data Generator	Describes the entity delivering the dataset artifact.
Dataset Instance	Describes the nature of the dataset artifact.
Recency	Describes the recency and completeness of data within the artifact.
Schema	Describes the schema of data within the artifact.

Requirements

Metadata adheres to the following requirements:

- Data generators SHOULD provide FOCUS metadata in a format that is accessible programmatically (e.g., file, website, API, or table).
- Data generators SHOULD provide documentation on their implementation of the FOCUS metadata.

Metadata ID

Metadata

Metadata Name

Metadata

Version Introduced

1.0

5.1. Data Generator

The FOCUS metadata about the generator of the FOCUS data.

5.1.1. Requirements

DataGenerator adheres to the following requirements:

- DataGenerator MUST be present in the Metadata.
- DataGenerator MUST be of type Object.
- DataGenerator MUST NOT be null.

5.1.2. Metadata ID

DataGenerator

5.1.3. Metadata Name

Data Generator

5.1.4. Examples

For an example of the FOCUS Data Generator metadata please refer to: Data Generator Example.

5.1.5. Version Introduced

1.0

5.1.6. Data Generator

Human-readable name of the entity that generated the dataset instance. The Data Generator ensures the technical accuracy and delivery of the data.

DataGenerator property adheres to the following requirements:

- DataGenerator MUST be present in the Data Generator object.
- DataGenerator MUST be of type String.
- DataGenerator MUST conform to StringHandling requirements.
- DataGenerator MUST NOT be null.
- DataGenerator SHOULD reflect the entity that generated the *FOCUS dataset*.

5.1.6.1. Metadata ID

DataGenerator

5.1.6.2. Metadata Name

Data Generator

5.1.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.1.6.4. Version Introduced

1.0

5.2. Dataset Instance

The Dataset Instance metadata object provides information about the *dataset instance* and its content. Dataset Instance is a data generator-delivered instance of a FOCUS Dataset. For example, a Data Generator may provide multiple datasets utilizing the FOCUS specification, including multiple instances of the FOCUS Cost and Usage dataset, each representing a different time granularity.

5.2.1. Requirements

DatasetInstance adheres to the following requirements:

- DatasetInstance MUST be present in the Metadata.
- DatasetInstance MUST be structured as a collection of objects.
- DatasetInstance MUST NOT be null.

- DatasetInstance collection MUST contain at least one object for every *FOCUS dataset* supported by the data generator.
- DatasetInstance collection object MUST NOT be null.
- DatasetInstance collection object MUST be associated with one and only one *FOCUS dataset*.

5.2.2. Metadata ID

DatasetInstance

5.2.3. Metadata Name

Dataset Instance

5.2.4. Examples

For an example of the FOCUS dataset instance metadata, please refer to: Dataset Instance Metadata Example.

5.2.5. Version Introduced

1.3

5.2.6. Dataset Instance ID

The Dataset Instance ID is a data generator-specified unique identifier that represents a specific FOCUS dataset instance provided by the data generator.

DatasetInstanceId adheres to the following requirements:

- DatasetInstanceId MUST be present in an object within the DatasetInstance collection.
- DatasetInstanceId MUST be of type String.
- DatasetInstanceId MUST NOT contain null values.
- DatasetInstanceId MUST be a unique identifier within a data generator.
- DatasetInstanceId SHOULD be a Globally Unique Identifier (GUID).

5.2.6.1. Metadata ID

DatasetInstanceId

5.2.6.2. Metadata Name

Dataset Instance ID

5.2.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	GUID (recommended)

5.2.6.4. Version Introduced

1.3

5.2.7. Dataset Instance Name

The human-readable name of the dataset instance as provided by the data generator.

DatasetInstanceName adheres to the following requirements:

- DatasetInstanceName MUST be present in an object within the DatasetInstance collection.
- DatasetInstanceName MUST be of type String.
- DatasetInstanceName MUST NOT be null.

5.2.7.1. Metadata ID

DatasetInstanceName

5.2.7.2. Metadata Name

Dataset Instance Name

5.2.7.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.2.7.4. Version Introduced

1.3

5.2.8. FOCUS Dataset ID

The identifier of the FOCUS dataset for which the schema and its data conform to. This indicates which FOCUS dataset the data artifact aligns with, such as "FOCUS Cost and Usage" or "FOCUS Contract."

The FocusDatasetId property adheres to the following requirements:

- FocusDatasetId MUST be present in an object within the DatasetInstance collection.
- FocusDatasetId MUST be of type String.
- FocusDatasetId MUST NOT be null.
- FocusDatasetId MUST match the Dataset ID of one of the *FOCUS datasets* defined in the FOCUS specification.

5.2.8.1. Metadata ID

FocusDatasetId

5.2.8.2. Metadata Name

FOCUS Dataset ID

5.2.8.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.2.8.4. Version Introduced

1.3

5.3. Recency

The recency metadata object and its contents provide information about how up-to-date FOCUS datasets are and when they have been updated.

5.3.1. Requirements

Recency adheres to the following requirements:

- Recency MAY be present in the Metadata.
- Recency MUST be structured as a collection of objects.
- Recency MUST NOT be null.
- Recency collection MUST NOT contain null objects.
- Recency collection MAY contain one and only one object for every DatasetInstance object.
- Recency collection object MUST be associated with one and only one DatasetInstance object.
- Recency collection object MUST be retrievable without inspection of the contents of *dataset instance artifacts*.
- Recency collection object SHOULD be updated when the data generator updates the corresponding *dataset instance artifact*.

5.3.2. Metadata ID

Recency

5.3.3. Metadata Name

Recency

5.3.4. Examples

Example scenarios include but are not limited to:

- Updating Recency metadata for a time series dataset
- Updating Recency metadata for a non time series dataset

For an example of the FOCUS recency metadata, please refer to: Recency Metadata Example.

5.3.5. Version Introduced

1.3

5.3.6. Dataset Instance ID

The Dataset Instance ID provides the reference item to associate which Dataset Instance the recency metadata is for.

DatasetInstanceId adheres to the following requirements:

- DatasetInstanceId MUST be present in an object within the Recency collection.
- DatasetInstanceId MUST be of type String.
- DatasetInstanceId MUST NOT be null.
- DatasetInstanceId SHOULD be a Globally Unique Identifier (GUID).

5.3.6.1. Metadata ID

DatasetInstanceId

5.3.6.2. Metadata Name

Dataset Instance ID

5.3.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	GUID (recommended)

5.3.6.4. Version Introduced

1.3

5.3.7. Dataset Instance Complete

Dataset Instance Complete provides a boolean value to indicate that the dataset instance is considered complete by the Data Generator. The definition of complete is determined by the Data Generator and should be provided in documentation provided by the Data Generator. For Datasets that are time series, the Complete value indicates that the time sector is complete and therefore is located in as key value in a time sector. For datasets that are not time series, a value of "true" indicates that the dataset is complete and therefore is a property of the recency object.

DatasetInstanceComplete adheres to the following requirements:

- DatasetInstanceComplete MUST be present in an object within the Recency collection when the dataset instance is not a time-series dataset.
- DatasetInstanceComplete MUST be of type Boolean.
- DatasetInstanceComplete MUST NOT be null.

5.3.7.1. Metadata ID

DatasetInstanceComplete

5.3.7.2. Metadata Name

Dataset Instance Complete

5.3.7.3. Content Constraints

Constraint	Value
Feature level	Conditional
Allows nulls	False
Data type	Boolean
Value format	<not specified>

5.3.7.4. Version Introduced

1.3

5.3.8. Dataset Instance Last Updated

Datetime when the data present in the Dataset Instance was updated.

DatasetInstanceLastUpdated adheres to the following requirements:

- DatasetInstanceLastUpdated MUST be present in an object within the Recency collection.
- DatasetInstanceLastUpdated MUST be of type Date/Time.
- DatasetInstanceLastUpdated MUST conform to DateTimeFormat requirements.
- DatasetInstanceLastUpdated MUST NOT be null.

5.3.8.1. Metadata ID

DatasetInstanceLastUpdated

5.3.8.2. Metadata Name

Dataset Instance Last Updated

5.3.8.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.3.8.4. Version Introduced

1.3

5.3.9. Recency Last Updated

Datetime the recency metadata object was updated.

RecencyLastUpdated adheres to the following requirements:

- RecencyLastUpdated MUST be present in the metadata.
- RecencyLastUpdated MUST be of type Date/Time.
- RecencyLastUpdated MUST NOT be null.
- RecencyLastUpdated MUST conform to DateTimeFormat.

5.3.9.1. Metadata ID

RecencyLastUpdated

5.3.9.2. Metadata Name

Recency Last Updated

5.3.9.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.3.9.4. Version Introduced

1.3

5.3.10. Time Sectors

The FOCUS recency metadata's Time Sectors provide a list of time periods and metadata about them. Time Sectors are used when the associated *FOCUS dataset* is defined as a time series dataset (i.e., its dataset artifacts represent data distributed over time). Each time sector represents a single time period and the completeness of that time period as it pertains to the dataset artifact. Time sectors do not represent start and end dates of the dataset artifact but rather periods of time relative to the datasets Charge Period Start and Charge Period End. Length of time sectors can be determined by the Data Generator, though it is suggested to align time sector periods to the reports time granularity (Hourly cost reports = hourly time sectors).

5.3.10.1. Requirements

TimeSectors adheres to the following requirements:

- TimeSectors MUST be present in an object within the Recency collection when the associated *FOCUS dataset* is defined as a time series dataset.
- TimeSectors MUST be structured as a collection of objects.
- TimeSectors MUST NOT be null.
- TimeSectors collection MUST contain at least one object.
- TimeSectors collection MUST NOT contain null objects.
- TimeSectors collection object MUST be updated, when already present, or added to the collection whenever the data generator updates or provides new dataset artifacts.

5.3.10.2. Metadata ID

TimeSectors

5.3.10.3. Metadata Name

Time Sectors

5.3.10.4. Version Introduced

1.3

5.3.10.5. Time Sector Complete

Time Sector Complete provides a boolean value to indicate that the time sector is considered complete by the DataGenerator. The definition of complete is determined by the DataGenerator and should be provided in documentation provided by the DataGenerator.

TimeSectorComplete adheres to the following requirements:

- TimeSectorComplete MUST be present in the TimeSectors subsection of the Recency metadata section.
- TimeSectorComplete MUST be of type Boolean.
- TimeSectorComplete MUST not be null.

5.3.10.5.1. Metadata ID

TimeSectorComplete

5.3.10.5.2. Metadata Name

Time Sector Complete

5.3.10.5.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Boolean
Value format	<not specified>

5.3.10.5.4. Version Introduced

1.3

5.3.10.6. Time Sector Last Updated

Datetime the data in the time sector was last updated.

TimeSectorLastUpdated adheres to the following requirements:

- TimeSectorLastUpdated MUST be present in TimeSectors subsection of the Recency metadata section.
- TimeSectorLastUpdated MUST be of type Date/Time.
- TimeSectorLastUpdated MUST NOT be null.
- TimeSectorLastUpdated MUST conform to DateTimeFormat.

5.3.10.6.1. Metadata ID

TimeSectorLastUpdated

5.3.10.6.2. Metadata Name

Time Sector Last Updated

5.3.10.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.3.10.6.4. Version Introduced

1.3

5.3.10.7. Time Sector Start

The Time Sector Start is the datetime of the start of the time sector.

TimeSectorStart adheres to the following requirements:

- TimeSectorStart MUST be present in the TimeSectors subsection of the Recency metadata section.
- TimeSectorStart MUST be of type Date/Time.
- TimeSectorStart MUST NOT be null.
- TimeSectorStart MUST conform to DateTimeFormat.

5.3.10.7.1. Metadata ID

TimeSectorStart

5.3.10.7.2. Metadata Name

Time Sector Start

5.3.10.7.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.3.10.7.4. Version Introduced

1.3

5.3.10.8. Time Sector End

The Time Sector End is the datetime of the end of the time sector. The Time Sector End MUST be later than the Time Sector Start.

TimeSectorEnd adheres to the following requirements:

- TimeSectorEnd MUST be present in the TimeSectors subsection of the Recency metadata section.
- TimeSectorEnd MUST be of type Date/Time.
- TimeSectorEnd MUST NOT be null.
- TimeSectorEnd MUST conform to DateTimeFormat.
- TimeSectorEnd must be exclusive of the end time of the subsequent time sector.
- TimeSectorEnd MUST be later than TimeSectorStart.

5.3.10.8.1. Metadata ID

TimeSectorEnd

5.3.10.8.2. Metadata Name

Time Sector End

5.3.10.8.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.3.10.8.4. Version Introduced

1.3

5.4. Schema

The schema metadata object and its content provide information about the structure of the data provided.

5.4.1. Requirements

Schema adheres to the following requirements:

- Schema MUST be present in the Metadata.
- Schema MUST be structured as a collection of objects.
- Schema MUST NOT be null.
- Schema collection MUST contain at least one object for every DatasetInstance object.
- Schema collection MUST NOT contain null objects.
- Schema collection object MUST be associated with one and only one DatasetInstance object.
- Schema collection object MUST be added to the collection whenever the structure of the *dataset instance artifacts* changes (including, but not limited to, additions or removals of columns, modifications to any ColumnDefinition, or updates to the FOCUSVersion or DataGeneratorVersion).
- Schema collection object MUST be referenced by *dataset instance artifacts* that conform to the structure defined by that Schema collection object.
- Schema collection object MUST define the exact structure of the *dataset instance artifacts* that reference it.
- Schema collection object MUST be retrievable independently from the *dataset instance artifacts* that conform to the structure defined by that Schema collection object.
- Schema collection object SHOULD be provided separately from the *dataset instance artifacts* that conform to the

structure defined by that Schema collection object.
- Schema collection object MAY be provided through the structure and/or schema of the delivery mechanism (e.g., database tables).

5.4.2. Examples

There are many scenarios that would result in an update to the Schema metadata. These scenarios include but are not limited to:

- Adding a new column
- Removing a column
- Changing column metadata
- FOCUS Version has changed
- Data Generator Version has changed
- Correcting schema metadata errors

For an example of the FOCUS schema metadata, please refer to: Schema Metadata Example.

5.4.3. Metadata ID

Schema

5.4.4. Metadata Name

Schema

5.4.5. Version Introduced

1.0

5.4.6. Schema ID

The Schema ID provides the reference item to associate which Schema was used for the generation of a *FOCUS dataset*.

SchemaId adheres to the following requirements:

- SchemaId MUST be present in an object within the Schema collection.
- SchemaId MUST be of type String.
- SchemaID MUST NOT be null.
- SchemaId SHOULD be a Globally Unique Identifier (GUID).

5.4.6.1. Metadata ID

SchemaId

5.4.6.2. Metadata Name

Schema ID

5.4.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory

Constraint	Value
Allows nulls	False
Data type	String
Value format	GUID (recommended)

5.4.6.4. Version Introduced

1.0

5.4.7. Creation Date

Date the schema was created.

CreationDate adheres to the following requirements:

- CreationDate MUST be present in an object within the Schema collection.
- CreationDate MUST be of type Date/Time.
- CreationDate MUST conform to DateTimeFormat.
- CreationDate MUST NOT be null.

5.4.7.1. Metadata ID

CreationDate

5.4.7.2. Metadata Name

Creation Date

5.4.7.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	Date/Time
Value format	Date/Time Format

5.4.7.4. Version Introduced

1.0

5.4.8. FOCUS Version

The version of FOCUS utilized for building the dataset.

FocusVersion adheres to the following requirements:

- FocusVersion MUST be present in an object within the Schema collection.
- FocusVersion MUST be of type String.
- FocusVersion MUST NOT be null.
- FocusVersion MUST match one of the published versions of the FOCUS specification.
- FocusVersion MUST match the version of the FOCUS specification that the *dataset instance artifact* conforms to.

5.4.8.1. Metadata ID

FocusVersion

5.4.8.2. Metadata Name

FOCUS Version

5.4.8.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.8.4. Version Introduced

1.0

5.4.9. Data Generator Version

The DataGeneratorVersion may be supplied to declare the version of logic by which the *dataset instance artifact* was generated and is separate from FOCUS Version. DataGeneratorVersion allows for the provider to specify changes that may not result in a structural change in the data. It is suggested that the DataGeneratorVersion use a versioning approach such as SemVer version.

DataGeneratorVersion adheres to the following requirements:

- DataGeneratorVersion MAY be present in an object within the Schema collection.
- DataGeneratorVersion MUST be of type String.
- DataGeneratorVersion MUST conform to StringHandling requirements.
- DataGeneratorVersion MUST NOT be null.
- DataGeneratorVersion MUST be changed when FocusVersion is changed.
- Data generators MUST document what changes are present in the DataGeneratorVersion.

5.4.9.1. Metadata ID

DataGeneratorVersion

5.4.9.2. Metadata Name

Data Generator Version

5.4.9.3. Content Constraints

Constraint	Value
Feature level	Optional
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.9.4. Version Introduced

1.1

5.4.10. Dataset Instance ID

The Dataset Instance ID is a unique identifier for the specific dataset instance provided by the data generator. It identifies the dataset instance that this schema and the corresponding dataset artifacts are aligned with.

DatasetInstanceId adheres to the following requirements:

- DatasetInstanceId MUST be present in an object within the Schema collection.
- DatasetInstanceID MUST be of type String.
- DatasetInstanceID MUST NOT be null.
- DatasetInstanceID MUST be a unique identifier within a data generator.

5.4.10.1. Metadata ID

DatasetInstanceId

5.4.10.2. Metadata Name

Dataset Instance ID

5.4.10.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	GUID (recommended)

5.4.10.4. Version Introduced

1.3

5.4.11. Column Definition

The FOCUS metadata schema column definition provides a list of the columns present in the *dataset instance artifact* along with metadata about the columns.

5.4.11.1. Requirements

ColumnDefinition adheres to the following requirements:

- ColumnDefinition MUST be present in an object within the Schema collection.
- ColumnDefinition MUST be structured as a collection of objects.
- ColumnDefinition MUST NOT be null.
- ColumnDefinition collection MUST contain one and only one object for every column provided in *dataset instance artifacts* that reference the parent Schema object.
- ColumnDefinition collection MUST NOT contain null objects.

5.4.11.2. Metadata ID

ColumnDefinition

5.4.11.3. Metadata Name

Column Definition

5.4.11.4. Version Introduced

1.0

5.4.11.5. Column Name

The name of the column provided in the *FOCUS dataset*.

ColumnName adheres to the following requirements:

- ColumnName MUST be present in an object within the ColumnDefinition collection.
- ColumnName MUST be of type String.
- ColumnName MUST NOT be null.

5.4.11.5.1. Metadata ID

ColumnName

5.4.11.5.2. Metadata Name

Column Name

5.4.11.5.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.11.5.4. Version Introduced

1.0

5.4.11.6. Data Type

The data type of the column provided in the *FOCUS dataset*.

DataType adheres to the following requirements:

- DataType MUST be present in an object within the ColumnDefinition collection.
- DataType MUST be of type String.
- DataType MUST NOT contain null values.

5.4.11.6.1. Metadata ID

DataType

5.4.11.6.2. Metadata Name

Data Type

5.4.11.6.3. Content Constraints

Constraint	Value
Feature level	Mandatory
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.11.6.4. Version Introduced

1.0

5.4.11.7. Deprecated

The deprecation status of a column in a Dataset Instance.

Deprecated adheres to the following requirements:

- Deprecated MUST be present in an object within the ColumnDefinition collection when the column is planned for removal.
- Deprecated MUST be of type Boolean.
- Deprecated MUST NOT contain null values.
- Deprecated SHOULD only be "true" when the column is deprecated.
- Deprecated MUST be "true" when the data generator removes a column at a future date, or the column has been identified for deprecation for the FOCUS version identified in the schema definition.

5.4.11.7.1. Metadata ID

Deprecated

5.4.11.7.2. Metadata Name

Deprecated

5.4.11.7.3. Content Constraints

Constraint	Value
Feature level	Conditional
Allows nulls	False
Data type	Boolean
Value format	<not specified>

5.4.11.7.4. Version Introduced

1.2

5.4.11.8. Numeric Precision

Numeric Precision is the maximum number of digits for the values in the column.

NumericPrecision adheres to the following requirements:

- NumericPrecision SHOULD be present in an object within the ColumnDefinition collection when the column is of Decimal data type.
- NumericPrecision MUST be of type Integer.
- NumericPrecision MUST NOT contain null values.

5.4.11.8.1. Metadata ID

NumericPrecision

5.4.11.8.2. Metadata Name

Numeric Precision

5.4.11.8.3. Content Constraints

Constraint	Value
Feature level	Recommended
Allows nulls	False
Data type	Integer
Value format	Numeric Format

5.4.11.8.4. Version Introduced

1.0

5.4.11.9. Number Scale

The number scale of the data provides the maximum number of digits after the decimal point in decimal numbers.

NumberScale adheres to the following requirements:

- NumberScale SHOULD be present in an object within the ColumnDefinition collection when the column is of Decimal data type.
- NumberScale MUST be of type Integer.
- NumberScale MUST conform to NumericFormat requirements.
- NumberScale MUST NOT be null.

5.4.11.9.1. Metadata ID

NumberScale

5.4.11.9.2. Metadata Name

Number Scale

5.4.11.9.3. Content Constraints

Constraint	Value
Feature level	Recommended
Allows nulls	False
Data type	Integer
Value format	Numeric Format

5.4.11.9.4. Version Introduced

1.0

5.4.11.10. PreviousColumnName

The PreviousColumnName field indicates that on that schema the column where the key is included was renamed.

PreviousColumnName adheres to the following requirements:

- PreviousColumnName MUST be present in an object within the ColumnDefinition collection when the column was renamed.
- When PreviousColumnName is present, PreviousColumnName adheres to the following normative requirements:
  - PreviousColumnName MUST be of type String.
  - PreviousColumnName MUST not be null.
  - PreviousColumnName MUST be the name used in previous versions of the schema.
  - PreviousColumnName MUST NOT be present in schema versions created after the rename.

5.4.11.10.1. Metadata ID

PreviousColumnName

5.4.11.10.2. Metadata Name

Previous Column Name

5.4.11.10.3. Content Constraints

Constraint	Value
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.11.10.4. Version Introduced

1.2

5.4.11.11. Provider Tag Prefixes

The Provider Tag Prefixes define the list of prefixes used in the tag name of provider-defined tags. This metadata is useful for the consumer to identify which tags are provider-defined vs user-defined.

ProviderTagPrefixes adheres to the following requirements:

- ProviderTagPrefixes MUST be present in an object within the ColumnDefinition collection when ColumnName is "Tags".
- ProviderTagPrefixes MUST be of type Collection of Strings.
- ProviderTagPrefixes SHOULD be easily associated with the data generator who generated the *dataset instance* and the corresponding *dataset instance artifacts*.

5.4.11.11.1. Metadata ID

ProviderTagPrefixes

5.4.11.11.2. Metadata Name

Provider Tag Prefixes

5.4.11.11.3. Content Constraints

Constraint	Value
Feature level	Conditional
Allows nulls	False
Data type	Collection of Strings
Value format	<not specified>

5.4.11.11.4. Version Introduced

1.0

5.4.11.12. String Encoding

The string encoding scheme of the column provided in the *FOCUS dataset*.

StringEncoding adheres to the following requirements:

- StringEncoding MUST be present in an object within the ColumnDefinition collection when this information is required in order to successfully read the data.
- StringEncoding MUST be of type String.
- StringEncoding MUST NOT be null.

5.4.11.12.1. Metadata ID

StringEncoding

5.4.11.12.2. Metadata Name

StringEncoding

5.4.11.12.3. Content Constraints

Constraint	Value

Constraint	Value
Feature level	Conditional
Allows nulls	False
Data type	String
Value format	<not specified>

5.4.11.12.4. Version Introduced

1.0

5.4.11.13. String Max Length

The string max length of the data that can be stored in the column.

StringMaxLength adheres to the following requirements:

- StringMaxLength SHOULD be present in an object within the ColumnDefinition collection when the column is of String data type.
- StringMaxLength MUST be of type Integer.
- StringMaxLength MUST NOT be null.

5.4.11.13.1. Metadata ID

StringMaxLength

5.4.11.13.2. Metadata Name

String Max Length

5.4.11.13.3. Content Constraints

Constraint	Value
Feature level	Recommended
Allows nulls	False
Data type	Integer
Value format	Numeric Format

5.4.11.13.4. Version Introduced

1.0

6. Schemas

This section provides machine-readable JSON Schema definitions for validating the structure and content of JSON-formatted columns in FOCUS datasets.

6.1. Contract Commitment

6.1.1. Contract Commitment Applicability Object Schema

The Contract Commitment Applicability Object Schema defines the structure for the Contract Commitment Applicability column.

Download from GitHub


```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://focus.finops.org/schemas/contractcommitmentapplicabilityobject.json",
 "title": "Contract Commitment Applicability Object",
 "description": "Schema for validating the Contract Commitment Applicability column JSON object structure in FOCUS datasets",
 "type": "object",
 "$defs": {
  "ApplicabilityObject": {
   "type": "object",
   "properties": {
    "Cost": {
     "type": "number",
     "description": "Percentage applicable to ContractCommitmentCost",
     "minimum": 0.0,
     "maximum": 1.0,
     "default": 1.0
    },
    "Usage": {
     "type": "number",
     "description": "Percentage applicable to ContractCommitmentQuantity",
     "minimum": 0.0,
     "maximum": 1.0,
     "default": 1.0
    }
   },
   "additionalProperties": false
  },
  "ApplicabilityRule": {
   "type": "object",
   "required": [
    "Dimension",
    "Operator",
    "Values"
   ],
   "properties": {
    "Dimension": {
     "type": "string",
     "description": "A valid FOCUS Column Name"
    },
    "Operator": {
     "type": "string",
     "description": "The comparison logic to apply",
     "enum": [
      "In",
      "NotIn",
      "StartsWith",
```

```
        "NotStartsWith",
        "Contains",
        "NotContains",
        "EndsWith",
        "Exists",
        "DoesNotExist"
       ]
      },
      "Values": {
       "type": "array",
       "description": "List of strings to compare",
       "items": {
        "type": "string"
       },
       "minItems": 1
      },
      "Applicability": {
       "$ref": "#/$defs/ApplicabilityObject",
       "description": "Optional rule-level applicability that overrides top-level"
      }
     },
     "allOf": [
      {
       "description": "Wildcard constraint: If '*' is used, it must be the only value and Operator must be compatible.",
       "if": {
        "properties": {
         "Values": {
          "contains": {
           "const": "*"
          }
         }
        }
       },
       "then": {
        "properties": {
         "Values": {
          "maxItems": 1
         },
         "Operator": {
          "enum": [
           "In",
           "Contains",
           "Exists",
           "DoesNotExist"
          ]
         }
        }
       }
      },
      {
       "description": "Existence Logic: Exists/DoesNotExist operators require the specific wildcard value.",
       "if": {
        "properties": {
         "Operator": {
          "enum": [
           "Exists",
           "DoesNotExist"
          ]
         }
        }
       },
       "then": {
        "properties": {
         "Values": {
          "items": {
           "const": "*"
          },
          "maxItems": 1
```

```
          }
        }
      }
    }
  ],
  "additionalProperties": false
 }
},
"properties": {
 "IsGlobalScope": {
  "type": "boolean",
  "description": "If true, commitment applies to all entities",
  "default": false
 },
 "IsComplexScope": {
  "type": "boolean",
  "description": "If true, applicability logic exceeds schema capabilities",
  "default": false
 },
 "Applicability": {
  "$ref": "#/$defs/ApplicabilityObject",
  "description": "Top-level fractional mapping for metrics"
 },
 "InclusionOperator": {
  "type": "string",
  "description": "Logical operator for Inclusions array",
  "enum": [
   "And",
   "Or"
  ]
 },
 "Inclusions": {
  "type": "array",
  "description": "List of rules defining the applicability boundary",
  "items": {
   "$ref": "#/$defs/ApplicabilityRule"
  }
 },
 "ExclusionOperator": {
  "type": "string",
  "description": "Logical operator for Exclusions array",
  "enum": [
   "And",
   "Or"
  ]
 },
 "Exclusions": {
  "type": "array",
  "description": "List of rules defining entities to exclude",
  "items": {
   "$ref": "#/$defs/ApplicabilityRule"
  },
  "minItems": 1
 }
},
"allOf": [
 {
  "description": "IsGlobalScope and IsComplexScope cannot both be true simultaneously.",
  "not": {
   "required": [
    "IsGlobalScope",
    "IsComplexScope"
   ],
   "properties": {
    "IsGlobalScope": {
     "const": true
    },
```

```
      "IsComplexScope": {
        "const": true
      }
    }
  }
},
{
  "description": "If IsGlobalScope is true, Inclusions must be empty or omitted.",
  "if": {
    "properties": {
      "IsGlobalScope": {
        "const": true
      }
    },
    "required": [
      "IsGlobalScope"
    ]
  },
  "then": {
    "properties": {
      "Inclusions": {
        "maxItems": 0
      },
      "InclusionOperator": false
    }
  }
},
{
  "description": "If IsComplexScope is true, Inclusions must be empty or omitted.",
  "if": {
    "properties": {
      "IsComplexScope": {
        "const": true
      }
    },
    "required": [
      "IsComplexScope"
    ]
  },
  "then": {
    "properties": {
      "Inclusions": {
        "maxItems": 0
      },
      "InclusionOperator": false
    }
  }
},
{
  "description": "If neither Global nor Complex scope is active, Inclusions and InclusionOperator are required.",
  "if": {
    "not": {
      "anyOf": [
        {
          "properties": {
            "IsGlobalScope": {
              "const": true
            }
          },
          "required": [
            "IsGlobalScope"
          ]
        },
        {
          "properties": {
            "IsComplexScope": {
              "const": true
            }
```

```
          },
          "required": [
            "IsComplexScope"
          ]
        }
      ]
    }
   },
   "then": {
    "required": [
     "InclusionOperator",
     "Inclusions"
    ],
    "properties": {
     "Inclusions": {
      "minItems": 1
     }
    }
   }
  },
  {
   "description": "If Exclusions are present, ExclusionOperator is required.",
   "if": {
    "required": [
     "Exclusions"
    ]
   },
   "then": {
    "required": [
     "ExclusionOperator"
    ]
   }
  }
 ],
 "patternProperties": {
  "^x_[A-Z][a-zA-Z0-9]*$": {
   "description": "Custom data generator-defined properties prefixed with x_ in PascalCase format"
  }
 },
 "additionalProperties": false
}
```

6.2. Cost and Usage

6.2.1. Allocated Method Details Object Schema

The Allocated Method Details Object Schema defines the structure for theAllocated Method Details column.

Download from GitHub

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://focus.finops.org/schemas/allocatedmethoddetailsobject.json",
  "title": "Allocated Method Details Object",
  "description": "Schema for validating the Allocated Method Details column JSON object structure in FOCUS datasets",
  "type": "object",
  "required": [
    "Elements"
  ],
  "properties": {
    "Elements": {
      "type": "array",
```

```
        "description": "Array containing one or more objects describing allocation properties",
        "minItems": 1,
        "items": {
          "type": "object",
          "required": [
            "AllocatedRatio"
          ],
          "properties": {
            "AllocatedRatio": {
              "type": "number",
              "description": "Percentage of overall cost derived from corresponding method and metric",
              "minimum": 0,
              "maximum": 1
            },
            "UsageUnit": {
              "type": [
                "string",
                "null"
              ],
              "description": "Unit being measured used to calculate allocation"
            },
            "UsageQuantity": {
              "type": [
                "number",
                "null"
              ],
              "description": "Volume of UsageUnit consumed or used"
            }
          },
          "anyOf": [
            {
              "description": "Scenario 1: UsageQuantity is absent or explicitly null. UsageUnit can be anything (or absent).",
              "properties": {
                "UsageQuantity": {
                  "type": "null"
                }
              }
            },
            {
              "description": "Scenario 2: UsageQuantity is provided and is a number. UsageUnit MUST be provided and MUST be a string.",
              "required": [
                "UsageQuantity",
                "UsageUnit"
              ],
              "properties": {
                "UsageQuantity": {
                  "type": "number"
                },
                "UsageUnit": {
                  "type": "string"
                }
              }
            }
          ],
          "patternProperties": {
            "^x_[A-Z][a-zA-Z0-9]*$": {
              "description": "Custom data generator-defined properties prefixed with x_ in PascalCase format"
            }
          },
          "additionalProperties": false
        }
      }
    },
    "patternProperties": {
      "^x_[A-Z][a-zA-Z0-9]*$": {
        "description": "Custom data generator-defined top-level properties prefixed with x_ in PascalCase format"
      }
```

```
    },
    "additionalProperties": false
}
```

6.2.2. Commitment Program Eligibility Details Object Schema

The Commitment Program Eligibility Details Object Schema defines the structure for theCommitment Program Eligibility Details column.

Download from GitHub

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "https://focus.finops.org/schemas/commitmentprogrameligibilitydetailsobject.json",
    "title": "Commitment Program Eligibility Details Object",
    "description": "Schema for validating the Commitment Program Eligibility Details column JSON object structure in FOCUS datasets",
    "type": "object",
    "required": [
        "CommitmentPrograms"
    ],
    "properties": {
        "CommitmentPrograms": {
            "type": "array",
            "description": "Array of objects identifying commitment programs for which the usage is eligible.",
            "minItems": 1,
            "items": {
                "type": "object",
                "required": [
                    "ProgramType"
                ],
                "properties": {
                    "ProgramType": {
                        "type": "string",
                        "description": "The specific type of commitment program (e.g., discount or capacity reservation) available for this usage."
                    }
                },
                "patternProperties": {
                    "^x_[A-Z][a-zA-Z0-9]*$": {
                        "description": "Custom data generator-defined properties prefixed with x_ in PascalCase format"
                    }
                },
                "additionalProperties": false
            }
        }
    },
    "patternProperties": {
        "^x_[A-Z][a-zA-Z0-9]*$": {
            "description": "Custom data generator-defined top-level properties prefixed with x_ in PascalCase format"
        }
    },
    "additionalProperties": false
}
```

6.2.3. Contract Applied Object Schema

The Contract Applied Object Schema defines the structure for theContract Applied column.

Download from GitHub

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "https://focus.finops.org/schemas/contractappliedobject.json",
```

```
"title": "Contract Applied Object",
"description": "Schema for validating the Contract Applied column JSON object structure in FOCUS datasets",
"type": "object",
"required": [
  "Elements"
],
"properties": {
  "Elements": {
    "type": "array",
    "description": "Array containing one or more objects describing contract commitments applied to the charge",
    "minItems": 1,
    "items": {
      "type": "object",
      "required": [
        "ContractId",
        "ContractCommitmentId"
      ],
      "properties": {
        "ContractId": {
          "type": "string",
          "description": "Unique identifier for the contract"
        },
        "ContractCommitmentId": {
          "type": "string",
          "description": "Unique identifier for the contract commitment term"
        },
        "ContractCommitmentAppliedCost": {
          "type": [
            "number",
            "null"
          ],
          "description": "Cost value of the charge applied to the contract commitment"
        },
        "ContractCommitmentAppliedQuantity": {
          "type": [
            "number",
            "null"
          ],
          "description": "Quantity of usage applied to the contract commitment"
        },
        "ContractCommitmentAppliedUnit": {
          "type": [
            "string",
            "null"
          ],
          "description": "Unit of measure for the applied quantity"
        }
      },
      "oneOf": [
        {
          "description": "Scenario 1: Cost has a value. Quantity and Unit MUST be absent or explicitly null.",
          "required": [
            "ContractCommitmentAppliedCost"
          ],
          "properties": {
            "ContractCommitmentAppliedCost": {
              "type": "number"
            },
            "ContractCommitmentAppliedQuantity": {
              "type": "null"
            },
            "ContractCommitmentAppliedUnit": {
              "type": "null"
            }
          }
        },
        {
```

```
                "description": "Scenario 2: Quantity and Unit have values. Cost MUST be absent or explicitly null.",
                "required": [
                  "ContractCommitmentAppliedQuantity",
                  "ContractCommitmentAppliedUnit"
                ],
                "properties": {
                  "ContractCommitmentAppliedCost": {
                    "type": "null"
                  },
                  "ContractCommitmentAppliedQuantity": {
                    "type": "number"
                  },
                  "ContractCommitmentAppliedUnit": {
                    "type": "string"
                  }
                }
              }
            ],
            "patternProperties": {
              "^x_[A-Z][a-zA-Z0-9]*$": {
                "description": "Custom data generator-defined properties prefixed with x_ in PascalCase format"
              }
            },
            "additionalProperties": false
          }
        }
      },
      "patternProperties": {
        "^x_[A-Z][a-zA-Z0-9]*$": {
          "description": "Custom data generator-defined top-level properties prefixed with x_ in PascalCase format"
        }
      },
      "additionalProperties": false
    }
```

7. Glossary

Accrual-Based Accounting

An accounting method used in technology cost management to record costs in the period when resources are utilized, services are delivered, or entitlements are available for or actually consumed (including unused or expired entitlements). This approach ensures that expenses are recognized in the same timeframe as the value they provide, independent of when charges are invoiced.

Adjustment

A charge representing a modification to billing data to account for certain events or circumstances not previously captured, or captured incorrectly. Examples include billing errors, service disruptions, or pricing changes.

Allocated Charge

The charge that was created as the result of an allocation operation. This is used in the context of Data Generator-Calculated Split Cost Allocation to identify the charges that were created from the origin charge resulting from the application of Data Generator-Calculated Split Cost Allocation.

Allocated Method

The process or formula by which cost is being allocated from an origin charge to produce allocated charges. This is used in the context of Data Generator-Calculated Split Cost Allocation which requires documentation of the method to be provided for any and all allocated methods used. May also be colloquially referred to as allocation method.

Amortization

The distribution of upfront costs over time to accurately reflect the consumption or benefit derived from the associated resources or services. Amortization is valuable when the commitment *period* extends beyond the granularity of the source report.

Availability Zone

A collection of geographically separated locations containing a data center or cluster of data centers. Each availability zone (AZ) should have its own power, cooling, and networking, to provide redundancy and fault tolerance.

Billing Account

A container for resources and/or services that are billed together in an invoice. A billing account may have sub accounts, all of whose costs are consolidated and invoiced to the billing account.

Billing Currency

An identifier that represents the currency that a charge for resources and/or services was billed in.

Billing Period

The time window that an organization receives an invoice for, inclusive of the start date and exclusive of the end date. It is independent of the time of usage and consumption of resources and services.

Block Pricing

A pricing approach where the cost of a particular resource or service is determined based on predefined quantities or tiers of usage. In these scenarios, the Pricing Unit and the corresponding Pricing Quantity can be different from the Consumed Unit and Consumed Quantity.

Cash-Based Accounting

An accounting method used in technology cost management to record costs in the period when charges are invoiced. This approach aligns expenses with billing cycles, independent of when resources are utilized, services are delivered, or entitlements are available for or actually consumed (including unused or expired entitlements).

Capacity Reservation

A capacity reservation is an agreement that secures a dedicated amount of resources or services for a specified period. This ensures the reserved capacity is always available and accessible, even if it's not fully utilized. Customers are typically charged for the reserved capacity, regardless of actual consumption.

Charge

A row in a FOCUS-compatible cost and usage dataset.

Charge Period

The time window for which a charge is effective, inclusive of the start date and exclusive of the end date. The charge period for continuous usage should match the time granularity of the *dataset instance* (e.g., 1 hour for hourly, 1 day for daily). The charge period for a non-usage charge with time boundaries should match the period of eligibility.

Closed Billing Period

A *billing period* with Billing Period Status set to "Closed". The period has been financially closed after all anticipated invoices for the period have been *issued* by designated *invoice issuers* and no additional invoices will be associated with this period. Exceptionally, additional invoices may be associated with a closed billing period if explicitly requested or approved by the customer.

Cloud Service Provider (CSP)

A company or organization that provides remote access to computing resources, infrastructure, or applications for a fee.

Commitment

A customer's agreement to either spend a defined monetary amount or consume a specific quantity of resources or services over a specified *period*.

Commitment Discount

A publicly available *contract commitment* that provides discounted pricing on preselected SKUs in exchange for a commitment to specific spend or usage goals over a specified *period*, based on publicly disclosed standard terms and pricing. Committed spend or usage is evenly distributed across predefined Contract Commitment Fulfillment Intervals (e.g., hourly), and unused benefits cannot be carried over to subsequent Intervals. Only one commitment discount may be applied to a single charge at a time, and it may overlap with one or more *negotiated discounts*.

This term has been used in the past by FinOps teams managing Public Cloud to refer to specific *contract commitments* (including Reserved Instances, Savings Plans and Committed Use Discounts) offered by cloud providers, and is defined here to maintain consistency with the terms of that subset of *contract commitments*.

Commitment discounts are classified with the following designations:

- Contract Commitment Offer Category: "Public"
- Contract Commitment Benefit Category: "Discount"
- Contract Commitment Model: "Continuous"
- Unused benefits cannot be carried over to subsequent Contract Commitment Fulfillment Intervals

Commitment Discount Flexibility

A feature of *commitment discounts* that may further transform the predetermined amount of usage purchased or consumed based on additional, service-provider-specific requirements.

Commitment Program

A service-provider offering that allows a customer to enter into a *commitment*. Commitment programs include *commitment discounts*, *capacity reservations*, and other constructs that require advance spend or usage agreements.

Contract

A collection of agreed terms between a service provider and a customer.

Contract Commitment

A specific term within a *contract* that defines a measurable obligation agreed upon by a provider and a customer, such as a minimum spend or usage over an agreed period of time.

Contracted Unit Price

The agreed-upon unit price for a single Pricing Unit of the associated SKU, inclusive of negotiated discounts, if present, and exclusive of any other discounts. This price is denominated in the Billing Currency.

Correction

Any modification (including updates, additions, or omissions) to previously delivered records within a defined *delivery scope* (e.g., temporal grouping such as a *billing period* or non-temporal, logical grouping such as a *contract*).

A correction (lowercase) may consist of one or more simultaneous changes, including updates to or omission of previously delivered records, or the addition of new records that supplement previously delivered data within the affected *delivery scope*. This concept applies across all FOCUS datasets.

In contrast to the broader concept of correction (lowercase), the term "Correction" (capitalized) refers to a specific allowed value in the Charge Class column in Cost and Usage datasets. It designates *charges* used to correct cost or usage data from a previously *closed billing period*.

Covered Charge

A *charge* whose Billed Cost is fully or partially absorbed by a corresponding *covering charge*. Common examples include usage

charges applied against *commitment discounts*, or consumption of SaaS offerings drawn from a prepayment, such as marketplace purchases.

Covering Charge

A purchase *charge* whose cost is applied against one or more *covered charges*, offsetting their Billed Cost. Common examples include *commitment discount* purchases, prepayment charges, and marketplace purchases that cover consumption-based usage.

Credit

A financial incentive or allowance granted by a service provider unrelated to other past/current/future charges.

Custom Column

A column not defined by FOCUS and included in a *FOCUS dataset*. Custom columns are prefixed with `x_` and provide additional context from *native datasets* beyond what is captured in FOCUS columns. See Dataset Completeness for inclusion requirements.

Dataset Artifact

An abbreviated term for *dataset instance artifact*.

Dataset Instance

A specific implementation of a *FOCUS dataset* provided by a data generator. A Data Generator may provide multiple dataset instances of the same *FOCUS dataset*, each with different properties such as time granularity or differing custom column inclusions. For example, the same 'FOCUS Cost and Usage' *FOCUS dataset* may be provided at an hourly or daily time granularity by a Data Generator. Each would be a distinct Dataset Instance.

Dataset Instance Artifact

A physical representation of a specific *dataset instance* delivered by a data generator.

Delivery Scope

A dataset-specific boundary or set of boundaries that determines which records are included in a *dataset artifact* delivery. Scopes can be temporal (e.g., a *billing period*) or non-temporal (e.g., a *contract* or other logical grouping), and multiple scopes may be applicable for a single dataset depending on use case or delivery configuration. Scopes determine how Overwrite and Append *dataset artifact* deliveries, as well as corrections, are applied.

Dimension

A specification-defined categorical attribute that provides context or categorization to billing data.

Exclusive End Bound

A Date/Time Format value that is not contained within the ending bound of a time period.

Finalized Tag

A tag with one tag value chosen from a set of possible tag values after being processed by a set of service-provider-defined or user-defined rules.

FinOps Cost and Usage Specification (FOCUS)

An open-source specification that defines requirements for billing data.

FOCUS Column

A column defined by FOCUS and included in a *FOCUS dataset*. See the Columns section of each dataset for definitions.

FOCUS Dataset

A structured collection of columns that conforms to the BCP14 criteria established by FOCUS. All columns included must be defined in the FOCUS Columns section of the specification.

In addition to these standardized columns, data generators include *custom columns* to capture information from *native datasets* that is not represented by *FOCUS columns*. If custom columns introduce record-splitting (i.e., a single original charge results in multiple rows), the data generator is responsible for ensuring that all cost and quantity metrics still meet the aggregation and consistency rules required by the specification.

The collection of datasets are designed to provide billing insight, additional context, metadata, mapping, or enrichment information that enhances the interpretability or completeness. See also: *Native Dataset*.

FOCUS Dataset Column

A column included in a *FOCUS dataset*. A FOCUS dataset column is either a *FOCUS column* or a *custom column*.

Inclusive Start Bound

A Date/Time Format value that is contained within the beginning bound of a time period.

Interruptible

A category of compute resources that can be paused or terminated by the CSP within certain criteria, often advertised at reduced unit pricing when compared to the equivalent non-interruptible resource.

Invoice

A document that summarizes the charges for resources or services consumed by a customer.

Invoice Issuer

An entity responsible for issuing payable *invoices* for the *resources* or *services* consumed. Common examples include *cloud service providers*, *managed service providers*, or marketplace operators.

Invoice Reconciliation

The process of verifying that the costs, quantities, and applied adjustments on an *invoice* are equal to the underlying usage records and contracted rates for a specific *billing period*.

In the context of FOCUS, this process ensures consistency by reconciling cost and usage data across the invoice, the Invoice Detail dataset, and the Cost and Usage dataset to identify and resolve any discrepancies.

Issued Invoice

An *invoice* that has been formally *reconciled* and issued by the designated *invoice issuer* (Invoice Issue Status set to "Issued"). Once issued, the invoice becomes the authoritative financial document and is considered finalized. The financial data presented on the invoice is not expected to change.

JSON

A common acronym for JavaScript Object Notation, a data format codified in ECMA-404 as a standard for human-readable, serializable data objects. This data format is used in FOCUS to communicate multiple pieces of information about a charge (tags, properties, etc.) in a single column.

List Unit Price

The suggested service-provider-published unit price for a single Pricing Unit of the associated SKU, exclusive of any discounts. This price is denominated in the Billing Currency.

Managed Service Provider (MSP)

A company or organization that provides outsourced management and support of a range of IT services, such as network infrastructure, cybersecurity, cloud computing, and more.

Metric

A FOCUS-defined column that provides numeric values, allowing for aggregation operations such as arithmetic operations (sum, multiplication, averaging etc.) and statistical operations.

National Currency

A government-issued currency (e.g., US dollars, Euros).

Native Dataset

A dataset provided by a *data generator* in a format other than FOCUS. For *service providers*, this typically refers to their proprietary data exports (e.g., billing exports, contract details, or other FinOps-related datasets). For FinOps tool vendors, this refers to any non-FOCUS dataset they offer to *practitioners*. See also: *FOCUS Dataset*.

Negotiated Discount

A privately agreed *contract commitment* that provides discounted pricing in exchange for a commitment to specific spend or usage goals over a specified *period*, with terms and pricing specifically modified from standard. Multiple negotiated discounts may be applied to a single charge at a time, and they may overlap with a *commitment discount*.

Negotiated discounts are classified with the following designations:

- Contract Commitment Offer Category: "Negotiated"
- Contract Commitment Benefit Category: "Discount"

On-Demand

A service that is available and provided immediately or as needed, without requiring a pre-scheduled appointment or prior arrangement. In cloud computing, virtual machines can be created and terminated as needed, i.e., on demand.

Open Billing Period

A *billing period* with Billing Period Status set to "Open". Billing activities are ongoing, and the period remains subject to

updates until formally closed.

Origin Charge

The charge that existed prior to an operation. This is used in the context of Data Generator-Calculated Split Cost Allocation to identify the charge that existed prior to the application of Data Generator-Calculated Split Cost Allocation to produce allocated charges.

Pascal Case

Pascal Case (PascalCase, also known as UpperCamelCase) is a format for identifiers which contain one or more words meaning the words are concatenated together with no delimiter and the first letter of each word is capitalized.

Period

A time window, with a specifically defined start and end date/time.

Potato

A long and often painful conversation had by the FOCUS contributors. Sometimes the name of a thing that we could not yet name. No starchy root vegetables were harmed during the production of this specification. We thank potato for its contribution in the creation of this specification.

Practitioner

An individual who performs FinOps within an organization to maximize the business value of using cloud and cloud-like services.

Price List

A comprehensive list of prices offered by a service provider.

Service Provider

An entity that provides the *resources* or *services* available for usage or purchase.

Refund

A return of funds that have previously been charged.

Resource

A unique component that incurs a charge.

Row

A row in a FOCUS-compatible cost and usage dataset.

Service

An offering that can be purchased from a service provider, and can include many types of usage or other charges; e.g., a cloud database service may include compute, storage, and networking charges.

SKU

A construct composed of the common properties of a product offering associated with one or many SKU Prices.

SKU Price

A pricing construct that encompasses SKU properties (e.g., functionality and technical specifications), along with core stable pricing details for a particular SKU, while excluding dynamic or negotiable pricing elements such as unit price amounts; currency (and related exchange rates); temporal validity (e.g., effective dates); and contract- or negotiation-specific factors (e.g., contract or account identifiers and negotiable discounts).

Sub Account

A sub account is an optional service-provider-supported construct for organizing resources and/or services connected to a billing account. Sub accounts must be associated with a billing account as they do not receive invoices.

Tag

A metadata label assigned to a resource to provide information about it or to categorize it for organizational and management purposes.

Tag Scheme

A distinct framework for assigning metadata to charges, resources, or other assets. Different metadata types within a single provider (e.g., "tags" vs. "labels") are considered separate schemes. They can be user-defined (i.e., allowing user input) or provider-defined (i.e., controlled by the service provider).

Tag Source

A Resource or Service-Provider-defined construct for grouping resources and/or other Service-Provider-defined construct that a Tag can be assigned to.

Term

An agreement specified on a *contract* or *invoice*.

Virtual Currency

A proprietary currency (e.g., credits, tokens) issued by service providers and independent of government regulation.

8. Appendix

Note: The following section is informative and non-normative. It does not define requirements.

Appendix Entries

Topic	Description
Discount Handling	Explains how discounts are represented and applied to charges in a FOCUS dataset.
Examples: Commitment Discounts	Explains the purchasing, usage, and amortization of commitment discounts in a FOCUS dataset.
Examples: Commitment Discount Flexibility	Demonstrates scenarios for usage-based commitment discounts with and without commitment discount flexibility.
Examples: Commitment Program Eligibility Details	Demonstrates how commitment program eligibility details interact with capacity reservation columns for capacity reservation programs.
Examples: Contract Commitments	Provides a structured representation and examples of commercial agreements between a customer and their service providers.
Examples: Invoice Detail	Demonstrates scenarios for issuing invoices, including typical cloud invoices, multi-currency settlements, and billing error corrections.
Examples: JSON Object	Provides examples for columns using the JSON Object Format, such as Contract Commitment Applicability.
Examples: Metadata	Contains JSON payload examples for updating Data Generator, Dataset, Schema, and Recency metadata.
Examples: Participating Entity Identification	Illustrates how to identify the roles of participating entities (e.g., Service Provider, Invoice Issuer, Host Provider, Data Generator) across various supply chain scenarios.
Examples: SaaS	Illustrates how to model SaaS billing scenarios, including simple SaaS agreements, SaaS spend agreements, and virtual currency pricing models.
Grouping Constructs for Resources or Services	Outlines and compares the two distinct levels of resource or service grouping mechanisms supported by FOCUS: billing accounts and sub accounts.
Invoice and Billing Period Handling	Outlines invoice reconciliation, invoice issuance, and open vs. closed billing periods across FOCUS datasets, including correction handling.
Rounding Variance Tolerance	Defines the statistical tolerance formula and provides scenarios for handling precision differences during invoice reconciliation between detailed cost data and invoices.

Fictitious Data Generator Reference

To illustrate how FOCUS normalizes the presentation of data across diverse technology environments, the appendix uses a standardized set of fictitious *data generators*. These represent common architectural components, ranging from core cloud infrastructure to SaaS platforms. Using these examples demonstrates cross-vendor cost allocation, standardized billing schemas, and multi-cloud reporting without relying on proprietary vendor data.

Disclaimer: *The fictitious entities (data generators, customers, and commitment programs) referenced in this appendix are intended solely for illustrative purposes to resemble real-world services and organizations. They do not reflect, represent, or imply the actual current or future FOCUS implementations, billing schemas, or data formats of any real-world companies or equivalents listed herein.*

The table below outlines the fictitious *data generators* used throughout the appendix, their primary functions, and their real-world counterparts for context:

Fictitious Data Generator	Service Offering	Fictitious Data Generator Description	Similar Real-World Examples
Aura Web	Cloud Service Provider	A highly scalable, market-leading cloud infrastructure provider offering extensive compute, storage, and serverless options.	Amazon Web Services (AWS)
CrestNode	Cloud Service Provider	An enterprise-focused cloud platform with deep integrations into existing corporate software ecosystems and directory services.	Microsoft Azure
LatticeScale	Cloud Service Provider	A cloud provider heavily optimized for machine learning, data analytics, and containerized Kubernetes workloads.	Google Cloud Platform (GCP)

Fictitious Data Generator	Service Offering	Fictitious Data Generator Description	Similar Real-World Examples
OmniQuery	Data Platform	A centralized hub for storing, processing, and analyzing massive datasets to drive business intelligence.	Snowflake, Databricks
StackLens	SaaS Observability	A monitoring tool that tracks application performance, logs, and system health in real-time to prevent downtime.	Datadog, New Relic
SprintCanvas	Project Management	A collaborative workspace for planning, assigning, and tracking team tasks and agile workflows.	Jira, Asana, Trello
StoreStack	Database as a Service	A fully managed, scalable cloud database solution that handles provisioning, backups, and routine maintenance.	MongoDB Atlas
CollabChat	Team Communications	A messaging platform offering organized chat channels, direct messaging, and secure file sharing for remote teams.	Slack
PulseMail	Email API	A developer-friendly service for reliably routing, sending, and tracking both transactional and marketing emails.	SendGrid, Mailgun
PipelCRM	CRM	A customer relationship management platform designed to track sales pipelines, manage contacts, and optimize lead conversion.	Salesforce, HubSpot
Budget Beacon	Cost Management	A cloud cost-optimization platform that shines a spotlight on overspending, waste, and savings opportunities across multi-cloud environments.	Cloudability, CloudHealth, ProsperOps
SchemaWeaver	Open Source Library	An open-source tool that refines raw cloud cost and usage data, normalizing it into FOCUS-compliant schemas for downstream analytics and reporting. Not a public service.	OpenCost, Cloud Intelligence Dashboards, FinOps toolkit

Fictitious Customer Reference

To contextualize the billing and cost allocation examples, this appendix utilizes fictitious customer profiles. These profiles represent common organizational structures and cloud adoption patterns.

Fictitious Customer	Company Profile	Customer Description
Acme Corp	Large Enterprise	A traditional multinational corporation undergoing a major cloud transformation. They manage a complex, hybrid multi-cloud environment with strict regulatory and compliance requirements.
AeroScale	Cloud-Native Startup	A fast-growing tech startup operating entirely in the cloud. They heavily utilize serverless architectures, managed databases, and agile deployment pipelines.
GearPeak Outdoors	Mid-Market Retailer	An outdoor apparel and equipment brand with massive seasonal traffic spikes. They leverage auto-scaling infrastructure for their e-commerce storefront and a heavy mix of SaaS for supply chain and CRM.

Fictitious Commitment Program Reference

To illustrate commitment program application and amortization without relying on vendor-specific terminology, the examples in this appendix use standardized fictitious commitment instruments. These constructs abstract the common commitment mechanisms used by major cloud and SaaS providers.

Fictitious Commitment Program	Acronym	Category	Fictitious Commitment Program Description	Similar Real-World Programs
Resource Reservation	RR	Usage	An upfront commitment to use a specific resource type, family, and region for a set term (e.g., 1 or 3 years) in exchange for a significantly reduced hourly rate.	Reserved Instances (AWS/Azure), Resource-based CUDs (GCP)
Flexible Spend Plan	FSP	Spend	A commitment to spend a specific monetary amount per hour across a broad category of compute or service offerings, providing high flexibility as workloads shift.	Savings Plans (AWS)

Fictitious Commitment Program	Acronym	Category	Fictitious Commitment Program Description	Similar Real-World Programs
Dynamic Compute Commitment	DCC	Spend	A spend-based commitment covering aggregate compute resources (such as vCPU and memory) across multiple regions and machine families, converting the hourly spend into a usage discount.	Flexible CUDs (GCP)
Enterprise Spend Agreement	ESA	Spend	An overarching, contractual agreement where an organization commits to a minimum aggregate spend across a provider's portfolio over a set term (e.g., 1-3 years) in exchange for a blanket percentage discount.	Enterprise Discount Programs (AWS), MACC (Azure)
Interval Spend Commitment	ISC	Spend	A recurring minimum-spend or minimum-usage agreement at fixed intervals (e.g., monthly, annual), common among SaaS observability and infrastructure monitoring providers. Program names inherently include the period reference.	Monthly/Annual Commitments (Datadog)
Bulk Capacity Credit	BCC	Spend	A pre-purchased pool of platform-specific credits or capacity units, consumed against usage over a contract period. Common among data and analytics platforms.	Capacity Commitments (Snowflake), Committed Use Discounts (Databricks)
Advance Resource Commitment	ARC	Usage	An advance reservation of specific compute capacity in a region or availability zone, guaranteeing resource availability without necessarily providing a unit discount. Distinct from Resource Reservations, which are commitment discounts.	Capacity Reservations (AWS), On-demand Capacity Reservations (Azure), Zonal reservations (GCP)

8.1. Discount Handling

While *service providers* may use different terms and mechanisms to describe and represent discounts in their native datasets, FOCUS represents discounts as reduced pricing reflected directly in the *charges* they pertain to, rather than as separate offsetting *charges*. Unlike discounts, *credits* are financial incentives or allowances represented as separate *charges* unrelated to other *charges*.

In Cost and Usage, cost and unit price columns reflect the effect of discounts. Discounts associated with *contract commitments* are further identified and attributed using the Contract Applied column (also serves as an association to the Contract Commitment dataset), as well as *commitment discount*-specific columns. *Custom columns* may be used when needed to further identify or describe discounts.

For additional context and related normative requirements associated with discount handling, refer to the following specification sections:

Concept	Description	Entity	Specification Location
General dataset rules	Handling unused portions, splitting discounted charges, and avoiding negating rows	Dataset	CostAndUsage
Effect of discounts	Reflecting reduced pricing from discounts	Column	CostAndUsage.BilledCost CostAndUsage.EffectiveCost CostAndUsage.ContractedCost CostAndUsage.ListCost CostAndUsage.ContractedUnitPrice CostAndUsage.ListUnitPrice
Commitment Discount specifics	Identification and utilization of *commitment discounts*	Column	CostAndUsage.CommitmentDiscountId CostAndUsage.CommitmentDiscountStatus
Resource tracking	Identifying the exact resource that received the commitment discount	Column	CostAndUsage.ResourceId
Contract allocations	Associating a charge with specific contract commitments	Column	CostAndUsage.ContractApplied

8.2. Examples: Commitment Discounts

This appendix section defines the concept of a *commitment discount*. It then lays out a series of FOCUS dataset examples.

8.2.1. Overview

A *commitment discount* is a billing discount model that offers reduced rates on preselected *SKUs* in exchange for an obligated usage or spend amount over a specified *period*. *Commitment discounts* typically consist of purchase and usage records within cost and usage datasets.

Usage-based *commitment discounts* obligate a customer to a predetermined amount of usage over a specified *period*. In some cases, usage-based *commitment discounts* also feature *commitment discount flexibility* which may expand the types of *resources* that a *commitment discount* can cover. It is important to note when mixing *commitment discounts* with and without *commitment discount flexibility*, the CommitmentDiscountUnit should reflect this difference.

Spend-based *commitment discounts* obligate a customer to a predetermined amount of spend over a specified *period*. In the usage examples below, each *row* measures the monetary amount of the hourly commit consumed by the *commitment discount*, so the CommitmentDiscountUnit chosen is "USD", or the *billing currency*.

8.2.1.1. Purchasing

While customers are bound to the *period* of a *commitment discount*, service providers offer some or all of the following payment options before and/or during the *period*:

- *All Upfront* - The *commitment discount* is paid in full before the *period* begins.
- *No Upfront* - The *commitment discount* is paid on a repeated basis, typically over each *billing period* of the *period*.
- *Partial Upfront* - Some of the *commitment discount* is paid before the *period* begins, and the rest is paid repeatedly over the *period*.

For example, if a customer buys a 1-year, spend-based *commitment discount* with a $1.00 hourly commit and pays with the partial option, the *commitment discount's* payment consists of a one-time purchase in the beginning of the *period and* monthly recurring purchases. The one-time payment covers half of the annual commitment (Flexible Spend Plans are half, Resource Reservations are a portion of the cost), while the recurring payment covers the remaining half and is calculated based on the exact number of hours in each *billing period*:

1. *One-Time* - $4,380 (24 hours × 365 days × $1.00 × 0.5)
2. *Recurring* - $336.00 for February (672 hours in the month × $1.00 × 0.5)

8.2.1.2. Usage

Commitment discounts follow a "use-it-or-lose-it" model where the *amortization* of a *commitment discount's* purchase applies evenly to eligible *resources* over each *charge period* of the *period*.

For example, if a customer buys a spend-based *commitment discount* with a $1.00 hourly commit in January (31 days), only $1.00 is eligible for consumption for each hourly *charge period*. If a customer has eligible *resources* running during this *charge period*, an amount of up to $1.00 will be allocated to these *resources*. Conversely, if a customer does not have eligible *resources* running that fully take advantage of this $1.00 during this *charge period*, then some or all of this amount will go to waste.

8.2.2. Data Generator Scenarios

Below are tables listing some common commitment discount scenarios for a few prominent FOCUS data generators.

8.2.2.1. Data Generator Scenarios: Aura Web

Scenario	What You'll Learn

Scenario	What You'll Learn
Resource Reservation - All Upfront	How a single large upfront purchase amortizes across usage rows. BilledCost=0 on usage rows; EffectiveCost carries the amortized value. Usage-based commitment (`CommitmentDiscountCategory=Usage`).
Resource Reservation - Partial Upfront	How partial upfront splits into two purchase rows: one `One-Time` and one `Recurring`. Demonstrates the hybrid payment model for Resource Reservations (RRs).
Flexible Spend Plan - All Upfront	How Flexible Spend Plans (FSPs) differ from RRs: `CommitmentDiscountCategory=Spend` (dollar-based) instead of `Usage` (instance-based). Quantities measured in USD, not instance hours.
Flexible Spend Plan - Partial Upfront	The partial upfront pattern applied to FSPs. Two purchase rows (one-time + recurring) mirror the RR partial model but with spend-based commitment fields.
Flexible Spend Plan - No Upfront	The no-upfront model: only a `Recurring` purchase row, no initial capital outlay. Compare the higher effective rate against all-upfront and partial-upfront variants.
Flexible Spend Plan - 100% Utilization with Overage	What happens when demand exceeds commitment capacity. Committed hours apply the effective unit price; overage hours spill to standard pricing at full list price (`PricingCategory=Standard`).
Flexible Spend Plan - 75% Utilization	Moderate underutilization: 18 hours `Used`, 6 hours `Unused`. Unused rows carry EffectiveCost with null resource fields, representing wasted spend.
Flexible Spend Plan - 50% Utilization	Significant underutilization: 12 hours `Used`, 12 hours `Unused`. Half the commitment value is wasted - key pattern for identifying commitment right-sizing opportunities.
Flexible Spend Plan - 0% Utilization	Worst case: commitment purchased but never applied. All 24 hours show `Unused` status. EffectiveCost accrues entirely as waste.

8.2.2.2. Data Generator Scenarios: CrestNode

Scenario	What You'll Learn
Resource Reservation - All Upfront	CrestNode's usage-based reservation model in FOCUS format. Compare structure and rates against Aura Web Resource Reservations.
Resource Reservation - No Upfront	CrestNode no-upfront reservations with monthly recurring payments only. Note the higher effective rate vs all-upfront, reflecting the deferred-payment premium.
Flexible Spend Plan - All Upfront	CrestNode's spend-based Flexible Spend Plan (`CommitmentDiscountCategory=Spend`). Compare against CrestNode Resource Reservations and Aura Web Flexible Spend Plans.

8.2.2.3. Data Generator Scenarios: LatticeScale

Scenario	What You'll Learn
Resource Reservation - No Upfront	LatticeScale's usage-based commitment: `CommitmentDiscountCategory=Usage`, quantities in `Hours`. Monthly billing only (no upfront option). Deepest discount.
Dynamic Compute Commitment - No Upfront	LatticeScale's spend-based commitment: `CommitmentDiscountCategory=Spend`, quantities in `USD`. Monthly recurring billing, no upfront payment. Compare against Aura Web and CrestNode.

8.2.3. Aura Web Resource Reservation - All Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Usage
Utilization	100%
Hours Generated	24
Annual Commitment	$402,960.00

Parameter	Value
List Unit Price	$69.00/hour

CSV Example

8.2.3.1. Scenario Description

This example shows an **Aura Web Resource Reservation**, which is a commitment (with a Commitment Discount Category of `Usage`) where you commit to a specific quantity of resource capacity (e.g., instance hours).

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.3.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$402,960.00	$0.00
Usage (Used)	24	$0.00	$1,104.00
Total	25	**$402,960.00**	**$1,104.00**

8.2.3.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.3.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	1 (commitment units)

For usage-based commitments: CommitmentDiscountQuantity represents the quantity of resources (e.g., instance hours), not a dollar amount. For a 1-hour reservation, CommitmentDiscountQuantity = 1.

8.2.3.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$69.00
ContractedUnitPrice	Negotiated unit price	$69.00

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount

savings are reflected in EffectiveCost, not in unit prices.

8.2.3.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$402,960.00	$0.00	$402,960.00
Used Row	$0.00	$46.00	$69.00

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.3.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$402,960.00	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	8760.00	Total commitment capacity for the 1-year term (1 instance-hr/hr × 8,760 hrs/yr)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.3.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$46.00	Amortized cost (annual / hours)
ListCost	$69.00	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	1	Commitment units applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:compute:us-east-1:123456789012:resource-reserv...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.4. Aura Web Resource Reservation - Partial Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	Partial Upfront
Commitment Discount Category	Usage
Utilization	100%
Hours Generated	24
Annual Commitment	$440,014.80
List Unit Price	$75.35/hour

CSV Example

8.2.4.1. Scenario Description

This example shows an **Aura Web Resource Reservation**, which is a commitment (with a Commitment Discount Category of `Usage`) where you commit to a specific quantity of resource capacity (e.g., instance hours).

The **Partial Upfront** payment option combines an initial upfront payment with a reduced recurring monthly fee. This results in two Purchase rows: one One-Time for the upfront portion and one Recurring for the monthly fee, both with zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.4.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	2	$236,884.68	$0.00
Usage (Used)	24	$0.00	$1,205.52
Total	26	**$236,884.68**	**$1,205.52**

8.2.4.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.4.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	1 (commitment units)

For usage-based commitments: CommitmentDiscountQuantity represents the quantity of resources (e.g., instance hours), not a dollar amount. For a 1-hour reservation, CommitmentDiscountQuantity = 1.

8.2.4.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$75.35
ContractedUnitPrice	Negotiated unit price	$75.35

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.4.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row (One-Time)	$220,007.40	$0.00	$220,007.40
Purchase Row (Recurring)	$16,877.28	$0.00	$16,877.28
Used Row	$0.00	$50.23	$75.35

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.4.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$220,007.40	Upfront portion (50% of annual commitment)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	8760.00	Full commitment capacity for the 1-year term (1 instance-hr/hr × 8,760 hrs/yr)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.4.5. Recurring Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$16,877.28	Monthly portion (hourly rate / 2 × 672 hours in Feb)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	672.00	Commitment capacity for Feb (1 instance-hr/hr × 672 hrs)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)

Column	Value	Explanation
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-RECURRING	Price point for recurring purchase

8.2.4.6. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$50.23	Amortized cost (annual / hours)
ListCost	$75.35	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	1	Commitment units applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:compute:us-east-1:123456789012:resource-reserv...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.5. Aura Web Flexible Spend Plan - All Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	100%
Hours Generated	24
Annual Commitment	$628,004.40
List Unit Price	$107.54/hour

CSV Example

8.2.5.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.5.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$628,004.40	$0.00
Usage (Used)	24	$0.00	$1,720.56
Total	25	**$628,004.40**	**$1,720.56**

8.2.5.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.5.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	71.69 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $71.69/hour commitment, this value is $71.69.

8.2.5.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$107.54
ContractedUnitPrice	Negotiated unit price	$107.54

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.5.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$628,004.40	$0.00	$628,004.40
Used Row	$0.00	$71.69	$107.54

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.5.4. Purchase Row Details

Column	Value	Explanation

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$628,004.40	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	628,004.40	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	628,004.40	Full annual commitment ($71.69/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.5.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$71.69	Amortized cost (annual / hours)
ListCost	$107.54	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	71.69	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.6. Aura Web Flexible Spend Plan - Partial Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	Partial Upfront
Commitment Discount Category	Spend
Utilization	100%
Hours Generated	24
Annual Commitment	$447,986.40
List Unit Price	$76.71/hour

CSV Example

8.2.6.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **Partial Upfront** payment option combines an initial upfront payment with a reduced recurring monthly fee. This results in two Purchase rows: one One-Time for the upfront portion and one Recurring for the monthly fee, both with zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.6.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	2	$241,176.24	$0.00
Usage (Used)	24	$0.00	$1,227.36
Total	26	**$241,176.24**	**$1,227.36**

8.2.6.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.6.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	51.14 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $51.14/hour commitment, this value is $51.14.

8.2.6.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$76.71
ContractedUnitPrice	Negotiated unit price	$76.71

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.6.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row (One-Time)	$223,993.20	$0.00	$223,993.20
Purchase Row (Recurring)	$17,183.04	$0.00	$17,183.04
Used Row	$0.00	$51.14	$76.71

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.6.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$223,993.20	Upfront portion (50% of annual commitment)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	223,993.20	Upfront portion in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	447,986.40	Full annual commitment capacity ($51.14/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.6.5. Recurring Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$17,183.04	Monthly portion (hourly rate / 2 × 672 hours in Feb)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	17,183.04	Monthly portion in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	34,366.08	Full monthly commitment capacity ($51.14/hr × 672 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-RECURRING	Price point for recurring purchase

8.2.6.6. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$51.14	Amortized cost (annual / hours)
ListCost	$76.71	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	51.14	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.7. Aura Web Flexible Spend Plan - No Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	No Upfront
Commitment Discount Category	Spend
Utilization	100%
Hours Generated	24
Annual Commitment	$462,966.00
List Unit Price	$79.28/hour

CSV Example

8.2.7.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **No Upfront** payment option means you pay nothing at purchase time and instead pay a recurring monthly fee. This results in a recurring Purchase row each billing period with BilledCost equal to the monthly fee and zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.7.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$35,515.20	$0.00
Usage (Used)	24	$0.00	$1,268.40

Row Type	Count	BilledCost	EffectiveCost
Total	25	**$35,515.20**	**$1,268.40**

8.2.7.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.7.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	52.85 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $52.85/hour commitment, this value is $52.85.

8.2.7.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$79.28
ContractedUnitPrice	Negotiated unit price	$79.28

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.7.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$35,515.20	$0.00	$35,515.20
Used Row	$0.00	$52.85	$79.28

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.7.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$35,515.20	Monthly recurring payment (hourly rate × 672 hours in Feb)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows

Column	Value	Explanation
PricingQuantity	35,515.20	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	35,515.20	Commitment capacity for Feb ($52.85/hr × 672 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-MONTHLY	Price point for recurring purchase

8.2.7.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$52.85	Amortized cost (annual / hours)
ListCost	$79.28	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	52.85	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.8. Aura Web Flexible Spend Plan - All Upfront - 100% Utilization with Overage

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	100% (with overage to standard pricing)
Hours Generated	24 committed + 12 standard overflow
Annual Commitment	$211,992.00
List Unit Price	$36.30/hour

CSV Example

8.2.8.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of

`Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **100% utilization with overage** where demand exceeds commitment capacity. The 24 Used rows represent full utilization of the commitment. The 12 Standard rows represent compute usage beyond the commitment that spills to standard pricing. Standard pricing rows have no CommitmentDiscountStatus, PricingCategory='Standard', and BilledCost=EffectiveCost at the full list price.

8.2.8.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$211,992.00	$0.00
Usage (Used)	24	$0.00	$580.80
Usage (Standard)	12	$435.60	$435.60
Total	37	**$212,427.60**	**$1,016.40**

8.2.8.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.8.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	24.20 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $24.20/hour commitment, this value is $24.20.

8.2.8.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered	Standard
ListUnitPrice	List (public) unit price	$36.30	$36.30
ContractedUnitPrice	Negotiated unit price	$36.30	$36.30

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.8.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$211,992.00	$0.00	$211,992.00
Used Row	$0.00	$24.20	$36.30
Standard Row	$36.30	$36.30	$36.30

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.
- **Standard pricing rows:** `BilledCost` = `EffectiveCost` = `ListCost`. No commitment discount applies.

8.2.8.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$211,992.00	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	211,992.00	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	211,992.00	Full annual commitment ($24.20/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.8.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$24.20	Amortized cost (annual / hours)
ListCost	$36.30	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	24.20	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.8.6. Standard Pricing Usage Row Details

Column	Value	Explanation
ChargeCategory	Usage	Compute consumption (standard pricing)
PricingCategory	Standard	No discount applied
BilledCost	$36.30	Same as ListCost, no negotiation/commitments
EffectiveCost	$36.30	Same as BilledCost, no pre/post payments
ListCost	$36.30	Public, non-negotiated cost
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours consumed
CommitmentDiscountQuantity	null	**No commitment applied**
CommitmentDiscountStatus	null	No commitment
CommitmentDiscountId	null	No associated commitment
ContractedUnitPrice	$36.30	Equals ListUnitPrice (no negotiated discount)
SkuId	AURAWEB-USEAST1-COMPUTE-ONDEMAND	Standard (on-demand) resource SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-ONDEMAND-STANDARD	Price point for standard pricing

8.2.9. Aura Web Flexible Spend Plan - All Upfront - 75% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	75%
Hours Generated	24
Annual Commitment	$459,024.00
List Unit Price	$78.60/hour

CSV Example

8.2.9.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **underutilization** at 75% where only 18 of 24 commitment hours are consumed. The remaining 6 hours appear as 'Unused' rows with CommitmentDiscountStatus='Unused'. These unused rows still have EffectiveCost to reflect the wasted commitment value.

8.2.9.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$459,024.00	$0.00
Usage (Used)	18	$0.00	$943.20
Usage (Unused)	6	$0.00	$314.40

Row Type	Count	BilledCost	EffectiveCost
Total	25	**$459,024.00**	**$1,257.60**

8.2.9.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.9.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	52.40 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $52.40/hour commitment, this value is $52.40.

8.2.9.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$78.60
ContractedUnitPrice	Negotiated unit price	$78.60

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.9.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$459,024.00	$0.00	$459,024.00
Used Row	$0.00	$52.40	$78.60
Unused Row	$0.00	$52.40	$52.40

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.
- **Unused rows:** `BilledCost` = 0 but `EffectiveCost` > 0 to represent wasted commitment value.

8.2.9.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$459,024.00	Full annual commitment payment

Column	Value	Explanation
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	459,024.00	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	459,024.00	Full annual commitment ($52.40/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.9.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$52.40	Amortized cost (annual / hours)
ListCost	$78.60	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	52.40	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.9.6. Unused Commitment Row Details

Column	Value	Explanation
ChargeCategory	Usage	Represents commitment capacity
BilledCost	$0.00	No additional billing (already paid at purchase)
EffectiveCost	$52.40	**Wasted value** - lost commitment
PricingQuantity	52.40	Hourly commitment in USD (PricingUnit = USD)
ListCost	$52.40	$1.00 × 52.40 USD
ConsumedQuantity	null	**No resource consumed**
CommitmentDiscountQuantity	52.40	Commitment wasted
CommitmentDiscountStatus	Unused	Commitment not utilized

Column	Value	Explanation
ResourceId	auraweb:flexspend::123456789012:flexspendplan/fsp-abc123def456	must match CommitmentDiscountId (no resource used)
ResourceName	Compute Flexible Spend Plan	Carried from Purchase row (no resource consumed)
ResourceType	Commitment	Carried from Purchase row (no resource consumed)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	must match Purchase row (no resource consumed)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	must match Purchase row

For spend-based unused rows, PricingUnit is USD and PricingQuantity is the hourly commitment amount. ListCost = ListUnitPrice ($1.00) × PricingQuantity, which equals the wasted commitment dollars per hour.

8.2.10. Aura Web Flexible Spend Plan - All Upfront - 50% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	50%
Hours Generated	24
Annual Commitment	$693,003.60
List Unit Price	$118.67/hour

CSV Example

8.2.10.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **underutilization** at 50% where only 12 of 24 commitment hours are consumed. The remaining 12 hours appear as 'Unused' rows with CommitmentDiscountStatus='Unused'. These unused rows still have EffectiveCost to reflect the wasted commitment value.

8.2.10.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$693,003.60	$0.00
Usage (Used)	12	$0.00	$949.32
Usage (Unused)	12	$0.00	$949.32
Total	25	**$693,003.60**	**$1,898.64**

8.2.10.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.10.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	79.11 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $79.11/hour commitment, this value is $79.11.

8.2.10.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$118.67
ContractedUnitPrice	Negotiated unit price	$118.67

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.10.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$693,003.60	$0.00	$693,003.60
Used Row	$0.00	$79.11	$118.67
Unused Row	$0.00	$79.11	$79.11

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.
- **Unused rows:** `BilledCost` = 0 but `EffectiveCost` > 0 to represent wasted commitment value.

8.2.10.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$693,003.60	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	693,003.60	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	693,003.60	Full annual commitment ($79.11/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU

Column	Value	Explanation
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.10.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$79.11	Amortized cost (annual / hours)
ListCost	$118.67	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	79.11	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	auraweb:flexspend::123456789012:flexspendplan/fsp-...	Links usage to purchase
SkuId	AURAWEB-USEAST1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.10.6. Unused Commitment Row Details

Column	Value	Explanation
ChargeCategory	Usage	Represents commitment capacity
BilledCost	$0.00	No additional billing (already paid at purchase)
EffectiveCost	$79.11	**Wasted value** - lost commitment
PricingQuantity	79.11	Hourly commitment in USD (PricingUnit = USD)
ListCost	$79.11	$1.00 × 79.11 USD
ConsumedQuantity	null	**No resource consumed**
CommitmentDiscountQuantity	79.11	Commitment wasted
CommitmentDiscountStatus	Unused	Commitment not utilized
ResourceId	auraweb:flexspend::123456789012:flexspendplan/fsp-abc123def456	must match CommitmentDiscountId (no resource used)
ResourceName	Compute Flexible Spend Plan	Carried from Purchase row (no resource consumed)
ResourceType	Commitment	Carried from Purchase row (no resource consumed)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	must match Purchase row (no resource consumed)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	must match Purchase row

For spend-based unused rows, PricingUnit is USD and PricingQuantity is the hourly commitment amount. ListCost = ListUnitPrice ($1.00) × PricingQuantity, which equals the wasted commitment dollars per hour.

8.2.11. Aura Web Flexible Spend Plan - All Upfront - 0% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	0%
Hours Generated	24
Annual Commitment	$353,028.00
List Unit Price	$60.45/hour

CSV Example

8.2.11.1. Scenario Description

This example shows an **Aura Web Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **zero utilization** where the commitment is purchased but no resources are consumed. All usage rows have CommitmentDiscountStatus='Unused', representing wasted commitment capacity. The EffectiveCost on these rows reflects the cost of unused commitment that cannot be recovered.

8.2.11.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$353,028.00	$0.00
Usage (Unused)	24	$0.00	$967.20
Total	25	**$353,028.00**	**$967.20**

8.2.11.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.11.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	40.30 (USD, hourly rate)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	40.30 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For this commitment, the value equals the hourly dollar commitment.

8.2.11.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$1.00
ContractedUnitPrice	Negotiated unit price	$1.00

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices. For spend-based purchase and unused rows, PricingUnit is USD and ListUnitPrice is $1.00, because you are fundamentally purchasing a block of dollars.

8.2.11.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$353,028.00	$0.00	$353,028.00
Unused Row	$0.00	$40.30	$40.30

This scenario has no Used or Standard rows because utilization is 0% and no resources were consumed.

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Unused rows:** `BilledCost` = 0 but `EffectiveCost` > 0 to represent wasted commitment value.

8.2.11.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$353,028.00	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	353,028.00	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	353,028.00	Full annual commitment ($40.30/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.11.5. Unused Commitment Row Details

Column	Value	Explanation
ChargeCategory	Usage	Represents commitment capacity
BilledCost	$0.00	No additional billing (already paid at purchase)
EffectiveCost	$40.30	**Wasted value** - lost commitment

Column	Value	Explanation
PricingQuantity	40.30	Hourly commitment in USD (PricingUnit = USD)
ListCost	$40.30	$1.00 × 40.30 USD
ConsumedQuantity	null	**No resource consumed**
CommitmentDiscountQuantity	40.30	Commitment wasted
CommitmentDiscountStatus	Unused	Commitment not utilized
ResourceId	auraweb:flexspend::123456789012:flexspendplan/fsp-abc123def456	must match CommitmentDiscountId (no resource used)
ResourceName	Compute Flexible Spend Plan	Carried from Purchase row (no resource consumed)
ResourceType	Commitment	Carried from Purchase row (no resource consumed)
SkuId	AURAWEB-USEAST1-COMPUTE-PURCHASE	must match Purchase row (no resource consumed)
SkuPriceId	AURAWEB-USEAST1-COMPUTE-PURCHASE-UPFRONT	must match Purchase row (no resource consumed)

For spend-based unused rows, PricingUnit is USD and PricingQuantity is the hourly commitment amount. ListCost = ListUnitPrice ($1.00) × PricingQuantity, which equals the wasted commitment dollars per hour.

8.2.12. CrestNode Resource Reservation - All Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Usage
Utilization	100%
Hours Generated	24
Annual Commitment	$358,021.20
List Unit Price	$61.31/hour

CSV Example

8.2.12.1. Scenario Description

This example shows a **CrestNode Resource Reservation**, which is a commitment (with a Commitment Discount Category of `Usage`) where you commit to a specific quantity of resource capacity (e.g., instance hours).

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.12.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$358,021.20	$0.00
Usage (Used)	24	$0.00	$980.88
Total	25	**$358,021.20**	**$980.88**

8.2.12.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.12.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	1 (commitment units)

For usage-based commitments: CommitmentDiscountQuantity represents the quantity of resources (e.g., instance hours), not a dollar amount. For a 1-hour reservation, CommitmentDiscountQuantity = 1.

8.2.12.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$61.31
ContractedUnitPrice	Negotiated unit price	$61.31

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.12.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$358,021.20	$0.00	$358,021.20
Used Row	$0.00	$40.87	$61.31

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.12.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$358,021.20	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	8760.00	Total commitment capacity for the 1-year term (1 instance-hr/hr × 8,760 hrs/yr)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)

Column	Value	Explanation
SkuId	CRESTNODE-EASTUS-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.12.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$40.87	Amortized cost (annual / hours)
ListCost	$61.31	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	1	Commitment units applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	crestnode:compute:eastus:f0e9d8c7-b6a5-4321-0987-654321...	Links usage to purchase
SkuId	CRESTNODE-EASTUS-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.13. CrestNode Resource Reservation - No Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	No Upfront
Commitment Discount Category	Usage
Utilization	100%
Hours Generated	24
Annual Commitment	$664,008.00
List Unit Price	$113.70/hour

CSV Example

8.2.13.1. Scenario Description

This example shows a **CrestNode Resource Reservation**, which is a commitment (with a Commitment Discount Category of `Usage`) where you commit to a specific quantity of resource capacity (e.g., instance hours).

The **No Upfront** payment option means you pay nothing at purchase time and instead pay a recurring monthly fee. This results in a recurring Purchase row each billing period with BilledCost equal to the monthly fee and zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.13.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$55,334.00	$0.00
Usage (Used)	24	$0.00	$1,819.20
Total	25	**$55,334.00**	**$1,819.20**

8.2.13.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.13.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	1 (commitment units)

For usage-based commitments: CommitmentDiscountQuantity represents the quantity of resources (e.g., instance hours), not a dollar amount. For a 1-hour reservation, CommitmentDiscountQuantity = 1.

8.2.13.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$113.70
ContractedUnitPrice	Negotiated unit price	$113.70

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.13.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$55,334.00	$0.00	$55,334.00
Used Row	$0.00	$75.80	$113.70

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.13.4. Purchase Row Details

Column	Value	Explanation

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$55,334.00	Monthly recurring payment (annual / 12)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	672.00	Commitment capacity for Feb (1 instance-hr/hr × 672 hrs)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)
SkuId	CRESTNODE-EASTUS-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-PURCHASE-MONTHLY	Price point for recurring purchase

8.2.13.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$75.80	Amortized cost (annual / hours)
ListCost	$113.70	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	1	Commitment units applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	crestnode:compute:eastus:f0e9d8c7-b6a5-4321-0987-654321...	Links usage to purchase
SkuId	CRESTNODE-EASTUS-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.14. CrestNode Flexible Spend Plan - All Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	All Upfront
Commitment Discount Category	Spend
Utilization	100%
Hours Generated	24
Annual Commitment	$462,002.40
List Unit Price	$79.11/hour

CSV Example

8.2.14.1. Scenario Description

This example shows a **CrestNode Flexible Spend Plan**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **All Upfront** payment option means the entire commitment cost is paid at purchase time. This results in a single Purchase row with the full BilledCost and zero EffectiveCost (since the cost is amortized to usage rows).

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.14.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$462,002.40	$0.00
Usage (Used)	24	$0.00	$1,265.76
Total	25	**$462,002.40**	**$1,265.76**

8.2.14.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.14.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	52.74 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $52.74/hour commitment, this value is $52.74.

8.2.14.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$79.11
ContractedUnitPrice	Negotiated unit price	$79.11

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.14.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$462,002.40	$0.00	$462,002.40
Used Row	$0.00	$52.74	$79.11

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.14.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	One-Time	One-time upfront payment
BilledCost	$462,002.40	Full annual commitment payment
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	462,002.40	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	462,002.40	Full annual commitment ($52.74/hr × 8,760 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	CRESTNODE-EASTUS-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-PURCHASE-UPFRONT	Price point for upfront purchase

8.2.14.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$52.74	Amortized cost (annual / hours)
ListCost	$79.11	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	52.74	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	crestnode:compute:eastus:f0e9d8c7-b6a5-4321-0987-654321...	Links usage to purchase
SkuId	CRESTNODE-EASTUS-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	CRESTNODE-EASTUS-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.15. LatticeScale Resource Reservation - No Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	No Upfront
Commitment Discount Category	Usage
Utilization	100%

Parameter	Value
Hours Generated	24
Annual Commitment	$257,982.00
List Unit Price	$44.18/hour

CSV Example

8.2.15.1. Scenario Description

This example shows a **LatticeScale Resource Reservation**, which is a commitment (with a Commitment Discount Category of `Usage`) where you commit to a specific quantity of resource capacity (e.g., instance hours).

The **No Upfront** payment option means you pay nothing at purchase time and instead pay a recurring monthly fee. LatticeScale commitments are billed monthly with no upfront payment option. This results in a recurring Purchase row each billing period with BilledCost equal to the monthly fee and zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.15.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$19,790.40	$0.00
Usage (Used)	24	$0.00	$706.80
Total	25	**$19,790.40**	**$706.80**

8.2.15.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.15.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	1 (commitment units)

For usage-based commitments: CommitmentDiscountQuantity represents the quantity of resources (e.g., instance hours), not a dollar amount. For a 1-hour reservation, CommitmentDiscountQuantity = 1.

8.2.15.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$44.18
ContractedUnitPrice	Negotiated unit price	$44.18

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.15.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$19,790.40	$0.00	$19,790.40
Used Row	$0.00	$29.45	$44.18

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.15.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$19,790.40	Monthly fee (hourly rate × 672 hours in Feb)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	1	One commitment unit purchased
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	672.00	Commitment capacity for Feb (1 instance-hr/hr × 672 hrs)
CommitmentDiscountUnit	Hours	Unit of commitment capacity (usage-based)
SkuId	LATTICESCALE-USCENTRAL1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	LATTICESCALE-USCENTRAL1-COMPUTE-PURCHASE-MONTHLY	Price point for recurring purchase

8.2.15.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$29.45	Amortized cost (annual / hours)
ListCost	$44.18	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	1	Commitment units applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	latticescale:compute:us-central1:proj-123456:commitment-dis...	Links usage to purchase
SkuId	LATTICESCALE-USCENTRAL1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)

Column	Value	Explanation
SkuPriceId	LATTICESCALE-USCENTRAL1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.2.16. LatticeScale Dynamic Compute Commitment - No Upfront - 100% Utilization

Parameter	Value
Scenario Type	commitment
Payment Model	No Upfront
Commitment Discount Category	Spend
Utilization	100%
Hours Generated	24
Annual Commitment	$553,018.80
List Unit Price	$94.70/hour

CSV Example

8.2.16.1. Scenario Description

This example shows a **LatticeScale Dynamic Compute Commitment**, which is a commitment (with a Commitment Discount Category of `Spend`) where you commit to a specific dollar amount of usage per hour.

The **No Upfront** payment option means you pay nothing at purchase time and instead pay a recurring monthly fee. LatticeScale commitments are billed monthly with no upfront payment option. This results in a recurring Purchase row each billing period with BilledCost equal to the monthly fee and zero EffectiveCost.

This scenario demonstrates **full utilization** where exactly 100% of the commitment capacity is consumed. All usage rows have CommitmentDiscountStatus='Used', indicating the commitment was fully applied. BilledCost=0 on usage rows because they're covered by the commitment.

8.2.16.2. Row Summary

The following row summary reflects only the rows included in the 24-hour sample CSV.

Row Type	Count	BilledCost	EffectiveCost
Purchase	1	$42,423.36	$0.00
Usage (Used)	24	$0.00	$1,515.12
Total	25	**$42,423.36**	**$1,515.12**

8.2.16.3. Column Interactions

Understanding how columns relate to each other is critical for validating FOCUS data. This section explains the key relationships.

8.2.16.3.1. Quantity Columns: PricingQuantity vs. ConsumedQuantity vs. CommitmentDiscountQuantity

These three quantity columns serve different purposes and must be understood in context:

Column	Purpose	When Populated	Typical Value
PricingQuantity	Quantity used for pricing calculation	All priced rows	1 (per hour/unit)
ConsumedQuantity	Actual resource consumption	Usage rows with resources	1 (hours consumed)

Column	Purpose	When Populated	Typical Value
CommitmentDiscountQuantity	Commitment capacity applied	Rows with commitment discount	63.13 (USD)

For spend-based commitments: CommitmentDiscountQuantity represents the dollar amount applied, not a count of resources. For a $63.13/hour commitment, this value is $63.13.

8.2.16.3.2. Pricing Columns: ListUnitPrice vs. ContractedUnitPrice

Column	Purpose	Commitment-Covered
ListUnitPrice	List (public) unit price	$94.70
ContractedUnitPrice	Negotiated unit price	$94.70

Why this matters: ContractedUnitPrice reflects enterprise-negotiated pricing (e.g., enterprise-negotiated rates), not commitment discount savings. In non-negotiated scenarios, ContractedUnitPrice equals ListUnitPrice. Commitment discount savings are reflected in EffectiveCost, not in unit prices.

8.2.16.3.3. Cost Columns: BilledCost vs. EffectiveCost vs. ListCost

Scenario	BilledCost	EffectiveCost	ListCost
Purchase Row	$42,423.36	$0.00	$42,423.36
Used Row	$0.00	$63.13	$94.70

The following critical rules apply to commitment discount data:

- **Purchase rows:** `EffectiveCost` must be 0. The cost is distributed to usage rows.
- **Used rows:** `BilledCost` must be 0. Usage is covered by the commitment.

8.2.16.4. Purchase Row Details

Column	Value	Explanation
ChargeCategory	Purchase	Commitment purchase transaction
ChargeFrequency	Recurring	Monthly recurring fee
BilledCost	$42,423.36	Monthly fee (hourly rate × 672 hours in Feb)
EffectiveCost	$0.00	**must be 0** - cost is amortized to usage rows
PricingQuantity	42,423.36	Total commitment in USD (PricingUnit = USD)
CommitmentDiscountStatus	null	Status only applies to usage rows
CommitmentDiscountQuantity	42,423.36	Commitment capacity for Feb ($63.13/hr × 672 hrs)
CommitmentDiscountUnit	USD	Unit of commitment capacity (spend-based)
SkuId	LATTICESCALE-USCENTRAL1-COMPUTE-PURCHASE	Commitment purchase SKU
SkuPriceId	LATTICESCALE-USCENTRAL1-COMPUTE-PURCHASE-MONTHLY	Price point for recurring purchase

8.2.16.5. Usage Row Details (Commitment-Covered)

Column	Value	Explanation
ChargeCategory	Usage	Compute resource consumption
PricingCategory	Committed	Priced under commitment discount
BilledCost	$0.00	**must be 0** - covered by commitment
EffectiveCost	$63.13	Amortized cost (annual / hours)
ListCost	$94.70	What you would have paid at list price
PricingQuantity	1	Units priced
ConsumedQuantity	1	Hours used
CommitmentDiscountQuantity	63.13	Hourly commitment spend applied
CommitmentDiscountStatus	Used	Commitment applied
CommitmentDiscountId	latticescale:compute:us-central1:proj-123456:commitment-dis...	Links usage to purchase
SkuId	LATTICESCALE-USCENTRAL1-COMPUTE-USAGE	Resource usage SKU (differs from Purchase)
SkuPriceId	LATTICESCALE-USCENTRAL1-COMPUTE-USAGE-COMMITTED	Price point for committed usage

8.3. Examples: Commitment Discount Flexibility

A usage-based *commitment discount* obligates a customer to a usage amount for one or more related SKUs in return for reduced rates. For example, when a usage-based *commitment discount* is purchased to cover a specific database SKU, this commitment will cover every hour over the period where at least one instance of this SKU is running. The usage-based commitment can cover 1 resource over the hour, or in the case of *commitment discount flexibility*, it can cover a portion of 1 resource or multiple resources at a time.

When mixing usage-based commitment discounts with and without *commitment discount flexibility* and *CommitmentDiscountQuantity* measured by time, it is important to differentiate the *CommitmentDiscountUnit* for each type of *commitment discount*. In each scenario below, *commitment discounts without commitment discount flexibility* applied use "Hour" as the *CommitmentDiscountUnit*, and conversely commitment discounts *with commitment discount flexibility* applied use "Normalized Hour" as the *CommitmentDiscountUnit*.

For more details on exactly how *commitment discounts* purchase and usage rows appear with and without *commitment discount flexibility*, see the following scenarios:

8.3.1. 100% Utilization Without Commitment Discount Flexibility

8.3.1.1. Context

For this example, fictitious service provider, *LatticeScale*, offers the following SKU catalog which is used in the scenario below.

8.3.1.2. SKU Catalog

Service	Sku Id	Sku Price Id	Sku Price Unit Price	Normalization Factor
Compute	VM_SMALL	VM_SMALL_COMMITTED_PURCHASE_NO_UPFRONT	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_PURCHASE_NO_UPFRONT	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_PURCHASE_NO_UPFRONT	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_PURCHASE_NO_UPFRONT	$2.00	4
Compute	VM_SMALL	VM_SMALL_COMMITTED_HOUR	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_HOUR	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_HOUR	$1.50	3

Service	Sku Id	Sku Price Id	Sku Price Unit Price	Normalization Factor
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_HOUR	$2.00	4
Compute	VM_SMALL	VM_SMALL_ON_DEMAND_HOUR	$1.00	1
Compute	VM_MEDIUM	VM_MEDIUM_ON_DEMAND_HOUR	$2.00	2
Compute	VM_LARGE	VM_LARGE_ON_DEMAND_HOUR	$3.00	3
Compute	VM_XLARGE	VM_XLARGE_ON_DEMAND_HOUR	$4.00	4

The above SKU Catalog shows that this service provider only has 1 service that offers 4 virtual machine SKUs at various list rates, *commitment discount* rates, and normalization factors. Each SKU's normalization factor classifies its relative size to its *commitment discount* rate. Usage-based *commitment discounts* with *commitment discount flexibility* can fully cover any combination of 1 or more SKUs where the sum of their normalization factor is less than or equal to the normalization factor of the *commitment discount*.

8.3.1.3. Scenario

- 1 no upfront *commitment discount* is purchased for 1 year (2023) for 1 VM_LARGE.
- 1 VM_LARGE resource runs for 1 hour from 2023-01-01T00:00:00 to 2023-01-01T01:00:00.

8.3.1.4. Outcome

- 1 recurring, purchase record exists for 1 eligible "Hour" of the no upfront, *commitment discount* and incurs a $1.50 *BilledCost*.
- The *commitment discount* covers the first *charge period* for 1 VM_LARGE resource incurring a $1.50 *EffectiveCost*.

CSV Example

8.3.2. 0% Utilization Without Commitment Discount Flexibility

8.3.2.1. Context

For this example, fictitious service provider, *LatticeScale*, offers the following SKU catalog which is used in the scenario below.

8.3.2.2. SKU Catalog

Service	Sku Id	Sku Price Id	Sku Price Unit Price	Normalization Factor
Compute	VM_SMALL	VM_SMALL_COMMITTED_PURCHASE_NO_UPFRONT	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_PURCHASE_NO_UPFRONT	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_PURCHASE_NO_UPFRONT	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_PURCHASE_NO_UPFRONT	$2.00	4
Compute	VM_SMALL	VM_SMALL_COMMITTED_HOUR	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_HOUR	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_HOUR	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_HOUR	$2.00	4
Compute	VM_SMALL	VM_SMALL_ON_DEMAND_HOUR	$1.00	1
Compute	VM_MEDIUM	VM_MEDIUM_ON_DEMAND_HOUR	$2.00	2
Compute	VM_LARGE	VM_LARGE_ON_DEMAND_HOUR	$3.00	3
Compute	VM_XLARGE	VM_XLARGE_ON_DEMAND_HOUR	$4.00	4

The above SKU Catalog shows that this service provider only has 1 service that offers 4 virtual machine SKUs at various list unit prices, *commitment discount* unit prices, and normalization factors. Each SKU's normalization factor classifies its relative size to its *commitment discount* unit price. Usage-based *commitment discounts* with *commitment discount flexibility* can fully cover any combination of 1 or more SKUs where the sum of their normalization factor is less than or equal to the normalization

factor of the *commitment discount*.

8.3.2.3. Scenario

- 1 no upfront *commitment discount* is purchased for 1 year (2023) for 1 VM_LARGE.
- 1 VM_MEDIUM resource runs for 1 hour from 2023-01-01T00:00:00 to 2023-01-01T01:00:00.

8.3.2.4. Outcome

- 1 recurring, purchase record exists for 1 eligible "Hour" of the no upfront, *commitment discount* and incurs a $1.50 *BilledCost*.
- The VM_LARGE *commitment discount* is unused for the corresponding *charge period* because no VM_LARGE resources are running and incurs a $1.50 *EffectiveCost*.
- 1 hour of on-demand usage is incurred by the VM_MEDIUM resource and incurs a $2.00 *BilledCost* and *EffectiveCost*.

CSV Example

8.3.3. 100% Utilization with Commitment Discount Flexibility with 1 Resource

8.3.3.1. Context

For this example, fictitious service provider, *LatticeScale*, offers the following SKU catalog which is used in the scenario below.

8.3.3.2. SKU Catalog

Service	Sku Id	Sku Price Id	Sku Price Unit Price	Normalization Factor
Compute	VM_SMALL	VM_SMALL_COMMITTED_PURCHASE_NO_UPFRONT	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_PURCHASE_NO_UPFRONT	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_PURCHASE_NO_UPFRONT	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_PURCHASE_NO_UPFRONT	$2.00	4
Compute	VM_SMALL	VM_SMALL_COMMITTED_HOUR	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_HOUR	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_HOUR	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_HOUR	$2.00	4
Compute	VM_SMALL	VM_SMALL_ON_DEMAND_HOUR	$1.00	1
Compute	VM_MEDIUM	VM_MEDIUM_ON_DEMAND_HOUR	$2.00	2
Compute	VM_LARGE	VM_LARGE_ON_DEMAND_HOUR	$3.00	3
Compute	VM_XLARGE	VM_XLARGE_ON_DEMAND_HOUR	$4.00	4

The above SKU catalog shows that this service provider only has 1 service that offers 4 virtual machine SKUs at various list rates, *commitment discount* rates, and normalization factors. Each SKU's normalization factor classifies its relative size to its *commitment discount* rate. Usage-based *commitment discounts* with *commitment discount flexibility* can fully cover any combination of 1 or more SKUs where the sum of their normalization factor is less than or equal to the normalization factor of the *commitment discount*.

8.3.3.3. Scenario

- 1 no upfront *commitment discount* is purchased for 1 year (2023) for 1 VM_SMALL which has a normalization factor of 1.
- 1 VM_LARGE resource runs for 1 hour from 2023-01-01T00:00:00 to 2023-01-01T01:00:00 with a normalization factor of 4.

8.3.3.4. Outcome

- 1 recurring, purchase record exists for 1 eligible "Normalized Hour" of the no upfront, *commitment discount* and incurs a $0.50 *BilledCost*.
- The VM_SMALL *commitment discount* is fully utilized within the corresponding *charge period*, covers 25% of the VM_LARGE resource, and incurs a $0.50 *EffectiveCost*.
- The VM_LARGE resource incurs an additional, on-demand $2.25 *BilledCost* and *EffectiveCost*.

CSV Example

8.3.4. 100% Utilization with Commitment Discount Flexibility with 2 Resources

8.3.4.1. Context

For this example, fictitious service provider, *LatticeScale*, offers the following SKU catalog which is used in the scenario below.

8.3.4.2. SKU Catalog

Service	Sku Id	Sku Price Id	Sku Price Unit Price	Normalization Factor
Compute	VM_SMALL	VM_SMALL_COMMITTED_PURCHASE_NO_UPFRONT	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_PURCHASE_NO_UPFRONT	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_PURCHASE_NO_UPFRONT	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_PURCHASE_NO_UPFRONT	$2.00	4
Compute	VM_SMALL	VM_SMALL_COMMITTED_HOUR	$0.50	1
Compute	VM_MEDIUM	VM_MEDIUM_COMMITTED_HOUR	$1.00	2
Compute	VM_LARGE	VM_LARGE_COMMITTED_HOUR	$1.50	3
Compute	VM_XLARGE	VM_XLARGE_COMMITTED_HOUR	$2.00	4
Compute	VM_SMALL	VM_SMALL_ON_DEMAND_HOUR	$1.00	1
Compute	VM_MEDIUM	VM_MEDIUM_ON_DEMAND_HOUR	$2.00	2
Compute	VM_LARGE	VM_LARGE_ON_DEMAND_HOUR	$3.00	3
Compute	VM_XLARGE	VM_XLARGE_ON_DEMAND_HOUR	$4.00	4

The above SKU Catalog shows that this service provider only has 1 service that offers 4 virtual machine SKUs at various list rates, *commitment discount* rates, and normalization factors. Each SKU's normalization factor classifies its relative size to its *commitment discount* rate. Usage-based *commitment discounts* with *commitment discount flexibility* can fully cover any combination of 1 or more SKUs where the sum of their normalization factor is less than or equal to the normalization factor of the *commitment discount*.

8.3.4.3. Scenario

- 1 no upfront *commitment discount* is purchased for 1 year (2023) for 1 VM_XLARGE which has a normalization factor of 8.
- 2 VM_MEDIUM resources run for 1 hour from 2023-01-01T00:00:00 to 2023-01-01T01:00:00 with a normalization factor of 4 for each.

8.3.4.4. Outcome

- 1 recurring, purchase record exists for 1 eligible "Normalized Hour" for a no upfront, *commitment discount* and incurs a $2.00 *BilledCost*.
- With *commitment discount flexibility*, 1 *commitment discount* for a VM_XLARGE covers 2 VM_MEDIUM resources within the corresponding *charge period* and incurs a $2.00 total *EffectiveCost*.
    - 1 *commitment discount* with a normalization factor of 8 covers 2 resources with normalization factors of 4 (i.e 4 + 4 = 8).

CSV Example

8.4. Examples: Commitment Program Eligibility Details

> Note: The following examples are informative and non-normative. They do not define requirements.

This section demonstrates how CommitmentProgramEligibilityDetails interacts with other columns for *capacity reservation* programs. For discount-bearing program SQL queries, see the Commitment Program Eligibility Details supported feature.

8.4.1. Capacity Reservation Eligible Spend

This example demonstrates how CommitmentProgramEligibilityDetails interacts with CapacityReservationId and CapacityReservationStatus for *capacity reservation* programs. Unlike discount-bearing *commitment programs*, capacity reservations secure resource availability and are tracked via their own columns rather than the *commitment discount* columns.

Acme Corp runs compute workloads on Aura Web and holds an Advance Resource Commitment (cr-arc-acme-001) for a single charge period (2025-04-01). The `ProgramType` values "Advance Resource Commitment" and "Zonal Resource Commitment" are illustrative and do not correspond to a specific provider's program names.

This example focuses on Usage rows. Purchase rows for the reservation itself are not shown.

Four usage rows for the period:

1. **Used capacity** (Row 1): Compute usage consuming the reservation. CapacityReservationStatus is "Used". BilledCost and EffectiveCost are both $180.00.
2. **Unused capacity** (Row 2): Reserved capacity that went idle. CapacityReservationStatus is "Unused". BilledCost and EffectiveCost are both $70.00. Unlike *commitment discount* unused rows (where BilledCost is $0.00 because the purchase is invoiced separately), capacity reservation rows reflect the cost of reserved capacity whether consumed or not.
3. **Eligible but unreserved** (Row 3): Compute usage eligible for a Zonal Resource Commitment but no reservation is active. CapacityReservationId and CapacityReservationStatus are null. BilledCost and EffectiveCost are both $120.00 at standard pricing.
4. **Ineligible** (Row 4): A support fee with no *commitment program* eligibility. CommitmentProgramEligibilityDetails is null.

CommitmentProgramEligibilityDetails is populated on both Used and Unused rows (Rows 1 and 2). The column requirement states that CommitmentProgramEligibilityDetails "MUST NOT be null when a charge is eligible for a commitment program, regardless of whether a commitment was actually applied to the charge." The underlying resource type remains eligible for the program regardless of utilization status.

The capacity reservation query filters on CommitmentProgramEligibilityDetails and specific `ProgramType` values. Row 4 is excluded because CommitmentProgramEligibilityDetails is null. Rows 1 through 3 appear in the output, grouped by `ProgramType` and CapacityReservationStatus, allowing practitioners to see used, unused, and unreserved eligible spend separately.

8.5. Examples: Contract Commitments

8.5.1. Overview

The **Contract Commitment** dataset provides a structured representation of the commercial agreements between a customer and their service providers. While the Cost and Usage dataset tracks the results of consumption, the Contract Commitment dataset tracks the intent and constraints of the relationship.

8.5.1.1. Core Logical Pillars

To ensure interoperability across different data generators, the dataset relies on three core logical pillars:

1. **Commitment Categorization:** Distinguishes between obligations based on **Spend** (e.g., "I will spend 1M") vs. **Usage** (e.g., "I will use 500 vCPUs"). This determines which metrics —- Cost or Quantity —- are used to measure fulfillment.
2. **Fulfillment Modeling:** Defines the operational behavior and consumption flexibility of a commitment.
    - **Continuous** models (like Resource Reservations) are "use-it-or-lose-it", typically within short windows (e.g., `Hourly`).
    - **Discontinuous** models (like Enterprise Spend Agreements) allow consumption to be aggregated over a longer duration (e.g., `Full Period`).
3. **Eligibility Boundaries:** Using a structured JSON format, the dataset defines the logical perimeter of a commitment,

specifying exactly which accounts, regions, or services are eligible to receive the negotiated benefit.

8.5.1.2. Expected Value Taxonomy

The following table defines the high-level expectations for key categorical columns in this dataset:

Attribute	Expected Value Logic	Example Values
Benefit Category	The primary economic advantage provided.	`Discount`, `Entitlement`, `Availability`, `Other`
Model	How the commitment is consumed.	`Continuous`, `Discontinuous`
Fulfillment Interval	The "Use-it-or-lose-it" or "Goal" window for reset.	`Hourly`, `Monthly`, `Annual`, `Full Period`
Lifecycle Status	The current lifecycle state of the record.	`Active`, `Exhausted`, `Pending`, `Expired`, `Canceled`
Offer Category	The "privacy" or source level of the pricing.	`Public`, `Negotiated`
Payment Model	The cash-flow timing for the commitment.	`No Upfront`, `Partial Upfront`, `All Upfront`

8.5.1.3. Why the Values Matter

By standardizing these values, organizations can move from manual spreadsheet tracking to **Automated FinOps Governance**.

For example, in a Discontinuous model with an Annual Fulfillment Interval, a reporting engine does not evaluate the commitment hour-by-hour. Instead, it accumulates usage across the full year and waits until the end of that annual interval to determine whether the contractual threshold has been met. At that point, it looks for a "True-up" event — the reconciliation step that settles any difference between the committed amount and actual consumption.

By contrast, in a Continuous model with an Hourly Fulfillment Interval, an engine evaluates the commitment during every hour of the billing period. For each individual hour, it compares committed capacity to actual usage. If usage falls short during a given interval, the engine calculates "Waste", representing the unused portion of the commitment for that specific hour. That unused capacity is measured and reported as it occurs rather than deferred for end-of-term reconciliation.

8.5.2. Examples

8.5.2.1. Common Offering Examples

The following table provides a reference for how common cloud and SaaS commercial offerings theoretically map to the Contract Commitment schema. (This table is provided for demonstration purposes only. Actual value assignments are left to the data generators upon the creation of dataset artifacts.)

In the below table, CC represents Contract Commitment.

Offering Example	CC Category	CC Model	CC Offer Category	CC Benefit Category	CC Fulfillment Interval	CC Duration Type	CC Payment Model	CC Payment Interval
Flexible Spend Plan	Spend	Continuous	Public	Discount	Hourly	1 Year	Partial Upfront	Monthly
Resource Reservation	Usage	Continuous	Public	Discount	Hourly	3 Years	All Upfront	One-Time
Dynamic Compute Commitment	Spend	Continuous	Public	Discount	Hourly	3 Years	No Upfront	Monthly
Advance Resource Commitment	Usage	Continuous	Public	Availability	Hourly	1 Month	No Upfront	Monthly
Enterprise Spend Agreement	Spend	Discontinuous	Negotiated	Discount	Full Period	3 Years	No Upfront	Monthly

Offering Example	CC Category	CC Model	CC Offer Category	CC Benefit Category	CC Fulfillment Interval	CC Duration Type	CC Payment Model	CC Payment Interval
Bulk Capacity Credits	Spend	Discontinuous	Negotiated	Entitlement	Full Period	1 Year	All Upfront	One-Time
Interval Spend Commitment	Spend	Discontinuous	Public	Entitlement	Full Period	1 Year	No Upfront	Monthly
Seat License	Usage	Continuous	Negotiated	Discount	Monthly	1 Year	No Upfront	Monthly
Multi-Year Pool	Spend	Discontinuous	Negotiated	Discount	Full Period	3 Years	All Upfront	One-Time
Growth Rebate	Spend	Discontinuous	Negotiated	Discount	Annual	2 Years	No Upfront	Annual
API Credit Pack	Usage	Discontinuous	Public	Entitlement	Transactional	2 Years	All Upfront	One-Time
Marketplace SaaS	Spend	Discontinuous	Negotiated	Entitlement	Annual	1 Year	All Upfront	One-Time
90-Day POC	Spend	Discontinuous	Negotiated	Entitlement	Custom	90 Days	No Upfront	Custom
Enterprise Support	Usage	Continuous	Negotiated	Other	Monthly	1 Year	No Upfront	Monthly

8.5.2.2. Scenario 1: Strategic Cloud Transformation Agreement

This example demonstrates a complex, multi-faceted agreement between a customer and a primary cloud provider, **Aura Web**. While governed by a single master contract (`AGR-99-BETA`), it contains three distinct commercial levers:

8.5.2.2.1. Commitment 1: The Global Spend Pool

- **Context:** A high-level Enterprise Agreement (EA) where the customer commits to spending **1M USD** over three years.
- **Commercial Logic:** A **Spend-based**, **Discontinuous** model. Every dollar spent within the three-year window fulfills the commitment.
- **Eligibility: Global**. Applies to any service or region.

8.5.2.2.2. Commitment 2: Regional Compute Reservations

- **Context:** Fixed capacity of virtual machines in a specific data center for stable production workloads.
- **Commercial Logic:** A **Usage-based**, **Continuous** model. This benefit is "use it or lose it" on an **Hourly** basis.
- **Eligibility:** Restricted to specific resource types running in the `us-east-1` region.

8.5.2.2.3. Commitment 3: Marketplace SaaS Add-On

- **Context:** A specialized analytics tool, **OmniQuery**, purchased through the Aura Web marketplace.
- **Commercial Logic:** A **Spend-based** "pass-through" for financial tracking with an **Annual Fulfillment Interval**.
- **The Issuer/Provider Split:** The **Service Provider** is **OmniQuery**, but the **Invoice Issuer** remains **Aura Web**, showing how the model tracks third-party spend in a unified ecosystem.

8.5.2.3. Data Example: AGR-99-BETA

Column	Commitment 1: Spend Pool	Commitment 2: Compute Reservation	Commitment 3: Marketplace SaaS

Column	Commitment 1: Spend Pool	Commitment 2: Compute Reservation	Commitment 3: Marketplace SaaS
Billing Currency	EUR	EUR	EUR
CC Benefit Category	Discount	Discount	Entitlement
CC Category	Spend	Usage	Spend
CC Cost	925000.00	46250.00	111000.00
CC Created	2025-12-01T09:00:00Z	2025-12-01T09:00:00Z	2026-01-15T14:30:00Z
CC Description	3yr Enterprise Spend Goal	us-east-1 m5 Resource Reservations	OmniQuery Pro via Marketplace
CC Discount %	0.15	0.40	0.10
CC Duration Type	3 Years	1 Year	1 Year
CC Applicability	{"IsGlobalScope": true}	{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "RegionId", "Operator": "In", "Values": ["us-east-1"]}]}	{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "ServiceCategory", "Operator": "In", "Values": ["Analytics"]}]}
CC Fulfillment Interval	Full Period	Hourly	Annual
CC ID	CMT-SPEND-001	CMT-RR-002	CMT-SaaS-003
CC Last Updated	2026-02-01T10:00:00Z	2025-12-01T09:00:00Z	2026-01-15T14:30:00Z
CC Model	Discontinuous	Continuous	Discontinuous
CC Offer Category	Negotiated	Public	Negotiated
CC Payment Interval	Monthly	One-Time	Monthly
CC Payment Model	No Upfront	All Upfront	Partial Upfront
CC Payment Upfront %	0.00	1.00	0.50
CC Period End	2028-12-01	2026-12-01	2027-01-15
CC Period Start	2025-12-01	2025-12-01	2026-01-15
CC Quantity	1000000.00	10.00	120000.00
CC Lifecycle Status	Active	Active	Pending
CC Type	Enterprise Spend Agreement	Resource Reservation	SaaS Subscription
CC Unit	USD	Instance-Hours	Credits
Contract ID	AGR-99-BETA	AGR-99-BETA	AGR-99-BETA
Contract Period End	2028-12-01	2028-12-01	2028-12-01
Contract Period Start	2025-12-01	2025-12-01	2025-12-01
Invoice Issuer Name	Aura Web	Aura Web	Aura Web
Pricing Currency	USD	USD	USD
Pricing Currency CC Cost	1000000.00	50000.00	120000.00
Service Provider Name	Aura Web	Aura Web	OmniQuery

CSV Example

8.5.2.4. Scenario 2: SaaS Expansion & Hybrid Connector

In this scenario, an enterprise with an existing master agreement with **Aura Web** (`AGR-44-GAMMA`) expands its footprint to include specialized AI training and security licensing. This example highlights how the model handles non-financial units (Seats) and project-based burst windows.

8.5.2.4.1. Commitment 1: AI Model Training (Usage-Based Burst)

- **Context:** A short-term, intensive commitment to a specific number of GPU-Hours for a specialized AI training run.
- **Commercial Logic:** A **Usage-based**, **Continuous** model with a short **3-Month** duration. It is paid **All Upfront** to secure priority capacity.
- **Eligibility:** Restricted to the `AI/ML` service category.

8.5.2.4.2. Commitment 2: Observability Seat License (Quantity-Based)

- **Context:** A commitment to **500 Seats** of an observability and monitoring platform.
- **Commercial Logic:** A **Quantity-based**, **Discontinuous** model. The unit of measure is **Seats** rather than a currency value.
- **Invoice/Provider Alignment:** Unlike the Marketplace example, this is billed directly by the vendor (**StackLens**), yet remains logically associated with the broader Cloud Transformation contract.

8.5.2.4.3. Commitment 3: Cross-Cloud Data Connector (Tiered Usage)

- **Context:** A commitment based on **Data Volume (TB)** specifically for egress traffic between cloud providers.
- **Commercial Logic:** A **Usage-based**, **Continuous** model tracked on a **Monthly Fulfillment Interval**.
- **Eligibility:** Targeted specifically at `Egress` usage types via string-match logic in the Eligibility JSON.

8.5.2.5. Data Example: AGR-44-GAMMA

Column	Commitment 1: AI Training	Commitment 2: Observability Seats	Commitment 3: Data Connector
Billing Currency	`USD`	`USD`	`USD`
CC Benefit Category	`Discount`	`Entitlement`	`Discount`
CC Category	`Usage`	`Usage`	`Usage`
CC Cost	`250000.00`	`120000.00`	`15000.00`
CC Created	`2026-02-01T08:00:00Z`	`2026-02-01T08:00:00Z`	`2026-02-01T08:00:00Z`
CC Description	`H100 GPU Reservation - Q1`	`StackLens Monitoring Seats`	`Inter-Cloud Egress Tier`
CC Discount %	`0.30`	`null`	`0.50`
CC Duration Type	`3 Months`	`1 Year`	`1 Year`
CC Applicability	`{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "ServiceCategory", "Operator": "In", "Values": ["AI/ML"]}]}`	`{"IsGlobalScope": true}`	`{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "UsageType", "Operator": "Contains", "Values": ["Egress"]}]}`

Column	Commitment 1: AI Training	Commitment 2: Observability Seats	Commitment 3: Data Connector
CC Fulfillment Interval	Monthly	Annual	Monthly
CC ID	CMT-AI-888	CMT-SEC-999	CMT-DATA-111
CC Last Updated	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z
CC Model	Continuous	Discontinuous	Continuous
CC Offer Category	Negotiated	Negotiated	Public
CC Payment Interval	One-Time	Monthly	Monthly
CC Payment Model	All Upfront	No Upfront	No Upfront
CC Payment Upfront %	1.00	0.00	0.00
CC Period End	2026-05-01	2027-02-01	2027-02-01
CC Period Start	2026-02-01	2026-02-01	2026-02-01
CC Quantity	5000.00	500.00	100.00
CC Lifecycle Status	Active	Active	Active
CC Type	Resource Reservation	SaaS Subscription	Usage Tier
CC Unit	GPU-Hours	Seats	Terabytes
Contract ID	AGR-44-GAMMA	AGR-44-GAMMA	AGR-44-GAMMA
Contract Period End	2029-02-01	2029-02-01	2029-02-01
Contract Period Start	2026-02-01	2026-02-01	2026-02-01
Invoice Issuer Name	Aura Web	StackLens	Aura Web
Pricing Currency	USD	USD	USD
Pricing Currency CC Cost	250000.00	120000.00	15000.00
Service Provider Name	Aura Web	StackLens	Aura Web

CSV Example

8.5.2.6. Scenario 3: Scale-Out & Overage

This scenario focuses on how the model handles growth beyond initial estimates. In the master agreement `AGR-11-DELTA`, the customer has established "safety nets" and tiered pricing to ensure that scale-out events are still covered by negotiated rates, even after a primary pool is exhausted.

8.5.2.6.1. Commitment 1 & 2: Database Storage Tiers (Base + Overage)

- **Context:** The customer commits to a base of 100TB of Database storage. To avoid "sticker shock" if they grow to

150TB, they have a pre-negotiated **Overage Tier**.

- **Commercial Logic:** Commitment 1 is the paid floor (`CC Cost: 50000.00`). Commitment 2 is a "Zero-Cost" commitment that exists solely to define the **CC Discount %** (10%) applied to any usage exceeding the first 100TB.
- **Eligibility:** Both rows target the same `Database` service category.

8.5.2.6.2. Commitment 3: CDN Annual (Volume Exhaustion)

- **Context:** An annual volume commitment of 1PB (1,000TB) for Content Delivery Network services.
- **Commercial Logic:** This is a **Discontinuous** model with an **Annual Fulfillment Interval**.
- **Overage Status:** Because the customer has already consumed their allotted volume before the `CC Period End`, the **CC Lifecycle Status** has shifted to `Exhausted`. This signals that the pool is empty and subsequent usage will be handled according to the contract's true-up or on-demand terms.

8.5.2.7. Data Example: AGR-11-DELTA

Column	Commitment 1: Base Storage	Commitment 2: Storage Overage	Commitment 3: CDN Annual
Billing Currency	USD	USD	USD
CC Benefit Category	Discount	Discount	Discount
CC Category	Usage	Usage	Usage
CC Cost	50000.00	0.00	100000.00
CC Created	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z
CC Description	Base 100TB DB Storage	Tier 2 Storage Overage	1PB Annual CDN Volume
CC Discount %	0.20	0.10	0.25
CC Duration Type	1 Year	1 Year	1 Year
CC Applicability	{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "ServiceCategory", "Operator": "In", "Values": ["Database"]}]}	{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "ServiceCategory", "Operator": "In", "Values": ["Database"]}]}	{"InclusionOperator": "Or", "Inclusions": [{"Dimension": "ServiceCategory", "Operator": "In", "Values": ["CDN"]}]}
CC Fulfillment Interval	Monthly	Monthly	Annual
CC ID	CMT-STR-BASE	CMT-STR-OVER	CMT-CDN-VOL
CC Last Updated	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z	2026-02-01T08:00:00Z
CC Model	Continuous	Continuous	Discontinuous
CC Offer Category	Negotiated	Negotiated	Negotiated
CC Payment Interval	Monthly	Monthly	One-Time
CC Payment Model	No Upfront	No Upfront	All Upfront
CC Payment Upfront %	0.00	0.00	1.00
CC Period End	2027-02-01	2027-02-01	2027-02-01
CC Period Start	2026-02-01	2026-02-01	2026-02-01
CC Quantity	100.00	0.00	1000.00
CC Lifecycle Status	Active	Active	Exhausted
CC Type	Usage Tier	Usage Tier	Volume Commitment

Column	Commitment 1: Base Storage	Commitment 2: Storage Overage	Commitment 3: CDN Annual
CC Unit	Terabytes	Terabytes	Terabytes
Contract ID	AGR-11-DELTA	AGR-11-DELTA	AGR-11-DELTA
Contract Period End	2029-02-01	2029-02-01	2029-02-01
Contract Period Start	2026-02-01	2026-02-01	2026-02-01
Invoice Issuer Name	Aura Web	Aura Web	Aura Web
Pricing Currency	USD	USD	USD
Pricing Currency CC Cost	50000.00	0.00	100000.00
Service Provider Name	Aura Web	Aura Web	Aura Web

CSV Example

8.6. Examples: Correction Handling

This section provides examples of how correction handling may be implemented in alignment with the FOCUS specification. The examples are limited to illustrating correction handling for the Cost and Usage FOCUS dataset, and cover scenarios involving corrections to open and closed billing periods, as well as the various delivery mechanisms and correction styles supported by FOCUS.

The examples that follow are sectioned by Billing Period Status (i.e., "Open" or "Closed") and Billing Period Category (i.e., current open period, or previous open period, or closed):

Billing Period Status	Billing Period Category	Description
Open	Current Open	Examples of corrections to the current, open billing period.
Open	Previous Open	Examples of corrections to a previous billing period that has not yet been closed.
Closed	Closed	Examples of corrections to a closed billing period.

Within each of the sections above, the following four correction scenarios are demonstrated:

Correction Scenario	Description
Partial Reallocation to Correct Resource	Correcting misattributed costs between resources.
Late-Arriving Usage	Accounting for omitted costs and usage incurred in a prior timeframe.
Itemized Cost-Only Corrections	Reconciling minor cost drift with explicit adjustments per SKU.
Bulk Cost-Only Corrections	Reconciling minor cost drift using a single, consolidated adjustment.

The example dataset artifacts provided in these scenarios demonstrate three distinct correction styles, each aligned with one of the supported delivery mechanisms:

Correction Style	Delivery Mechanism	Description
Replacement	Overwrite	Corrections are modeled through updates, additions, or omissions relative to the previous snapshot. This style reflects the latest state of data and does not retain historical correction records unless externally preserved. Net data volume is the lowest of the three correction styles as each dataset artifact supersedes previously delivered ones for the same delivery scope.
Delta	Append	Corrections are represented by additive records that increment or decrement selected cost and quantity values. Original records remain unchanged, and reversals are optional. This style offers limited audit transparency. Net data volume is typically higher than Replacement but lower than Ledger as all delivered dataset artifacts are preserved.

Correction Style	Delivery Mechanism	Description
Ledger	Append	Corrections typically follow a two-step approach: reversal of the original record (if needed), and addition of a corrected record. Reversal is performed by creating a record with opposite cost and quantity values, while all other columns match the original. This style preserves full correction history and supports comprehensive auditability. Net data volume is typically the highest of the three correction styles as each correction requires explicit reversal and re-entry records.

8.6.1. Corrections to Open Billing Period

The following examples illustrate how corrections to open billing periods, including both current and previous open periods, may be represented in FOCUS Cost and Usage datasets, using various delivery mechanisms and correction styles.

Note: Corrections in this section apply to billing periods that are still open, whether current (Current Open-Period Correction Scenarios) or previous (Previous Open-Period Correction Scenarios). In the examples that follow, the InvoiceId column contains a provisional or placeholder value, even though the invoice has not yet been issued.

8.6.1.1. Current Open-Period Correction Scenarios

These scenarios address discrepancies related to charges from the July 2025 billing period. The discrepancies were identified on July 5th, 2025, while the July 2025 billing period was still open.

8.6.1.1.1. Billing Period Alignment

Since the billing period is still open and invoices have not yet been issued, corrections can be applied without introducing retroactive semantics. The ChargeClass is set to null, and the BillingPeriodStart/End reflect the billing period in which the cost or usage was actually incurred (July).

8.6.1.1.2. Correction Style

CrestNode delivers corrections using Replacement, Delta, and Ledger styles, each providing varying levels of traceability and auditability.

8.6.1.1.3. Scenario 1: Current Open-Period Correction - Partial Reallocation to Correct Resource

On July 5th, 2025, CrestNode identified that a charge record for the current billing period (July 2025) was incorrectly attributed entirely to ResourceId `R-111`. In reality, only part of the cost and usage belonged to that resource, while the remainder pertained to ResourceId `R-222`.

Since the billing period was still open and invoices had not yet been issued, the correction was applied within the same billing period, allowing for more flexible correction mechanisms. To correct the misattribution, CrestNode had the option to use any of the following approaches:

- Replacement style correction, which replaced the original record attributed to R-111 with a corrected version, and introduced a new record for `R-222` to reflect the accurate resource attribution.
- Delta style correction, which used a decrement to reduce the cost from the incorrectly attributed resource (`R-111`), and an increment to assign the cost to the correct resource (`R-222`).
- Ledger style correction, which negated the original charge and introduced two new records (one for each resource) accurately reflecting the corrected cost and usage distribution.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The correction is modeled using either Overwrite or Append mechanisms, as the billing period is still open and invoice has not yet been issued.
- Original Dataset includes:
  - A charge record attributed entirely to ResourceId `R-111`.
- Replacement style correction includes:
  - A replacement of the original record to reflect the corrected portion for `R-111`.
  - An additional record for `R-222` to account for the remaining portion of the cost and usage.
- Delta style correction includes:
  - A decrement record for `R-111`, reducing the cost previously misattributed to that resource.
  - An increment record for `R-222`, assigning the corresponding portion of the cost to the correct resource.
- Ledger style correction includes:
  - A reversal record for the original charge.
  - A corrected record for `R-111`.
  - A corrected record for `R-222`.
- Each correction record has ChargeClass set to null, indicating that it pertains to an open billing period and is not a retroactive correction to a previously closed billing period.
- Each correction record is assigned to the current billing period (July 2025).

8.6.1.1.4. Scenario 2: Current Open-Period Correction - Late-Arriving Usage

On July 5th, 2025, CrestNode identified a cost incurred during the current billing period (ChargePeriodStart: `2025-07-01`) that was not included in the initial dataset.

Since the billing period was still open and invoices had not yet been issued, the correction was applied within the same billing period, allowing for more flexible correction mechanisms. To account for the previously omitted usage, CrestNode had the option to use either Overwrite or Append mechanisms, i.e.:

- Replacement style correction
- Delta style correction
- Ledger style correction

Regardless of the correction style used, the correction was realized by introducing a single increment record representing the late-arriving usage and associated cost.

CSV Examples:

- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was incomplete and did not capture late-arriving usage and associated cost for July 2025.
- The correction may be modeled using either Overwrite or Append mechanisms, as the billing period is still open.
- All three correction styles (Replacement, Delta, and Ledger) introduce a single increment record representing the previously omitted usage and associated cost.
- The correction record has ChargeClass set to null, indicating that it pertains to an open billing period and is not a retroactive correction to a previously closed billing period.
- The correction record is assigned to the current billing period (July 2025).

8.6.1.1.5. Scenario 3: Current Open-Period Correction - Itemized Cost-Only Corrections

On July 5th, 2025, CrestNode detected a minor cost discrepancy caused by accumulated rounding differences across multiple records spanning two distinct SkuPriceId values. While each individual record was correctly rounded, the aggregated cost differed slightly from the precise total, resulting in small drifts.

Since the billing period was still open and invoices had not yet been issued, the correction was applied within the same billing period, allowing for more flexible correction mechanisms. To reconcile this discrepancy, CrestNode had the option to use either Overwrite or Append mechanisms, i.e.:

- Replacement style correction

- Delta style correction
- Ledger style correction

Regardless of the correction style used, the correction was realized by introducing two itemized increment records, each representing a cost-only adjustment for one of the affected SkuPriceId values. Unlike bulk corrections, which consolidate adjustments into a single record without specifying a SkuPriceId, this approach explicitly itemizes the correction per SkuPriceId. Because the original records were individually correct and no single record requires reversal, the Ledger style correction does not include explicit reversals. In this case, the Ledger and Delta corrections are identical, as only additive records are needed to reconcile the accumulated drift.

Compared to the bulk correction approach, this method ensures transparency and traceability and is preferred when itemized correction is feasible.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was complete in terms of usage, but a minor cost discrepancy was identified due to accumulated rounding drift across multiple records spanning two SkuPriceId values.
- The correction may be modeled using either Overwrite or Append mechanisms, as the billing period is still open.
- All three correction styles (Replacement, Delta, and Ledger) introduce two itemized increment records representing cost-only adjustments.
- Each correction record explicitly references the affected SkuPriceId.
- Each correction record has ChargeClass set to null, indicating that it pertains to an open billing period and is not a retroactive correction to a previously closed billing period.
- Each correction record has ChargeCategory set to "Adjustment", which is the only valid value when both PricingQuantity and ChargeClass are null, due to the normative requirement that PricingQuantity must not be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
- Each correction record is assigned to the current billing period (July 2025).

8.6.1.1.6. Scenario 4: Current Open-Period Correction - Bulk Cost-Only Corrections

On July 5th, 2025, CrestNode detected a minor cost discrepancy caused by accumulated rounding differences across multiple records spanning two distinct SkuPriceId values. While each individual record was correctly rounded, the aggregated cost differed slightly from the precise total, resulting in small drifts.

Since the billing period was still open and invoices had not yet been issued, the correction was applied within the same billing period, allowing for more flexible correction mechanisms. To reconcile this discrepancy, CrestNode had the option to use either Overwrite or Append mechanisms, i.e.:

- Replacement style correction
- Delta style correction
- Ledger style correction

Regardless of the correction style used, the correction was realized by introducing a single increment record representing the bulk cost-only adjustment. Unlike itemized corrections, this record did not specify a SkuPriceId, as the discrepancy spanned multiple SKU Price IDs. Because the original records were individually correct and no single record requires reversal, the Ledger style correction does not include explicit reversals. In this case, the Ledger and Delta corrections are identical, as only additive records are needed to reconcile the accumulated drift.

Compared to the itemized correction approach, this method sacrifices transparency and traceability, but is suitable when itemized correction is not feasible.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was complete in terms of usage, but a minor cost discrepancy was identified due to accumulated rounding drift across multiple records spanning two SkuPriceId values.
- The correction may be modeled using either Overwrite or Append mechanisms, as the billing period is still open.
- All three correction styles (Replacement, Delta, and Ledger) introduce a single increment record representing the bulk

cost-only adjustment to reconcile the total drift.
- The correction record does not specify a SkuPriceId, as it spans multiple SKU Price IDs.
- The correction record has ChargeClass set to null, indicating that it pertains to an open billing period and is not a retroactive correction to a previously closed billing period.
- The correction record has ChargeCategory set to "Adjustment", which is the only valid value when both PricingQuantity and ChargeClass are null, due to the normative requirement that PricingQuantity must not be null when ChargeCategory is "Usage" or "Purchase" and ChargeClass is not "Correction".
- The correction record is assigned to the current billing period (July 2025).

8.6.1.2. Previous Open-Period Correction Scenarios

These scenarios address discrepancies related to charges from the June 2025 billing period. The discrepancies were identified on July 5th, 2025, while the June 2025 billing period was still open.

8.6.1.2.1. Billing Period Alignment

Although applied after the period had ended, these corrections do not introduce retroactive semantics, as the billing period was still open and invoices had not yet been issued when the corrections were applied. The ChargeClass is set to null, and the BillingPeriodStart/End reflect the billing period in which the cost or usage was originally incurred (June).

8.6.1.2.2. Correction Style

CrestNode (acting as both the data generator and *invoice issuer*) delivers corrections using Replacement, Delta, and Ledger correction styles, offering varying levels of traceability and auditability.

8.6.1.2.3. Scenario 1: Previous Open-Period Correction - Partial Reallocation to Correct Resource

This scenario is nearly identical to *Scenario 1: Current Open-Period Correction - Partial Reallocation to Correct Resource*. The only difference is that the original misattributed charge occurred in the previous billing period (June 2025), which has ended but has not yet been closed. The correction is applied before invoice issuance, using the same correction styles: Replacement, Delta, and Ledger.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

8.6.1.2.4. Scenario 2: Previous Open-Period Correction - Late-Arriving Usage

This scenario is nearly identical to *Scenario 2: Current Open-Period Correction - Late-Arriving Usage*. The only difference is that the late-arriving usage pertains to the previous billing period (June 2025), which has ended but has not yet been closed. The correction is applied before invoice issuance, using the same correction styles: Replacement, Delta, and Ledger.

CSV Examples:

- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

8.6.1.2.5. Scenario 3: Previous Open-Period Correction - Itemized Cost-Only Corrections

This scenario is nearly identical to *Scenario 3: Current Open-Period Correction - Itemized Cost-Only Corrections*. The only difference is that the original cost discrepancy occurred in the previous billing period (June 2025), which has ended but has not yet been closed. The correction is applied before invoice issuance, using the same correction styles: Replacement, Delta, and

Ledger.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

8.6.1.2.6. Scenario 4: Previous Open-Period Correction - Bulk Cost-Only Corrections

This scenario is nearly identical to *Scenario 4: Current Open-Period Correction - Bulk Cost-Only Corrections*. The only difference is that the original cost discrepancy occurred in the previous billing period (June 2025), which has ended but has not yet been closed. The correction is applied before invoice issuance, using the same correction styles: Replacement, Delta, and Ledger.

CSV Examples:

- Original Dataset Artifact
- Replacement Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

8.6.2. Corrections to Closed Billing Period

The following examples illustrate how corrections to previously closed billing periods may be represented in FOCUS Cost and Usage dataset artifacts, using delivery mechanisms and correction styles that preserve invoice integrity and auditability.

8.6.2.1. Closed-Period Correction Scenarios

These scenarios address discrepancies related to charges from the May 2025 billing period after it was closed and invoiced by the issuer CrestNode on June 8th, 2025. The discrepancies were identified on July 5th, 2025. At that time, the June 2025 billing period was still open.

8.6.2.1.1. Billing Period Alignment

Corrections are applied to the open billing period rather than modifying the closed one. The ChargeClass is set to "Correction", and the BillingPeriodStart/End reflect the current open period (June). However, the ChargePeriodStart/End remain mapped to the original timeframe (May) to preserve the historical accuracy of when the cost was incurred.

8.6.2.1.2. Correction Style

While the Overwrite mechanism is permissible when it doesn't impact issued invoices, CrestNode defaults to the Append mechanism (Delta and Ledger styles) for all closed period corrections. This preference prioritizes auditability and traceability, ensuring that downstream consumers - particularly those managing chargeback - receive a clear, additive history of changes.

8.6.2.1.3. Scenario 1: Closed-Period Correction - Partial Reallocation to Correct Resource

On July 5th, 2025, CrestNode identified that a charge record previously invoiced for May 2025 was incorrectly attributed entirely to ResourceId `R-111`. In reality, only part of the cost and usage belonged to that resource, while the remainder pertained to ResourceId `R-222`.

To correct this misattribution, CrestNode provisioned a reallocation correction using Append mechanisms. The correction was realized either through a Delta style correction, which redistributed the cost between resources using increment and decrement records, or through a Ledger style correction, which negated the original charge and introduced corrected records for each resource.

Note: *Replacement (i.e., Overwrite delivery mechanism) could have been applied in this scenario because the reallocation would not affect invoice reconciliation and would not require additional invoices. However, CrestNode chose Append to*

preserve traceability and auditability while ensuring that downstream processes, in particular chargeback, receive timely and accurate cost attribution.

CSV Examples:

- Original Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was delivered before the billing period was closed and includes a charge record that was part of the finalized invoice for May 2025.
- Correction records have ChargeClass set to "Correction", indicating they reallocate cost from a previously closed billing period.
- Original Dataset includes:
  - A charge record attributed entirely to ResourceId `R-111`.
- Delta style correction includes:
  - A decrement record for `R-111`, reducing the cost previously misattributed to that resource.
  - An increment record for `R-222`, assigning the corresponding portion of the cost to the correct resource.
- Ledger style correction includes:
  - A reversal record for the original charge.
  - A corrected record for `R-111`.
  - A corrected record for `R-222`.

8.6.2.1.4. Scenario 2: Closed-Period Correction - Late-Arriving Usage

On July 5th, 2025, CrestNode identified a cost that was incurred during May 2025 (ChargePeriodStart: `2025-05-01`) but was not included in the finalized invoice issued on June 8th, 2025. Since the May billing period was closed, the correction was delivered in the next open billing period (e.g., June or July).

To account for the previously omitted usage, CrestNode provisioned a correction using Append mechanisms. The correction was realized by introducing a single increment record in both Delta style correction and Ledger style correction formats, representing the late-arriving cost and usage.

CSV Examples:

- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was delivered before the billing period was closed.
- The late-arriving cost was incurred during May 2025 but was not captured in the original dataset and was therefore not reflected in the invoice for May 2025, issued on June 8th, 2025.
- The correction introduces a new charge record to account for this previously omitted cost.
- The correction record is assigned to the next open billing period (e.g., June 2025).
- The correction record has ChargeClass set to "Correction", indicating it accounts for usage from a previously closed billing period.
- Both Delta style and Ledger style corrections use a single increment record to represent the late-arriving usage and associated cost.

8.6.2.1.5. Scenario 3: Closed-Period Correction - Itemized Cost-Only Corrections

On July 5th, 2025, CrestNode detected a minor cost discrepancy caused by accumulated rounding differences across multiple previously invoiced records spanning several different SkuPriceId values. While each individual record was correctly rounded, the aggregated cost differed slightly from the precise total, resulting in a small drift.

To reconcile this discrepancy, CrestNode provisioned a cost-only correction using Append mechanism. In both Delta style correction and Ledger style correction formats, the correction was realized by introducing two itemized increment records, each representing a cost-only adjustment for one of the affected SkuPriceId values. Unlike bulk corrections, which consolidate adjustments into a single record without specifying a SkuPriceId, this approach explicitly itemizes the correction per SkuPriceId. Because the original records were individually correct and no single record requires reversal, the Ledger style correction does not include explicit reversals. In this case, the Ledger and Delta corrections are identical, as only additive records are needed to reconcile the accumulated drift.

Compared to the bulk correction approach, this method ensures transparency and traceability and is preferred when itemized correction is feasible.

CSV Examples:

- Original Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was delivered before the billing period was closed.
- The original records were correctly rounded individually, but a minor discrepancy was later identified due to accumulated rounding drift across multiple records spanning two SkuPriceId values.
- The discrepancies were not captured in the original dataset and therefore were not reflected in the invoice for May 2025, issued on June 8th, 2025.
- Both Delta style and Ledger style corrections introduce two itemized increment records representing cost-only adjustments.
- Each correction record is assigned to the next open billing period (e.g., June 2025).
- Each correction record has ChargeClass set to "Correction", indicating it reconciles cost discrepancies from a previously closed billing period.
- Each correction record is itemized and explicitly references the relevant SkuPriceId.
- Each correction record has ChargeCategory set to "Adjustment". While in this case "Usage" might be more precise and is permitted (since ChargeClass is "Correction"), "Adjustment" was selected to denote a cost-only correction due to a rounding error.

8.6.2.1.6. Scenario 4: Closed-Period Correction - Bulk Cost-Only Corrections

On July 5th, 2025, CrestNode detected a minor cost discrepancy caused by accumulated rounding differences across multiple previously invoiced records spanning several different SkuPriceId values. While each individual record was correctly rounded, the aggregated cost differed slightly from the precise total, resulting in a small drift.

To reconcile this discrepancy, CrestNode provisioned a bulk cost-only correction using Append mechanism. The correction was realized by introducing a single increment record in both Delta style correction and Ledger style correction formats, representing the bulk cost-only adjustment. Unlike itemized corrections, this bulk record did not specify a SkuPriceId, as the discrepancy spanned multiple SKU Price IDs. Because the original records were individually correct and no single record requires reversal, the Ledger style correction does not include explicit reversals. In this case, the Ledger and Delta corrections are identical, as only additive records are needed to reconcile the accumulated drift.

CSV Examples:

- Original Dataset Artifact
- Delta Dataset Artifact
- Ledger Dataset Artifact

Note the following details in the example datasets:

- The original dataset was delivered before the billing period was closed.
- The original records were correctly rounded individually, but a minor discrepancy was later identified due to accumulated rounding drift across multiple records spanning two SkuPriceId values.
- The discrepancies were not captured in the original dataset and therefore were not reflected in the invoice for May 2025, issued on June 8th, 2025.
- Both Delta style and Ledger style corrections introduce a bulk cost-only record to reconcile the total drift and ensure invoice accuracy.
- The correction record is assigned to the next open billing period (e.g., June 2025).
- The correction record has ChargeClass set to "Correction", indicating it reconciles cost discrepancies from a previously closed billing period.
- The correction record does not specify a SkuPriceId, as it spans multiple SKU Price IDs.
- The correction record has ChargeCategory set to "Adjustment". While in this case "Usage" might be more precise and is permitted (since ChargeClass is "Correction"), "Adjustment" was selected to denote a cost-only correction due to a rounding error.

8.7. Examples: Invoice Detail

8.7.1. Overview

The **Invoice Detail** dataset provides a transactional representation of the financial obligations between a customer and an

invoice issuer. While the Cost and Usage dataset tracks granular consumption, the Invoice Detail dataset tracks the financial record of charges as they appear on a physical or electronic billing document.

8.7.1.1. Core Logical Pillars

To ensure interoperability across different billing systems, the dataset relies on three core logical pillars:

1. **Reconciliation & Lineage:** The dataset links every charge back to a specific document (`InvoiceId`) and allows for auditable corrections. If a charge is refunded or adjusted, the `ReferenceInvoiceId` connects the adjustment back to the original invoice, preserving the financial narrative.
2. **Granularity Definition:** Unlike standard datasets with fixed schemas, invoice line items vary wildly in detail (e.g., a single line for "Enterprise Support" vs. millions of lines for "Storage"). The `InvoiceDetailGrain` column uses a flexible JSON structure to capture the specific dimensions (SKU, Region, Project) relevant to that specific line item without breaking the schema.
3. **Currency Duality:** The dataset explicitly separates the currency of measurement (`BillingCurrency`) from the currency of settlement (`PaymentCurrency`). This allows organizations to validate usage costs in the original currency (e.g., USD) while reconciling the final cash outflow in their local currency (e.g., EUR, AUD).

8.7.1.2. Expected Value Taxonomy

The following table defines the high-level expectations for key categorical columns in this dataset:

Attribute	Expected Value Logic	Example Values
Charge Category	The high-level classification of the line item.	`Usage`, `Purchase`, `Tax`, `Credit`, `Adjustment`
Invoice Issue Status	The publication state of the document.	`Open`, `Issued`, `Voided`
Payment Terms	The agreed-upon timeframe for settlement.	`Net 30`, `Due on Receipt`, `Net 60`
Billing Currency	The currency used to measure the value of the service.	`USD`, `EUR`, `CNY`
Payment Currency	The currency required for the actual financial transfer.	`USD`, `GBP`, `AUD`

8.7.2. Examples

The following examples demonstrate some common patterns for issuing invoices from a major provider, **Aura Web**.

8.7.2.1. Scenario 1: Typical Monthly Cloud Invoice

This example includes a mix of standard consumption, a one-time purchase of a resource reservation, and taxes, all billed and paid in the same currency (USD).

- **Currencies:** Since Billing and Payment currencies are identical, `PaymentCurrencyBilledCost` equals `BilledCost`.
- **Aggregate Payment Currency**: Since no aggregate rows are present, `PaymentCurrencyInvoiceDetailId` is different for all rows.
- **Invoice Lineage:** Since this is an original invoice, `ReferenceInvoiceId` matches `InvoiceId`.

8.7.2.1.1. Data Example: INV-2025-001

Column	Line 1: Compute Usage	Line 2: Resource Reservation Purchase	Line 3: NY State Tax
Billed Cost	`450.00`	`1000.00`	`128.63`
Billing Account ID	`123456789`	`123456789`	`123456789`
Billing Currency	`USD`	`USD`	`USD`

Column	Line 1: Compute Usage	Line 2: Resource Reservation Purchase	Line 3: NY State Tax
Billing Period End	2025-02-01T00:00:00Z	2025-02-01T00:00:00Z	2025-02-01T00:00:00Z
Billing Period Start	2025-01-01T00:00:00Z	2025-01-01T00:00:00Z	2025-01-01T00:00:00Z
Charge Category	Usage	Purchase	Tax
Invoice Detail Created	2025-02-02T10:00:00Z	2025-02-02T10:00:00Z	2025-02-02T10:00:00Z
Invoice Detail Description	Use of m5.large in us-east-1	Upfront fee for Resource Reservation #998877	Sales Tax for NY Jurisdiction
Invoice Detail Grain	{"ServiceName": "Compute"}	{"ServiceName": "Compute"}	{}
Invoice Detail ID	LINE-001	LINE-002	LINE-003
Invoice Detail Last Updated	2025-02-02T10:00:00Z	2025-02-02T10:00:00Z	2025-02-02T10:00:00Z
Invoice ID	INV-2025-001	INV-2025-001	INV-2025-001
Invoice Issue Date	2025-02-03T00:00:00Z	2025-02-03T00:00:00Z	2025-02-03T00:00:00Z
Invoice Issue Status	Issued	Issued	Issued
Invoice Issuer Name	Aura Web	Aura Web	Aura Web
Payment Currency	USD	USD	USD
Payment Currency Billed Cost	450.00	1000.00	128.63
Payment Currency Invoice Detail ID	LINE-001	LINE-002	LINE-003
Payment Due Date	2025-03-05T00:00:00Z	2025-03-05T00:00:00Z	2025-03-05T00:00:00Z
Payment Terms	Net 30	Net 30	Net 30
Purchase Order Number	PO-998877	PO-998877	PO-998877
Reference Invoice ID	INV-2025-001	INV-2025-001	INV-2025-001

CSV Example

8.7.2.2. Scenario 2: Multi-Currency Settlement

This example demonstrates the "Divergent Grain" model, where usage is tracked in a global currency (USD), but the financial obligation is settled in a local currency (AUD).

- **Row 1 & 2 (Usage):** The detail lines, denominated in USD. `PaymentCurrencyBilledCost` is `0.00` because the detail does not have an exchange rate applied.
- **Row 3 (Settlement):** The aggregate payable line, denominated in AUD. `BilledCost` is `0.00` (USD) to avoid double-counting consumption. `PaymentCurrencyBilledCost` holds the 5.17 AUD obligation.
- **Aggregate Payment Currency**: Because aggregate Payment Currency rows are present, the same `PaymentCurrencyInvoiceDetailId` value of `LINE-AGG` associates these rows with each other.
- **Invoice Lineage:** Since this is an original invoice, `ReferenceInvoiceId` matches `InvoiceId`.

8.7.2.2.1. Data Example: AUIN25-1286479

Column	Line 1: Storage Usage	Line 2: Storage Tax	Line 3: AUD Settlement
Billed Cost	3.03	0.30	0.00
Billing Account ID	615703680694	615703680694	615703680694
Billing Currency	USD	USD	USD
Billing Period End	2025-07-01T00:00:00Z	2025-07-01T00:00:00Z	2025-07-01T00:00:00Z
Billing Period Start	2025-06-01T00:00:00Z	2025-06-01T00:00:00Z	2025-06-01T00:00:00Z
Charge Category	Usage	Tax	Adjustment

Column	Line 1: Storage Usage	Line 2: Storage Tax	Line 3: AUD Settlement
Invoice Detail Created	2025-07-01T12:00:00Z	2025-07-01T12:00:00Z	2025-07-01T12:00:00Z
Invoice Detail Description	AuraStore Standard Storage	Tax for AuraStore	Currency Conversion Settlement
Invoice Detail Grain	{"ServiceName": "AuraStore"}	{"ServiceName": "AuraStore"}	{}
Invoice Detail ID	LINE-001	LINE-002	LINE-AGG
Invoice Detail Last Updated	2025-07-01T12:00:00Z	2025-07-01T12:00:00Z	2025-07-01T12:00:00Z
Invoice ID	AUIN25-1286479	AUIN25-1286479	AUIN25-1286479
Invoice Issue Date	2025-07-01T00:00:00Z	2025-07-01T00:00:00Z	2025-07-01T00:00:00Z
Invoice Issue Status	Issued	Issued	Issued
Invoice Issuer Name	Aura Web	Aura Web	Aura Web
Payment Currency	AUD	AUD	AUD
Payment Currency Billed Cost	0.00	0.00	5.17
Payment Currency Invoice Detail ID	LINE-AGG	LINE-AGG	LINE-AGG
Payment Due Date	2025-08-01T00:00:00Z	2025-08-01T00:00:00Z	2025-08-01T00:00:00Z
Payment Terms	Net 30	Net 30	Net 30
Purchase Order Number	null	null	null
Reference Invoice ID	AUIN25-1286479	AUIN25-1286479	AUIN25-1286479

CSV Example

8.7.2.3. Scenario 3: Billing Error Correction

This example demonstrates the lineage of a billing error correction.

- **Line 1 (The Error):** The original charge appears on the January invoice (`INV-JAN`) with an overstated cost of 150.00.
- **Line 2 (The Correction):** In February (`INV-FEB`), the error is identified. A correction line is issued for -50.00. Crucially, the `ReferenceInvoiceId` points back to `INV-JAN`, linking the refund to the original transaction.

8.7.2.3.1. Data Example: INV-JAN and INV-FEB

Column	Line 1: Jan Usage (Overstated)	Line 2: Feb Correction (Adjustment)
Billed Cost	150.00	-50.00
Billing Account ID	987654321	987654321
Billing Currency	USD	USD
Billing Period End	2025-02-01T00:00:00Z	2025-03-01T00:00:00Z
Billing Period Start	2025-01-01T00:00:00Z	2025-02-01T00:00:00Z
Charge Category	Usage	Adjustment
Invoice Detail Created	2025-02-02T10:00:00Z	2025-03-02T10:00:00Z
Invoice Detail Description	Database Usage (Overstated)	Correction for Line-100
Invoice Detail Grain	{"ServiceName": "Database"}	{}
Invoice Detail ID	Line-100	ADJ-001
Invoice Detail Last Updated	2025-02-02T10:00:00Z	2025-03-02T10:00:00Z
Invoice ID	INV-JAN	INV-FEB
Invoice Issue Date	2025-02-03T00:00:00Z	2025-03-03T00:00:00Z
Invoice Issue Status	Issued	Issued
Invoice Issuer Name	Aura Web	Aura Web
Payment Currency	USD	USD
Payment Currency Billed Cost	150.00	-50.00
Payment Currency Invoice Detail ID	Line-100	ADJ-001
Payment Due Date	2025-03-05T00:00:00Z	2025-04-05T00:00:00Z

Column	Line 1: Jan Usage (Overstated)	Line 2: Feb Correction (Adjustment)
Payment Terms	Net 30	Net 30
Purchase Order Number	PO-554433	PO-554433
Reference Invoice ID	INV-JAN	INV-JAN

CSV Example

8.8. Examples: JSON Object

This section provides examples for the columns in the specification in the JSON Object Format.

8.8.1. Examples: Allocated Method Details

The JSON samples in the scenarios below each represent a single allocated record out of the multiple records derived from an origin record for that scenario. The sum AllocatedRatio will add up to 1 (100%) across all allocated records for an origin record, with the AllocatedRatio (or sum of AllocatedRatio) representing the allocated record's portion of the overall origin record.

8.8.1.1. Scenario 1: Single UsageUnit Value Used for Allocation

When only a single "UsageUnit" is used to calculate the allocation.

```
{
 "Elements" : [ {
  "AllocatedRatio" : 0.1,
  "UsageUnit" : "Hours",
  "UsageQuantity" : 300
  }
 ]
}
```

8.8.1.2. Scenario 2: Multiple UsageUnit Values Used for Allocation

When multiple "UsageUnit" values are used to calculate the allocation, another object is added to the "Elements" collection.

```
{
 "Elements": [
  {
   "AllocatedRatio": 0.05,
   "UsageUnit": "CPU",
   "UsageQuantity": 0.5
  },
  {
   "AllocatedRatio": 0.1,
   "UsageUnit": "Memory",
   "UsageQuantity": 4
  }
 ]
}
```

8.8.1.3. Scenario 3: Data Generator Omits Keys That are Not Required

This data generator does not wish to supply the "UsageUnit" or "UsageQuantity" keys but still provides cost allocation with some additional allocation method details. In this case, "UsageUnit" and "UsageQuantity" are omitted, and only the "AllocatedRatio" is supplied.

```
{
 "Elements" : [ {
```

```
   "AllocatedRatio" : 0.45
   }
  ]
}
```

8.8.1.4. Scenario 4: Additional Non-FOCUS Specified Properties

A data generator can add additional properties if they feel more context is helpful or necessary to the practitioner. In this scenario, the data generator is supplying additional context that shows only 0.5 of a unit was used. However, since 1 unit was requested by the service this allocation represents, the allocation is being charged at 1 regardless.

```
{
 "Elements": [
  {
   "AllocatedRatio": 0.6,
   "UsageUnit": "vCPU",
   "UsageQuantity": 1,
   "x_ReservedVCPU": 1,
   "x_UsedVCPU": 0.5,
   "x_AllocatedVCPU": 1
  }
 ]
}
```

8.8.2. Examples: Commitment Program Eligibility Details

The examples below are not exhaustive and may change over time. Service providers are the authoritative source for their *commitment programs*.

8.8.2.1. Aura Web (Partially Covered Compute Usage)

Scenario: A compute usage row that is partially covered by a Flexible Spend Plan. The eligibility column still reflects all programs this usage qualifies for, regardless of current coverage.

ServiceProviderName	ServiceName	CommitmentProgramEligibilityDetails
Aura Web	Compute	{"CommitmentPrograms": [{"ProgramType": "Flexible Spend Plan"}, {"ProgramType": "Resource Reservation"}]}

8.8.2.2. StackLens (Observability with Interval Spend Commitment)

Scenario: An observability platform usage row eligible for Monthly and Annual interval spend commitment pricing, offering lower effective rates than standard usage.

ServiceProviderName	ServiceName	CommitmentProgramEligibilityDetails
StackLens	Observability	{"CommitmentPrograms": [{"ProgramType": "Monthly Interval Spend Commitment"}, {"ProgramType": "Annual Interval Spend Commitment"}]}

8.8.2.3. LatticeScale (Ineligible Object Storage Usage)

Scenario: Standard object storage usage or a support fee, which is not eligible for any commitment program.

ServiceProviderName	ServiceName	CommitmentProgramEligibilityDetails
LatticeScale	ObjectStorage	null

8.8.2.4. Aura Web (Advance Resource Commitment-Eligible Compute Usage)

Scenario: A compute instance type and tenancy that are eligible for both discount-bearing programs and advance resource commitments. The eligibility column reflects all commitment constructs the usage qualifies for.

ServiceProviderName	ServiceName	CommitmentProgramEligibilityDetails
Aura Web	Compute	{"CommitmentPrograms": [{"ProgramType": "Flexible Spend Plan"}, {"ProgramType": "Resource Reservation"}, {"ProgramType": "Advance Resource Commitment"}, {"ProgramType": "Zonal Resource Commitment"}]}

8.8.2.5. Coverage Rate with Eligibility-Adjusted Denominator

This example demonstrates how to calculate an accurate *commitment* coverage rate using CommitmentProgramEligibilityDetails alongside CommitmentDiscountId.

Acme Corp runs compute workloads on Aura Web. Some usage is covered by a Resource Reservation, some is eligible but uncovered, and a support fee is ineligible for any *commitment program*.

Three usage rows for a single charge period (2025-04-01):

1. **Uncovered compute** (Row 1): Eligible for Flexible Spend Plan and Resource Reservation, not currently covered. BilledCost and EffectiveCost are both $200.00.
2. **Covered compute** (Row 2): Covered by a Resource Reservation. CommitmentProgramEligibilityDetails is populated. BilledCost is $0.00; EffectiveCost is $150.00.
3. **Support fee** (Row 3): Not eligible for any *commitment program*. Both CommitmentProgramEligibilityDetails and CommitmentDiscountId are null. BilledCost and EffectiveCost are both $50.00.

By filtering the denominator to rows where `CommitmentProgramEligibilityDetails IS NOT NULL`, the $50.00 support fee is correctly excluded from the eligible population:

Metric	Value
Eligible denominator	Row 1 ($200.00) + Row 2 ($150.00) = $350.00
Covered numerator	Row 2 ($150.00)
Coverage rate	`150 / 350` = **42.9%**

Row 3 (support fee) is correctly excluded because CommitmentProgramEligibilityDetails is null for ineligible charges.

CSV Example

8.8.2.6. Uncovered Eligible Spend by Program Type

This example demonstrates how to use CommitmentProgramEligibilityDetails to identify uncovered savings opportunities across *commitment program* types and providers.

Acme Corp runs compute workloads on Aura Web and uses StackLens for observability monitoring. Some Aura Web compute usage is covered by a Resource Reservation. StackLens usage is uncovered but eligible for Interval Spend Commitments at monthly or annual terms. A practitioner wants to answer: "Which *commitment program* and provider should I target for my next purchase?"

Six usage rows for a single charge period (2025-04-01):

1. **Uncovered compute** (Rows 1-2): Two Aura Web Compute rows eligible for both Flexible Spend Plan and Resource Reservation. BilledCost totals $500.00 across both rows.
2. **Covered compute** (Row 3): Aura Web Compute covered by an existing Resource Reservation. Filtered out by the query because CommitmentDiscountId is populated.
3. **Ineligible support** (Row 4): Aura Web Support with no CommitmentProgramEligibilityDetails. Filtered out because the column is null.
4. **Uncovered observability** (Rows 5-6): Two StackLens Observability rows eligible for Monthly Interval Spend Commitment and Annual Interval Spend Commitment. BilledCost totals $200.00.

To evaluate these purchasing options, the `CommitmentPrograms` JSON array must be flattened so each eligible program can be analyzed independently. By expanding the array (e.g., via `CROSS JOIN UNNEST`) and grouping the uncovered costs by

`ServiceProviderName`, `ServiceName`, and `EligibleProgramType`, the practitioner gets the summary presented in the table below. A reference implementation is available in the eligible uncovered spend query supported feature.

ServiceProviderName	ServiceName	EligibleProgramType	EligibleUncoveredCost
Aura Web	Compute	Flexible Spend Plan	$500.00
Aura Web	Compute	Resource Reservation	$500.00
StackLens	Observability	Monthly Interval Spend Commitment	$200.00
StackLens	Observability	Annual Interval Spend Commitment	$200.00

Aura Web Compute appears as $500.00 under both Flexible Spend Plan and Resource Reservation. This does not mean $1,000.00 is uncovered. The $500.00 is the same spend, and each program type represents an independent purchasing opportunity. Purchasing a Flexible Spend Plan would cover some or all of that $500.00, as would a Resource Reservation. The practitioner must choose between them (or split across both) based on flexibility requirements and discount depth.

The same logic applies to StackLens: $200.00 of observability spend could be covered by either a monthly or annual Interval Spend Commitment. The annual option typically offers a deeper discount in exchange for a longer commitment term.

The query uses BilledCost rather than EffectiveCost because all rows are uncovered (CommitmentDiscountId IS NULL). For uncovered usage, BilledCost equals EffectiveCost and reflects the actual amount paid.

CSV Example

8.8.3. Examples: Contract Applied

8.8.3.1. Scenario 1: Initial Contract Commitment

A single Cost and Usage charge represents the values stated on a contract and its three contract commitments agreed between a service provider and a customer:

1. 12345: Spend $500k overall. (This is the value of the contract, and thus ContractId = ContractCommitmentId.)
2. 23456: Spend $25k on a particular service.
3. 34567: Consume 100k compute hours on a particular resource type.

The Charge Category is denoted as Purchase, and the Contract ID, Resource ID, and Contract Commitment ID are all denoted as 12345.


```
{
 "ResourceId": "12345",
 "ChargeCategory": "Purchase",
 "BilledCost": 500000.00,
 "EffectiveCost": 0.00,
 "ContractApplied":
  {
   "Elements": [ {
    "ContractId": "12345",
    "ContractCommitmentId": "12345",
    "ContractCommitmentAppliedCost": 500000.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "23456",
    "ContractCommitmentAppliedCost": 25000.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "34567",
    "ContractCommitmentAppliedQuantity": 100000.00,
    "ContractCommitmentAppliedUnit": "compute_hours"
   } ]
  }
}
```


8.8.3.2. Scenario 2: Contract Commitment Usage with No Custom Columns

Assume the contract commitment as described in Scenario 1. Assume that only 50% of cost and usage gets applied to the

contract commitments, per the contract terms.

A single Cost and Usage charge for `myResource1` carries Effective Cost of 30 (denominated in USD) and Consumed Quantity of 1 (denominated in compute hours). The Charge Category is denoted as Usage.

This applies to the contract commitments in the following manner:


```
{
 "ResourceId": "myResource1",
 "ChargeCategory": "Usage",
 "BilledCost": 0.00,
 "EffectiveCost": 30.00,
 "ConsumedQuantity": 1,
 "ContractApplied":
  {
   "Elements": [ {
    "ContractId": "12345",
    "ContractCommitmentId": "12345",
    "ContractCommitmentAppliedCost": 15.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "23456",
    "ContractCommitmentAppliedCost": 15.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "34567",
    "ContractCommitmentAppliedQuantity": 0.50,
    "ContractCommitmentAppliedUnit": "compute_hours"
   } ]
  }
}
```


8.8.3.3. Scenario 3: Contract Commitment Usage with Custom Columns

The same as Scenario 2, except a custom key-value pair `x_ContractCommitmentCostBalance` is provided by the data generator. This datapoint represents the value remaining on a given contract commitment.


```
{
 "ResourceId": "myResource1",
 "ChargeCategory": "Usage",
 "BilledCost": 0.00,
 "EffectiveCost": 30.00,
 "ConsumedQuantity": 1,
 "ContractApplied":
  {
   "Elements": [ {
    "ContractId": "12345",
    "ContractCommitmentId": "12345",
    "ContractCommitmentAppliedCost": 15.00,
    "x_ContractCommitmentCostBalance": 499985.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "23456",
    "ContractCommitmentAppliedCost": 15.00,
    "x_ContractCommitmentCostBalance": 24985.00
   }, {
    "ContractId": "12345",
    "ContractCommitmentId": "34567",
    "ContractCommitmentAppliedQuantity": 0.50,
    "ContractCommitmentAppliedUnit": "compute_hours"
   } ]
  }
}
```

8.8.4. Examples: Contract Commitment Applicability

This section describes examples for the Contract Commitment Applicability column in the Contract Commitment dataset.

8.8.4.1. Global Scope and Applicability

If the commitment is 100% applicable to all resources, the `Applicability` object can be omitted entirely.

```
{
 "IsGlobalScope": true
}
```

8.8.4.2. Global Scope with Specific Exceptions

Organization-wide coverage **except** for Database services running in BillingAccountId 123456789012.

```
{
 "IsGlobalScope": true,
 "ExclusionOperator": "And",
 "Exclusions": [
  {
   "Dimension": "BillingAccountId",
   "Operator": "In",
   "Values": ["123456789012"]
  },
  {
   "Dimension": "ServiceCategory",
   "Operator": "In",
   "Values": ["Database"]
  }
 ]
}
```

8.8.4.3. Regional Scope

A commitment purchased for a specific region (e.g., `us-east-1`). Since `IsGlobalScope` and `IsComplexScope` are omitted, they default to `false`, requiring the inclusion block.

```
{
 "InclusionOperator": "Or",
 "Inclusions": [
  {
   "Dimension": "RegionId",
   "Operator": "In",
   "Values": ["us-east-1"]
  }
 ]
}
```

8.8.4.4. Custom Scope

A commitment that is only applicable to a specific value (Pay-As-You-Go) for a custom entity (x_BillingModel) in the `us-east-1` region.

```
{
 "InclusionOperator": "And",
 "Inclusions": [
```

```
    {
      "Dimension": "RegionId",
      "Operator": "In",
      "Values": ["us-east-1"]
    },
    {
      "Dimension": "x_BillingModel",
      "Operator": "In",
      "Values": ["Pay-As-You-Go"]
    }
  ]
}
```

8.8.4.5. Regional Compute Commitment with Exceptions

Applies to Compute services in either `us-east-1` or `us-west-2`, excluding any resources tagged with an `Environment` of `Sandbox`.

```
{
  "InclusionOperator": "And",
  "Inclusions": [
    {
      "Dimension": "RegionId",
      "Operator": "In",
      "Values": ["us-east-1", "us-west-2"]
    },
    {
      "Dimension": "ServiceCategory",
      "Operator": "In",
      "Values": ["Compute"]
    }
  ],
  "ExclusionOperator": "Or",
  "Exclusions": [
    {
      "Dimension": "Tags",
      "Operator": "Contains",
      "Values": ["\"Environment\": \"Sandbox\""]
    }
  ]
}
```

8.8.4.6. Regional Applicability

A commitment that applies fully to `us-east-1` but only 50% of cost and usage in `us-west-2` is eligible. Note the differing applicability percentages in the two regions.

```
{
  "InclusionOperator": "Or",
  "Inclusions": [
    {
      "Dimension": "RegionId",
      "Operator": "In",
      "Values": ["us-east-1"]
    },
    {
      "Dimension": "RegionId",
      "Operator": "In",
      "Values": ["us-west-2"],
      "Applicability": {
        "Cost": 0.5,
        "Usage": 0.5
```

```
    }
   }
  ]
}
```

8.8.4.7. Granular Applicability (Partial Object)

A scenario where 100% of Marketplace **Usage** counts toward a volume commitment, but only 50% of the **Cost** is applicable for financial credit. The engine defaults the missing `Usage` key to `1.0`.

```
{
 "InclusionOperator": "Or",
 "Inclusions": [
  {
   "Dimension": "InvoiceIssuerName",
   "Operator": "In",
   "Values": ["Cloud Marketplace"],
   "Applicability": {
    "Cost": 0.5
   }
  }
 ]
}
```

8.8.4.8. Complex Fallback

A commitment with dynamic or conditional logic that requires calculation against the total aggregate of cost or usage.

```
{
 "IsComplexScope": true
}
```

8.9. Examples: Metadata

The following sections contain examples of metadata provided by a hypothetical FOCUS data generator called CrestNode to supply the required reference between the *FOCUS dataset artifacts* and the Metadata. Data Generator implementations will vary on how the metadata is disseminated; however, the data generator's chosen metadata delivery approach should be able to support the structure represented in this example.

In this example, the data generator supports delivery of FOCUS data via file export to a data storage system. It uses JSON as the format for providing the metadata. The data generator delivers data every 12 hours into a path structure described below:

Type of data	Path
Export location	`/FOCUS`
Metadata location	`/FOCUS/metadata`
Cost data location	`/FOCUS/data`

Here are some metadata examples for various scenarios:

8.9.1. Data Generator Metadata

8.9.1.1. Scenario

CrestNode provides metadata about the data generator as a part of their FOCUS data export. They provide the relevant data via the Data Generator schema object.

8.9.1.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/data_generator.json`.

The updated Data Generator-related metadata could look like this:

```
{
  "DataGenerator": "CrestNode"
}
```

8.9.2. Dataset Metadata Example

8.9.2.1. Scenario

CrestNode provides two FOCUS datasets: Cost and Usage and Contract. Each Schema metadata object includes the Dataset metadata to indicate which FOCUS Dataset the Schema conforms to.

8.9.2.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-1234-abcde-12345-abcde-12345.json`.

The schema for the data artifact conforming to the dataset FOCUS Cost and Usage.

```
{
 "SchemaId": "1234-abcde-12345-abcde-12345",
 "FocusVersion": "1.0",
 "CreationDate": "2024-01-01T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
      {
          "ColumnName": "BillingAccountId",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "BillingAccountName",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
         "ColumnName": "ChargePeriodStart",
         "DataType": "DATETIME"
      },
      {
          "ColumnName": "ChargePeriodEnd",
          "DataType": "DATETIME"
      },
      {
          "ColumnName": "BilledCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "EffectiveCost",
          "DataType": "DECIMAL",
```

```
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "Tags",
            "DataType": "JSON",
            "ProviderTagPrefixes": ["crestnode", "cn"]
        }
    ]
}
```

The schema for the data artifact conforming to the dataset FOCUS Contracts.

```
{
 "SchemaId": "1234-abcde-12345-abcde-12345",
 "FocusVersion": "1.0",
 "CreationDate": "2024-01-01T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-246811",
 "ColumnDefinition": [
        {
            "ColumnName": "ContractId",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
           "ColumnName": "OverColumnName",
           "DataType": "STRING",
           "StringMaxLength": 64,
           "StringEncoding": "UTF-8"
        }
    ]
}
```

8.9.3. Deprecating Columns

8.9.3.1. Scenario

CrestNode has decided to deprecate columns prior to removal from their FOCUS data export. The column for deprecation is x_awesome_column3. The data generator creates a new Schema object to represent the new schema, with a unique SchemaId.

8.9.3.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-34567-abcde-34567-abcde-34567.json`.

The updated schema-related metadata could look like this:

```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "CreationDate": "2024-03-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
```

```
        {
            "ColumnName": "BillingAccountName",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
           "ColumnName": "ChargePeriodStart",
           "DataType": "DATETIME"
        },
        {
            "ColumnName": "ChargePeriodEnd",
            "DataType": "DATETIME"
        },
        {
            "ColumnName": "BilledCost",
            "DataType": "DECIMAL",
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "EffectiveCost",
            "DataType": "DECIMAL",
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "Tags",
            "DataType": "JSON",
            "ProviderTagPrefixes": ["crestnode", "cn"]
        },
        {
            "ColumnName": "x_awesome_column1",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
            "ColumnName": "x_awesome_column2",
            "DataType": "DATETIME"
        },
        {
            "ColumnName": "x_awesome_column3",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8",
            "Deprecated": true
         }
    ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.4. Renaming Columns

8.9.4.1. Scenario

CrestNode has decided to rename a column in their FOCUS data export. The column for rename is x_awesome_column1 and will be renamed to x_awesome_column_one. The data generator creates a new Schema object to represent the new schema, with a unique SchemaId. After this schema definition is created if the data generator creates another schema, the PreviousColumnName is removed.

8.9.4.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-34567-abcde-34567-abcde-34567.json`.

The updated schema related metadata for the schema where the rename took place could look like this:


```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "CreationDate": "2024-03-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "BillingAccountName",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
        "ColumnName": "ChargePeriodStart",
        "DataType": "DATETIME"
     },
     {
         "ColumnName": "ChargePeriodEnd",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "BilledCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "EffectiveCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "Tags",
         "DataType": "JSON",
         "ProviderTagPrefixes": ["crestnode", "cn"]
     },
     {
         "ColumnName": "x_awesome_column_one",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8",
         "PreviousColumnName": "x_awesome_column1"
     },
     {
         "ColumnName": "x_awesome_column2",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "x_awesome_column3",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8",
         "Deprecated": true
      }
   ]
```

```
}
```

The subsequent new schema metadata after the rename could look like this:

```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "CreationDate": "2024-03-02T12:01:03.083z",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "BillingAccountName",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
         "ColumnName": "ChargePeriodStart",
         "DataType": "DATETIME"
      },
      {
          "ColumnName": "ChargePeriodEnd",
          "DataType": "DATETIME"
      },
      {
          "ColumnName": "BilledCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "EffectiveCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "Tags",
          "DataType": "JSON",
          "ProviderTagPrefixes": ["crestnode", "cn"]
      },
      {
          "ColumnName": "x_awesome_column_one",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "x_awesome_column2",
          "DataType": "DATETIME"
      },
      {
          "ColumnName": "x_awesome_column3",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8",
          "Deprecated": true
       }
   ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to

8.9.5. Schema Metadata

8.9.5.1. Scenario

CrestNode has only provided one Schema for their FOCUS data export. CrestNode provides a directory of schemas and each schema is a single file. CrestNode provides a file representing the schema for the data they provide.

8.9.5.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-1234-abcde-12345-abcde-12345.json`.

The updated schema-related metadata could look like this:


```
{
 "SchemaId": "1234-abcde-12345-abcde-12345",
 "FocusVersion": "1.0",
 "CreationDate": "2024-01-01T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "BillingAccountName",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
        "ColumnName": "ChargePeriodStart",
        "DataType": "DATETIME"
     },
     {
         "ColumnName": "ChargePeriodEnd",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "BilledCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "EffectiveCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "Tags",
         "DataType": "JSON",
         "ProviderTagPrefixes": ["crestnode", "cn"]
     }
   ]
}
```

8.9.6. Schema Metadata to FOCUS Data Reference

8.9.6.1. Scenario

CrestNode makes a change to the Schema of their data exports. For each FOCUS data export, CrestNode includes a metadata reference to the schema object. Because multiple files are provided in each export, CrestNode has elected to include a metadata file in each export folder that includes the FOCUS schema reference that applies to the data export files within that folder. When the schema changes, they include the new Schema ID in their export metadata file of the new folder.

8.9.6.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/data/export1-metadata.json`

The export metadata could look like this:

```
{
 "SchemaId":"1234-abcde-12345-abcde-12345",
 "data_location":
 [
  {
   "filepath": "/FOCUS/data/export1/export1-part1.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  },
  {
   "filepath": "/FOCUS/data/export1/export1-part2.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  },
  {
   "filepath": "/FOCUS/data/export1/export1-part3.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  },
  {
   "filepath": "/FOCUS/data/export1/export1-part4.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  }
 ]
}
```

New metadata can be provided at a location such as `/FOCUS/data/export2-metadata.json`.

The new export metadata could look like this:

```
{
 "SchemaId":"23456-abcde-23456-abcde-23456",
 "data_location":
 [
  {
   "filepath": "/FOCUS/data/export2/export2-part1.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  },
  {
   "filepath": "/FOCUS/data/export2/export2-part2.csv",
   "total_bytes": 9010387,
   "total_rows": 4450
  },
  {
   "filepath": "/FOCUS/data/export2/export2-part3.csv",
   "total_bytes": 9010387,
```

```
    "total_rows": 4450
   },
   {
    "filepath": "/FOCUS/data/export2/export2-part4.csv",
    "total_bytes": 9010387,
    "total_rows": 4450
   }
  ]
}
```

8.9.7. Data Changed by Data Generator Using Data Generator Version

8.9.7.1. Scenario

CrestNode specifies the optional metadata property Data Generator Version in their Schema object. They made a change to the Cost and Usage *FOCUS dataset* they produce that does not adopt a new FOCUS Version, nor does it make a change to the included columns, but does impact values in the data. This example illustrates that Data Generator Version changes are independent of column changes, however data generator version changes may include column changes.

The data generator creates a new schema object to represent the new schema. The data generator includes both the FOCUS Version and Data Generator Version in the schema object.

8.9.7.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-56789-abcde-56789-abcde-56789.json`.

The updated schema-related metadata could look like this:

```
{
 "SchemaId": "56789-abcde-56789-abcde-56789",
 "FocusVersion": "1.1",
 "DataGeneratorVersion": "2.4",
 "CreationDate": "2024-05-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "BillingAccountName",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
        "ColumnName": "ChargePeriodStart",
        "DataType": "DATETIME"
     },
     {
         "ColumnName": "ChargePeriodEnd",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "BilledCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
```

```
        {
            "ColumnName": "EffectiveCost",
            "DataType": "DECIMAL",
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "Tags",
            "DataType": "JSON",
            "DataGeneratorTagPrefixes": ["crestnode", "cn"]
        },
        {
            "ColumnName": "x_awesome_column1",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
            "ColumnName": "x_awesome_column2",
            "DataType": "DATETIME"
        }
    ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.8. Adding New Columns

8.9.8.1. Scenario

CrestNode has decided to add additional columns to their FOCUS data export. The new columns are x_awesome_column1, x_awesome_column2, and x_awesome_column3. The data generator creates a new Schema object to represent the new schema, this schema object has a unique SchemaId. The subsequent data exports that use the new schema include the new schema's id as a reference to their corresponding schema object.

8.9.8.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-23456-abcde-23456-abcde-23456.json`.

The updated schema-related metadata could look like this:

```
{
 "SchemaId": "23456-abcde-23456-abcde-23456",
 "FocusVersion": "1.0",
 "CreationDate": "2024-02-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
        {
            "ColumnName": "BillingAccountId",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
            "ColumnName": "BillingAccountName",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
           "ColumnName": "ChargePeriodStart",
           "DataType": "DATETIME"
```

```
        },
        {
            "ColumnName": "ChargePeriodEnd",
            "DataType": "DATETIME"
        },
        {
            "ColumnName": "BilledCost",
            "DataType": "DECIMAL",
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "EffectiveCost",
            "DataType": "DECIMAL",
            "NumericPrecision": 20,
            "NumberScale": 10
        },
        {
            "ColumnName": "Tags",
            "DataType": "JSON",
            "ProviderTagPrefixes": ["crestnode", "cn"]
        },
        {
            "ColumnName": "x_awesome_column1",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
            "ColumnName": "x_awesome_column2",
            "DataType": "DATETIME"
        },
        {
            "ColumnName": "x_awesome_column3",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        }
    ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.9. Removing Columns

8.9.9.1. Scenario

CrestNode has decided to remove columns from their FOCUS data export. The column removed is x_awesome_column3. The data generator creates a new Schema object to represent the new schema, with a unique SchemaId.

8.9.9.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-34567-abcde-34567-abcde-34567.json`.

The updated schema related metadata could look like this:

```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "CreationDate": "2024-03-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
```

```
"ColumnDefinition": [
  {
    "ColumnName": "BillingAccountId",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "BillingAccountName",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
         "ColumnName": "ChargePeriodStart",
         "DataType": "DATETIME"
      },
      {
          "ColumnName": "ChargePeriodEnd",
          "DataType": "DATETIME"
      },
      {
          "ColumnName": "BilledCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "EffectiveCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "Tags",
          "DataType": "JSON",
          "ProviderTagPrefixes": ["crestnode", "cn"]
      },
      {
          "ColumnName": "x_awesome_column1",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "x_awesome_column2",
          "DataType": "DATETIME"
      }
   ]
}
```


For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.10. Changing Column Metadata

8.9.10.1. Scenario

CrestNode has decided to change the datatype of column x_awesome_column1 from a string to a number. CrestNode creates a new Schema object with the modification to x_awesome_column2.

8.9.10.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-67891-abcde-67891-abcde-67891.json`.

The updated schema-related metadata could look like this:


```
{
 "SchemaId": "67891-abcde-67891-abcde-67891",
 "FocusVersion": "1.0",
 "CreationDate": "2024-06-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "BillingAccountName",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
        "ColumnName": "ChargePeriodStart",
        "DataType": "DATETIME"
     },
     {
         "ColumnName": "ChargePeriodEnd",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "BilledCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "EffectiveCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "Tags",
         "DataType": "JSON",
         "ProviderTagPrefixes": ["crestnode", "cn"]
     },
     {
         "ColumnName": "x_awesome_column1",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "x_awesome_column2",
         "DataType": "DATETIME"
     }
   ]
}
```


For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.11. Data Generator Metadata Error Correction

8.9.11.1. Scenario

CrestNode has discovered that while their export includes the column x_awesome_column3, the Schema metadata does not include this column. In this case, the data generator fixes the metadata in the existing schema object and does not need to create a new schema object. Reference metadata remains the same.

8.9.11.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-34567-abcde-34567-abcde-34567.json`.

The updated schema-related metadata could look like this:


```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "CreationDate": "2024-03-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "BillingAccountName",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
        "ColumnName": "ChargePeriodStart",
        "DataType": "DATETIME"
     },
     {
         "ColumnName": "ChargePeriodEnd",
         "DataType": "DATETIME"
     },
     {
         "ColumnName": "BilledCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "EffectiveCost",
         "DataType": "DECIMAL",
         "NumericPrecision": 20,
         "NumberScale": 10
     },
     {
         "ColumnName": "Tags",
         "DataType": "JSON",
         "ProviderTagPrefixes": ["crestnode", "cn"]
     },
     {
         "ColumnName": "x_awesome_column1",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
     },
     {
         "ColumnName": "x_awesome_column2",
         "DataType": "STRING",
         "StringMaxLength": 64,
         "StringEncoding": "UTF-8"
```

```
      }
    ]
}
```

8.9.12. FOCUS Version Changed

8.9.12.1. Scenario

CrestNode's previous exports used FOCUS version 1.0. They are now going to adopt FOCUS version 1.1. It is required that they create a new schema metadata object which specifies the new FOCUS version via the FOCUS Version property—regardless of schema changes. In this example, the new FOCUS version adoption doesn't include columns changes. This is to illustrate that FOCUS version changes are independent of column changes, however, this scenario is unlikely.

8.9.12.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-45678-abcde-45678-abcde-45678.json`.

The updated schema-related metadata could look like this:

```
{
 "SchemaId": "45678-abcde-45678-abcde-45678",
 "FocusVersion": "1.1",
 "CreationDate": "2024-04-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
          "ColumnName": "BillingAccountName",
          "DataType": "STRING",
          "StringMaxLength": 64,
          "StringEncoding": "UTF-8"
      },
      {
         "ColumnName": "ChargePeriodStart",
         "DataType": "DATETIME"
      },
      {
          "ColumnName": "ChargePeriodEnd",
          "DataType": "DATETIME"
      },
      {
          "ColumnName": "BilledCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "EffectiveCost",
          "DataType": "DECIMAL",
          "NumericPrecision": 20,
          "NumberScale": 10
      },
      {
          "ColumnName": "Tags",
          "DataType": "JSON",
```

```
            "ProviderTagPrefixes": ["crestnode", "cn"]
        },
        {
            "ColumnName": "x_awesome_column1",
            "DataType": "STRING",
            "StringMaxLength": 64,
            "StringEncoding": "UTF-8"
        },
        {
            "ColumnName": "x_awesome_column2",
            "DataType": "DATETIME"
        }
    ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.13. FOCUS Version Changed by Data Generator Using Data Generator Version

8.9.13.1. Scenario

CrestNode specifies the optional metadata property Data Generator Version in their Schema object. Their data generator version 2.2 supported FOCUS version 1.0. They are now going to adopt FOCUS Version 1.1 which requires that they update their Data Generator Version when updating the FOCUS Version. They create a new schema object designating that both properties have changed. In this example, the adoption of the new FOCUS version doesn't include additional columns. This is to illustrate that Data Generator Version can change independent of column changes; however, this scenario is unlikely.

The data generator creates a new schema object to represent the new schema. The data generator includes both the new FOCUS Version and Data Generator Version in the schema object.

8.9.13.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/schemas/schema-45678-abcde-45678-abcde-45678.json`.

The updated schema-related metadata could look like this:

```
{
 "SchemaId": "45678-abcde-45678-abcde-45678",
 "FocusVersion": "1.1",
 "DataGeneratorVersion": "2.3",
 "name": "New Columns",
 "CreationDate": "2024-04-02T12:01:03.083z",
 "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
   "DataType": "STRING",
   "StringMaxLength": 64,
   "StringEncoding": "UTF-8"
  },
  {
   "ColumnName": "BillingAccountName",
   "DataType": "STRING",
   "StringMaxLength": 64,
   "StringEncoding": "UTF-8"
  },
  {
   "ColumnName": "ChargePeriodStart",
   "DataType": "DATETIME"
  },
  {
   "ColumnName": "ChargePeriodEnd",
```

```
    "DataType": "DATETIME"
   },
   {
    "ColumnName": "BilledCost",
    "DataType": "DECIMAL",
    "NumericPrecision": 20,
    "NumberScale": 10
   },
   {
    "ColumnName": "EffectiveCost",
    "DataType": "DECIMAL",
    "NumericPrecision": 20,
    "NumberScale": 10
   },
   {
    "ColumnName": "Tags",
    "DataType": "JSON",
    "ProviderTagPrefixes": ["crestnode", "cn"]
   },
   {
    "ColumnName": "x_awesome_column1",
    "DataType": "STRING",
    "StringMaxLength": 64,
    "StringEncoding": "UTF-8"
   },
   {
    "ColumnName": "x_awesome_column2",
    "DataType": "DATETIME"
   }
  ]
}
```


For reference, the prior schema object looked like this:


```
{
 "SchemaId": "34567-abcde-34567-abcde-34567",
 "FocusVersion": "1.0",
 "DataGeneratorVersion": "2.2",
 "CreationDate": "2024-04-02T12:01:03.083z",
 "Dataset": "FOCUS Cost and Usage",
 "ColumnDefinition": [
  {
   "ColumnName": "BillingAccountId",
   "DataType": "STRING",
   "StringMaxLength": 64,
   "StringEncoding": "UTF-8"
  },
  {
   "ColumnName": "BillingAccountName",
   "DataType": "STRING",
   "StringMaxLength": 64,
   "StringEncoding": "UTF-8"
  },
  {
   "ColumnName": "ChargePeriodStart",
   "DataType": "DATETIME"
  },
  {
   "ColumnName": "ChargePeriodEnd",
   "DataType": "DATETIME"
  },
  {
   "ColumnName": "BilledCost",
   "DataType": "DECIMAL",
   "NumericPrecision": 20,
   "NumberScale": 10
  },
```

```
    {
     "ColumnName": "EffectiveCost",
     "DataType": "DECIMAL",
     "NumericPrecision": 20,
     "NumberScale": 10
    },
    {
     "ColumnName": "Tags",
     "DataType": "JSON",
     "ProviderTagPrefixes": ["crestnode", "cn"]
    },
    {
     "ColumnName": "x_awesome_column1",
     "DataType": "STRING",
     "StringMaxLength": 64,
     "StringEncoding": "UTF-8"
    },
    {
     "ColumnName": "x_awesome_column2",
     "DataType": "DATETIME"
    }
  ]
}
```

For an example of how CrestNode ensures the schema metadata reference requirement is met see: Schema Metadata to FOCUS Data Reference

8.9.14. Dataset Instance Metadata

8.9.14.1. Scenario

CrestNode provides three dataset instances: "Cost and Usage Daily," "Cost and Usage Hourly," and "Contract Commitments," corresponding to the FOCUS datasets Cost and Usage and Contract Commitment. CrestNode also provides a metadata directory containing a single file with metadata for each dataset instance.

8.9.14.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/dataset_instances.json`.

The updated schema-related metadata could look like this:

```
[
  {
    "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-246811",
    "DatasetInstanceName": "Contract Commitments Report",
    "FocusDatasetId": "ContractCommitment"
  },
  {
    "DatasetInstanceId": "178151-dbad145e-178151-dbad145e-178151",
    "DatasetInstanceName": "Cost and Usage Daily",
    "FocusDatasetId": "CostAndUsage"
  },
  {
    "DatasetInstanceId": "178151-ja23h1287-387151-dbad145e-134657",
    "DatasetInstanceName": "Cost and Usage Hourly",
    "FocusDatasetId": "CostAndUsage"
  }
]
```

8.9.15. Recency Metadata

8.9.15.1. Scenario

CrestNode has elected to add recency metadata to its FOCUS data export. CrestNode provides a directory of recency metadata for each dataset they provide and each recency object is a single file.

8.9.15.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/recency/recency-1234-abcde-12345-abcde-12345.json`.

The provided recency metadata for a time series dataset could look like this:


```
{
 "DatasetInstanceId": "1234-abcde-12345-abcde-12345",
 "RecencyLastUpdateDate": "2025-01-291T12:01:03.083z",
 "TimeSectors": [
  {
   "TimeSectorStart": "2025-01-27T0:00:00z",
   "TimeSectorEnd" : "2025-01-27T1:00:00z",
   "TimeSectorComplete" : true,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T1:00:00z",
   "TimeSectorEnd" : "2025-01-27T2:00:00z",
   "TimeSectorComplete" : true,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T2:00:00z",
   "TimeSectorEnd" : "2025-01-27T3:00:00z",
   "TimeSectorComplete" : true,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T3:00:00z",
   "TimeSectorEnd" : "2025-01-27T4:00:00z",
   "TimeSectorComplete" : true,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T4:00:00z",
   "TimeSectorEnd" : "2025-01-27T5:00:00z",
   "TimeSectorComplete" : true,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T5:00:00z",
   "TimeSectorEnd" : "2025-01-27T6:00:00z",
   "TimeSectorComplete" : false,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  },
  {
   "TimeSectorStart": "2025-01-27T6:00:00z",
   "TimeSectorEnd" : "2025-01-27T7:00:00z",
   "TimeSectorComplete" : false,
   "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
  }
 ]
}
```


The provided recency metadata for non-time series dataset could look like this:


```
{
```

```
  "DatasetInstanceId": "54321-abcde-12345-abcde-12345",
  "RecencyLastUpdateDate": "2025-01-291T12:01:03.083z",
  "DatasetInstanceLastUpdated" : "2025-01-29T04:00:00z",
  "DatasetInstanceComplete" : true
}
```

8.9.16. Recency Metadata Update (Non Time Series)

8.9.16.1. Scenario

CrestNode provides recency metadata to accompany their FOCUS data export. CrestNode updates their FOCUS Contracts dataset, a non time-series dataset, every day. In this case, the most recent update to the recency data indicates the dataset and associated data artifact has been updated and is considered complete.

8.9.16.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/recency/recency-54321-abcde-12345-abcde-12345.json`.

The provided recency metadata for non-time series dataset could look like this:

```
{
  "DatasetInstanceId": "54321-abcde-12345-abcde-12345",
  "RecencyLastUpdate": "2025-01-291T15:01:03.083z",
  "DatasetInstanceLastUpdated" : "2025-01-29T010:00:00z",
  "DatasetInstanceComplete" : true
}
```

8.9.17. Recency Metadata Update (Time Series)

8.9.17.1. Scenario

CrestNode provides recency metadata to accompany its FOCUS data export. CrestNode updates its FOCUS Cost and Usage dataset (time series) every hour; however, the data lags by two days. Here, the most recent update to the recency data indicates the previous time sectors are now TimeSectorComplete. It also indicates that previous time sectors have been updated in the dataset. New time sectors have also been added.

8.9.17.2. Supplied Metadata

Metadata can be provided at a location such as `/FOCUS/metadata/recency/recency-1234-abcde-12345-abcde-12345.json`.

The provided recency metadata for time series dataset could look like this:

```
{
  "DatasetInstanceId": "1234-abcde-12345-abcde-12345",
  "RecencyLastUpdateDate": "2025-01-291T12:01:03.083z",
  "TimeSectors": [
    {
      "TimeSectorStart": "2025-01-27T0:00:00z",
      "TimeSectorEnd" : "2025-01-27T1:00:00z",
      "TimeSectorComplete" : true,
      "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
    },
    {
      "TimeSectorStart": "2025-01-27T1:00:00z",
```

```
    "TimeSectorEnd" : "2025-01-27T2:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T2:00:00z",
    "TimeSectorEnd" : "2025-01-27T3:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T3:00:00z",
    "TimeSectorEnd" : "2025-01-27T4:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T4:00:00z",
    "TimeSectorEnd" : "2025-01-27T5:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T5:00:00z",
    "TimeSectorEnd" : "2025-01-27T6:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T6:00:00z",
    "TimeSectorEnd" : "2025-01-27T7:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T11:15:24z"
   },
   {
    "TimeSectorStart": "2025-01-27T7:00:00z",
    "TimeSectorEnd" : "2025-01-27T8:00:00z",
    "TimeSectorComplete" : true,
    "TimeSectorLastUpdated" : "2025-01-29T11:15:24z"
   },
   {
    "TimeSectorStart": "2025-01-27T8:00:00z",
    "TimeSectorEnd" : "2025-01-27T9:00:00z",
    "TimeSectorComplete" : false,
    "TimeSectorLastUpdated" : "2025-01-29T04:00:00z"
   },
   {
    "TimeSectorStart": "2025-01-27T9:00:00z",
    "TimeSectorEnd" : "2025-01-27T10:00:00z",
    "TimeSectorComplete" : false,
    "TimeSectorLastUpdated" : "2025-01-29T10:23:10z"
   },
   {
    "TimeSectorStart": "2025-01-27T10:00:00z",
    "TimeSectorEnd" : "2025-01-27T11:00:00z",
    "TimeSectorComplete" : false,
    "TimeSectorLastUpdated" : "2025-01-29T11:15:24z"
   }
  ]
}
```


The provided recency metadata for non-time series dataset could look like this:


```
{
 "Dataset": "1234-abcde-12345-abcde-12345",
 "RecencyLastUpdateDate": "2025-01-291T12:01:03.083z",
 "DatasetInstanceLastUpdated" : "2025-01-29T04:00:00z",
```

```
  "DatasetInstanceComplete" : true
}
```

8.10. Examples: Participating Entity Identification

Understanding cost and usage data in billing datasets requires identifying the roles of several participating entities involved in resource or service provisioning, invoicing, and data generation. The FOCUS Specification includes multiple columns to identify key participating entities, these include:

- Service Provider Name
- Invoice Issuer Name
- Host Provider Name
- Data Generator

The value for each of these may vary depending on how *resources* or *services* are obtained — whether directly from a Cloud Service Provider (CSP) or a SaaS provider, via a Managed Service Provider (MSP), through a cloud marketplace, or from internal service offerings. The table below provides examples that illustrate how the value for each dimension may shift depending on the method of acquisition and other contributing factors.

#	Scenario	Service Provider	Invoice Issuer	Host Provider	Data Generator
1.1	Purchasing cloud resources or services directly from a CSP	CSP	CSP	CSP	CSP
1.2	Purchasing cloud resources or services from a CSP, where the underlying resources are operated by a 3rd party.	CSP	CSP	Entity operating the region for the CSP	CSP
2.1	Purchasing cloud resources or services via an MSP, with visibility into the underlying hosting provider.	MSP	MSP	CSP	MSP
2.2	Purchasing cloud resources or services via an MSP, without visibility into the underlying hosting provider.	MSP	MSP	MSP	MSP
2.3	Purchasing cloud-agnostic resources or services from an MSP	MSP	MSP	MSP	MSP
2.4	Purchasing labor services from an MSP	MSP	MSP		MSP
3.1.1	Purchase records for cloud marketplace offerings running on your CSP infrastructure and billed by the CSP.	Marketplace Seller	CSP	CSP	CSP
3.1.2	CSP Infrastructure usage records for cloud marketplace offerings running on your CSP infrastructure.	CSP	CSP	CSP	CSP
3.1.3	Usage records for cloud marketplace offerings running on your CSP infrastructure and billed by the CSP.	Marketplace Seller	CSP	CSP	Marketplace Seller
3.2.1	Purchase records for cloud marketplace offerings not running on your cloud infrastructure, with visibility into the underlying hosting provider.	Marketplace Seller	CSP	CSP	CSP
3.2.2	Usage records for cloud marketplace offerings not running on your cloud infrastructure, with visibility into the underlying hosting provider.	Marketplace Seller	CSP	CSP	Marketplace Seller
3.3.1	Purchase records for cloud marketplace offerings not running on your cloud infrastructure, without visibility into the underlying hosting provider.	Marketplace Seller	CSP	Marketplace Seller	CSP
3.3.2	Usage records for cloud marketplace offerings not running on your cloud infrastructure, without visibility into the underlying hosting provider.	Marketplace Seller	CSP	Marketplace Seller	Marketplace Seller
3.4.1	Purchase records for SaaS products not running on your cloud infrastructure, purchased via a reseller. Reseller is issuing payable invoices.	SaaS Provider	Reseller	SaaS Provider	SaaS Provider
3.4.2	Usage records for SaaS products not running on your cloud infrastructure, purchased via a reseller. Reseller is issuing payable invoices.	SaaS Provider	Reseller	SaaS Provider	SaaS Provider

#	Scenario	Service Provider	Invoice Issuer	Host Provider	Data Generator
3.5.1	Purchase records for SaaS products not running on your cloud infrastructure, purchased via a reseller. Reseller does not issue payable invoices.	SaaS Provider	SaaS Provider	SaaS Provider	SaaS Provider
3.5.2	Usage records for SaaS products not running on your cloud infrastructure, purchased via a reseller. Reseller does not issue payable invoices.	SaaS Provider	SaaS Provider	SaaS Provider	SaaS Provider
3.6.1	Purchase records for SaaS products that have been white-labeled and sold by a reseller, not running on your cloud infrastructure. Reseller issues payable invoices.	Reseller	Reseller	Reseller	Reseller
3.6.2	Usage records for SaaS products that have been white-labeled and sold by a reseller, not running on your cloud infrastructure. Reseller issues payable invoices.	Reseller	Reseller	Reseller	Reseller
4.1	Purchasing SaaS products directly from a SaaS provider, with visibility into the underlying hosting provider.	SaaS Provider	SaaS Provider	CSP	SaaS Provider
4.2	Purchasing SaaS products directly from a SaaS provider, without visibility into the underlying hosting provider.	SaaS Provider	SaaS Provider	SaaS Provider	SaaS Provider
4.3.1	Purchasing SaaS products running on your cloud infrastructure, purchased directly from a SaaS provider (see 4.3.2 for charges related to the underlying cloud infrastructure).	SaaS Provider	SaaS Provider		SaaS Provider
4.3.2	Purchasing resources and services from a CSP used to host SaaS products separately acquired from a SaaS provider (see 4.3.1 for charges related to the SaaS products).	CSP	CSP	CSP	CSP
5.1	Purchasing internal resources or services hosted in Data Center	Internal Name	Internal Name	Internal Name	Internal Name
6.1	Software license costs, reported separately from the costs of the resources or services they apply to.	Licensable Software Provider	License Seller		License Seller

8.11. Examples: SaaS

The following section contains examples of how SaaS data generators may implement the FOCUS specification. SaaS data generator implementations will vary on the level of detail available in their data, contract terms, purchasing options, and other factors.

8.11.1. Simple SaaS Agreements

Many SaaS providers provide simple contract terms, therefore don't need to support complex scenarios like spend commitments or pricing strategies in their billing data.

The scenarios described below illustrate how a Cost and Usage *FOCUS dataset* should look for simple SaaS agreement scenarios (these scenarios may not be specific to SaaS agreements only).

8.11.1.1. Scenario A1: Invoice Up-Front for a Purchase of a Service

PipelCRM allows its customers to purchase their service for a term (in this case, a year) for a $10,000. PipelCRM provides Acme Corp with a single invoice for their usage. PipelCRM does not provide detailed cost and usage reports to Acme Corp throughout the Charge Period after the initial purchase.

Given that PipelCRM does not charge based on or track usage, its usage details are irrelevant to this scenario.

CSV Example

Note the following details in the example dataset:

- The Charge Period is April 1st 2025 - April 1st 2026. The Billing Period is the month of April 2025 (when the licenses were ordered) and therefore will appear in the April invoice.
- A single charge representing the total payment for the 12-month agreement ($10,000) is charged in the first invoice. BilledCost and EffectiveCost are realized in the same record since detailed usage records will not be provided during the 12-month period to realize amortized portions of this up-front payment.
- The single charge record does not include a List Unit Price, Pricing Quantity, or SKU-related information. Alternatively, the Pricing Quantity could have been set to 1, and the List Unit Price could be the same as the total charge.

8.11.1.2. Scenario A2: Invoice Up-Front for a Quantity of a Service

PipelCRM offers its customer the ability to purchase a fixed quantity of licenses for their service. PipelCRM provides Acme Corp with a single invoice for their usage. PipelCRM does not provide detailed cost and usage reports to Acme Corp throughout the Charge Period after the initial purchase.

On April 1st, 2025, PipelCRM executes a contract and invoices Acme Corp $50,000 (Billed Cost) for a Charge Period of April 1st 2025 to April 1st 2026. As there is no negotiated discount, List Cost of the purchase is also $50,000.

CSV Example

Note the following details in the example dataset:

- The Charge Period is April 1st 2025 to April 1st 2026. The Billing Period is the month of April 2025 (when the licenses were ordered) and therefore will appear in the April invoice.
- A single charge representing the total payment for the 12-month agreement is charged in the first invoice. Billed Cost and Effective Cost are both realized in the same record since detailed usage records will not be provided during the 12-month period to realize amortized portions of this up-front payment.
- The single charge provided includes a ListUnitPrice for the licenses and a Pricing Quantity.

8.11.1.3. Scenario A3: Additional Purchase Records Provided in the SaaS Data Generator's FOCUS Dataset

On June 1st 2025 PipelCRM provides the following records due to Acme Corp's $1,000 mid-contract purchase of an additional 10 licenses for the same Charge Period (April 1st 2025 to April 1st 2026).

CSV Example

Note the following additional details in the example dataset:

- The Charge Period is still April 1st 2025 to April 1st 2026. The Billing Period is now the month of June 2025 (when the additional licenses were ordered) and therefore will appear in the June 2025 invoice.

8.11.1.4. Scenario B: Billed in Arrears for a Quantity of a Service

Similar to Scenario A above, PipelCRM offers its customer the ability to purchase their service with a fixed quantity of licenses. However, in Scenario B, PipelCRM issues the invoice at the end of the usage period.

On April 1st, 2026, PipelCRM invoices Acme Corp $50,000 (Billed Cost) for the Charge Period of April 1st 2025 to April 1st 2026. As there is no negotiated discount, List Cost of the purchase is also $50,000.

CSV Example

Note the following additional details in the example dataset:

- The Charge Period is April 1st 2025 to April 1st 2026. The Billing Period is now the month of March 2026 (since this charge is invoiced as of the last month of the Charge Period).

8.11.1.5. Scenario C: Simple SaaS Agreement with Monthly Billing

Like Scenario A2 above, PipelCRM offers its customers the ability to purchase their service with a fixed quantity of licenses. However, in Scenario C, PipelCRM issues invoices at the end of each month (usage period). For this scenario, contract terms additionally include the following terms:

- PipelCRM charges users monthly for the licenses that were consumed in that Billing Period
- The licenses are charged at $20 per license per month

Acme Corp's consumption looks like this:

- In April 2025, Acme Corp uses 505 licenses
- In May 2025, Acme Corp uses 650 licenses
- In June 2025, Acme Corp uses 635 licenses

CSV Example

Note the following additional details in the example dataset:

- The Charge Period and Billing Period are April 1st, 2025, to May 1st, 2025, for the first month. Subsequent months increment the Charge Period and Billing Period by one month to match the month the charges are incurred.
- Billed Cost and Effective Cost are the same value since there is no up-front payment to amortize

8.11.2. SaaS Spend Agreements

Many SaaS service providers support billing models that allow (or in some cases require) consumers to agree to an amount to spend over a period. In some cases, customers receive a negotiated discount for usage during that period in exchange for the spend agreement. Spend agreements can have different payment models like billing in arrears or pre-paid contracts and may impose minimum spend requirements for parts of the agreement.

The scenarios described below illustrate how a Cost and Usage *FOCUS dataset* should look for various spend agreement scenarios.

8.11.2.1. Baseline Scenario

The following baseline conditions apply to the scenarios described below:

- Acme Corp has signed an agreement with SaaS service provider StoreStack to use their database services
- On April 1 2025, Acme Corp agrees to spend $1200 (post-discounts) in the upcoming 12-months
- Acme Corp receives a 20% negotiated discount in return for the commitment
- StoreStack calculates the spend counted against the agreements after discounts (like the negotiated discounts). Other service providers may use the cost after discounts i.e., using List Cost for calculating the spend commitment.

8.11.2.2. Scenario A: Billed in Arrears

For this scenario A, contract includes the following terms in addition to the baseline scenario mentioned above:

- All charges will be billed in arrears at a monthly frequency

8.11.2.2.1. Scenario A1: Billed in Arrears with No Minimum Spend Requirement Per Month

For this scenario, contract additionally includes the following terms:

- Committed spend can be used anytime within the 1-year commitment period.

Acme Corp's consumption looks like this:

- In the first month, Acme Corp uses $48 of services (4 server hours). This usage has a List Cost of $60 (before discounts)
- In the following 2 months, Acme Corp has some more usage
- For the final 9 months, Acme Corp does not use StoreStack services

CSV Example

Note the following details in the example dataset:

- A single charge representing the total unused amount from the 12-month agreement is charged during the final month of the 12-month commitment period

8.11.2.2.2. Scenario A2: Billed in Arrears with a Minimum Spend Requirement Per Month

The spend agreement with StoreStack requires the customer to spend a minimum amount in each Billing Period (monthly). Unused fees are charged per Billing Period when the consumption is below this level (use-it or lose-it). For this scenario, contract additionally includes the following terms:

- A minimum of $60 needs to be spent each month

Acme Corp's consumption looks like this:

- In the first month, Acme Corp uses $48 of services (4 server hours). This usage has a List Cost of $60 (before discounts). For this month, StoreStack charges $12 (ListCost of $15) for not meeting the monthly minimum
- In the following 2 months, Acme Corp has usage at or above the minimum requirement
- For the final 9 months, Acme Corp does not use StoreStack services

CSV Example

Note the following details in the example dataset:

- A monthly charge representing the unused minimum monthly amount is charged during months 4 through 11 of the 12-month commitment period
- The final month has a charge that captures the overall unmet spend requirement for the 12-month contract. Alternatively, this could be provided as two charges, one for the unused portion of the final month, and one to capture the overall unmet spend requirement.

8.11.2.3. Scenario B: Prepaid Contract

For this scenario B, contract includes the following terms in addition to the baseline scenario mentioned above:

- The charges will be billed in arrears using monthly invoices

8.11.2.3.1. Scenario B1: Prepaid with No Minimum Spend Requirement Per Month

Scenario B1 is similar to scenario A1 with the difference being that it's a pre-paid contract.

CSV Example

Note the following details in the example dataset:

- A purchase record for the initial $1200 payment is present representing List, Billed, and Contracted cost of the purchase
- The charge for the unused amount has a $0 BilledCost (since the total amount was billed with the prepayment). However, the charge captures the unused portion as an EffectiveCost.
- The unused charge rows apply to the entire Charge Period the contract was signed for.
- This scenario shows List Cost and Contracted Cost column double counting dynamic (described here in ListCost and ContractedCost) where either the ChargeCategory Purchase or Usage rows need to be excluded depending on the reporting scenario.

8.11.2.3.2. Scenario B2: Prepaid with a Minimum Spend Requirement Per Month

Scenario B2 is similar to A2 with the difference being that it's a pre-paid contract.

CSV Example

Note the following details in the example dataset:

- A purchase record for the initial $1200 payment is present representing List, Billed, and Contracted cost of the purchase
- The monthly charge for the unused amount has a $0 BilledCost (since the total amount was billed with the prepayment). However, the charge captures the unused portion as an EffectiveCost.
- The final month has a charge that captures the overall unmet spend requirement for the 12-month contract. Alternatively, this could be provided as two charges, one for the unused portion of the final month, and one to capture the overall unmet spend requirement.
- This scenario shows List Cost and Contracted Cost column double counting dynamic (described here in ListCost and ContractedCost) where either the Purchase or Usage rows need to be excluded depending on the reporting scenario.

8.11.3. Virtual Currency Pricing Model

Many SaaS service providers support pricing models that utilize virtual currencies such as credits, tokens, or points. Charges may be provided using a virtual currency, which can subsequently be converted to a national currency such as USD or EUR at an advertised or agreed-upon conversion rate.

The scenarios described below illustrate how a Cost and Usage *FOCUS dataset* should look for various scenarios where a provider utilizes this pricing model.

8.11.3.1. Baseline Scenario

The following baseline conditions apply to the scenarios described below:

- Acme Corp has signed an agreement with SaaS service provider OmniQuery to use their services.
- OmniQuery offers a virtual currency pricing model for their services and requires a purchase of virtual currency in advance of usage. Their denomination of virtual currency is called "tokens".
- OmniQuery requires purchase of additional tokens in the event of usage exceeding purchased tokens.
- OmniQuery publicly lists the cost of their tokens at $2 per token.
- OmniQuery treats token purchases as resources; therefore, charges for token purchases include values for ResourceId, ResourceName, and ResourceType.
- OmniQuery publicly lists their usage to token rates. These rates are as follows:
    - 1 Q Widget Execution = 1 token
    - 1 Z Widget Execution = 2 tokens
    - 1 Workflow Operation = 3 tokens

8.11.3.2. Scenario A: Virtual Currency Not Offered at a Discount

For this scenario, contract terms include the following terms in addition to the baseline scenario mentioned above:

- Acme Corp offers no discount for purchased tokens.

8.11.3.3. Scenario A1: Purchase of Virtual Currency Without a Discount

For this scenario, the initial purchase of virtual currency is executed as follows:

- On April 1, 2025, Acme Corp agrees to purchase 100,000 tokens at $2 per token for a total spend $200,000. These tokens are only valid for 12 months.

CSV Example

Note the following details in the example dataset:

- The Charge Period is April 1st 2025 - April 1st 2026. The Billing Period is the month of April 2025 (when the tokens were purchased) and therefore will appear in the April invoice.
- Because OmniQuery uses a virtual currency pricing model for usage and publishes their token price in terms of dollars and their usage cost in terms of tokens, their Cost and Usage *FOCUS dataset* includes the columns PricingCurrency, PricingCurrencyContractedUnitPrice, PricingCurrencyEffectiveCost, and PricingCurrencyListUnitPrice.
- A single charge representing the total payment for the initial token purchase agreement ($200,000) is charged in the first invoice.
    - ListCost, BilledCost, and ContractedCost of the purchase are all represented in this charge, however EffectiveCost is zero since the tokens are not yet consumed.
- PricingQuantity is set to the total tokens purchased.
- Because Acme Corp is paying the list price, ListUnitPrice and ContractedUnitPrice are all set to the same value of $2.

8.11.3.4. Scenario A2: Usage of Virtual Currency Purchased Without a Discount

Acme Corp uses OmniQuery's services consuming tokens as follows in the first day:

- 245 executions of Q Widget
- 5 executions of Z Widget
- 120 operations of Workflow

CSV Example

Note the following details in the example dataset:

- The Charge Period is April 1st 2025 - April 2nd 2025. The Billing Period is the month of April 2025.
- PricingCurrency for these usage charges reflects the per usage token price of the particular usage.
- PricingQuantity reflects the amount of usage of the PricingUnit for each charge and is equivalent to ConsumedQuantity. While relevant to this example, there are scenarios including tiered pricing where ConsumedQuantity and PricingQuantity may not be the same.
- Because Acme Corp's usage includes no discount on usage to token rates, PricingCurrencyContractedUnitPrice and

PricingCurrencyListUnitPrice are equivalent.

8.11.3.5. Scenario B: Virtual Currency Offered at a Discount

For this scenario, contract terms include the following terms in addition to the baseline scenario mentioned above:

- Acme Corp offers a discount for purchased tokens.

8.11.3.6. Scenario B1: Purchase of Virtual Currency at a Discount

For this scenario, the initial purchase of virtual currency is executed as follows:

- On April 1, 2025, Acme Corp agrees to purchase 100,000 tokens at discounted cost of $1 per token for a total spend $100,000. These tokens are only valid for 12 months.

CSV Example

Note the following details in the example dataset:

- The Charge Period is April 1st 2025 - April 1st 2026. The Billing Period is the month of April 2025 (when the tokens were purchased) and therefore will appear in the April invoice.
- Because OmniQuery uses a virtual currency pricing model for usage and publishes their token price in terms of dollars and their usage cost in terms of tokens, their *FOCUS dataset* includes the columns PricingCurrency, PricingCurrencyContractedUnitPrice, PricingCurrencyEffectiveCost, and PricingCurrencyListUnitPrice.
- A single charge representing the total payment for the initial token purchase agreement ($100,000) is charged in the first invoice.
    - ListCost, BilledCost, and ContractedCost of the purchase are all represented in this charge, however EffectiveCost is zero, as required for prepaid purchases.
- PricingQuantity is set to the total tokens purchased.
- Because Acme Corp is receiving a discount on the token price, the ListUnitPrice is set to $2 and the ContractedUnitPrice is set to $1. A ListCost of ($200,000) and ContractedCost ($100,000) reflect the cost of the tokens at the list price and contracted price respectively. The BilledCost is set to $100,000 since this is the amount that Acme Corp will be charged for the purchase of tokens.

8.11.3.7. Scenario B2: Usage of Virtual Currency Purchased at a Discount

Acme Corp uses OmniQuery's services, consuming tokens as follows in the first day:

- 245 executions of Q Widget
- 5 executions of Z Widget
- 120 operations of Workflow

CSV Example

Note the following details in the example dataset:

- PricingQuantity reflects the amount of usage of the PricingUnit for each charge and is equivalent to ConsumedQuantity. While relevant to this example, there are scenarios including tiered pricing where ConsumedQuantity and PricingQuantity may not be the same.
- Because Acme Corp's usage includes no discount on usage to token rates, PricingCurrencyContractedUnitPrice and PricingCurrencyListUnitPrice are equivalent.

8.11.3.8. Scenario B3: Usage of Virtual Currency at a Modified Rate

Acme Corp uses OmniQuery's services consuming tokens as follows in the first day:

- 245 executions of Q Widget
- 5 executions of Z Widget
- 120 operations of Workflow

Additionally, OmniQuery offers a modified usage to token ratio for one of their services as follows:

- 1 Workflow Operation = 2 tokens

CSV Example

Note the following details in the example dataset:

- Because of the modified rate for Workflow Operations, the PricingCurrencyContractedUnitPrice and PricingCurrencyListUnitPrice are different for this charge. The ContractedUnitPrice is set to $1 and the ListUnitPrice is set to $2.
- The PricingCurrencyEffectiveCost is 240 tokens for this charge, which is less than example B2 above due to the modified rate.
- ListCost reflects the cost of the charge at both the list cost of the tokens and the list rate for which the usage consumes tokens.

8.11.3.9. Scenario C: Handling Virtual Currency Usage Overages

For this scenario, Acme Corp has exceeded their purchased tokens on October 1st 2025 by 1,500 tokens and OmniQuery has charged them for the overage. The following conditions apply:

- OmniQuery has charged Acme Corp for the cost of tokens at the list price of $2 per token, and this purchase is effective from April 1st 2025 to the date of the purchase, October 1st 2025.
- Acme Corp purchases an additional 25,000 tokens to facilitate usage to the end of their contract. These tokens are valid from October 1st 2025 to April 1st 2026.

CSV Example

Note the following details in the example dataset:

- This example focuses on the purchases records only for the overage and additional purchases. Neither usage charges nor earlier purchases are not included in this example.
- The Charge Period for the Overage Purchase is April 1st 2025 - October 1st 2025. This is because the overage charge is to cover the period of time the overage token purchase is applicable to.
- The Charge Period for the Additional Purchase is October 1st 2025 - April 1st 2026. This is because the additional purchase is to cover the period of time to which the additional token purchase is applicable. Because end dates are exclusive, ChargePeriodEnd is April 1st 2026.

8.11.4. Billing Scenario Examples

The following examples illustrate how BilledCost and EffectiveCost behave across common software as a service (SaaS) and platform as a service (PaaS) billing models. Each scenario uses a fictional *service provider* with illustrative pricing and demonstrates a distinct billing pattern that SaaS or PaaS data generators may implement via the FOCUS specification.

Each example targets a specific billing pattern. All examples share a consistent column set; columns not applicable to a given scenario contain null values.

Scenario	Service Provider	What You'll Learn
Credit-Based Consumption	OmniQuery	Custom consumption units (Credits) with `ChargeFrequency` split between "Usage-Based" (compute) and "Recurring" (storage). No *commitment discount*; BilledCost = EffectiveCost on all rows.
Host-Based SaaS Monitoring	StackLens	Multiple *services* billed on independent metrics (hosts vs. GB). "Recurring" for host-based charges, "Usage-Based" for log ingestion. No regional billing.
Seat-Based SaaS Subscription	SprintCanvas	Upfront annual purchase amortized to monthly Usage rows. BilledCost vs. EffectiveCost divergence. Spend-based `CommitmentDiscountCategory`, One-Time `ChargeFrequency`.
Multi-Unit PaaS Database	StoreStack	Three heterogeneous PricingUnit values (Hours, GB, GB) within one *service provider*. Region-specific billing with RegionId/RegionName populated.
Flat-Rate SaaS Licensing	CollabChat	Fixed monthly subscription where PricingUnit is "Subscriptions" and PricingQuantity is 1, decoupled from underlying user count.
Annual Commitment Billed Monthly	PipelCRM	Annual term contract with monthly billing where the billed rate equals list price. No *commitment discount* despite the annual obligation.
Tiered Pricing with Committed Minimum	PulseMail	Plan fee as a usage-denominated *commitment discount* with Used/Unused split and overage pricing. `CommitmentDiscountCategory` = "Usage". Two billing periods showing under- and over-minimum scenarios.

8.11.4.1. Credit-Based Consumption: On-Demand Data Platform Usage

A data platform *service provider*, OmniQuery, uses a credit-based consumption model. Customers consume credits based on warehouse compute activity and pay a fixed per-credit rate determined by their service edition. Storage is billed separately on a per-terabyte basis.

The *service provider*'s on-demand pricing for this example:

Service	SKU	Unit Price	Pricing Unit	Credits/Hour
Virtual Warehouse Compute	XS Warehouse	$3.00	Credits	1
Virtual Warehouse Compute	Medium Warehouse	$3.00	Credits	4
Storage	Active Storage	$23.00	TB	n/a

Credit consumption varies by warehouse size. An XS warehouse consumes 1 credit per hour. A Medium warehouse consumes 4 credits per hour.

A customer runs two warehouses in the US East region during January 2025:

- XS warehouse (analyst workload): 200 hours of runtime = 200 credits consumed
- Medium warehouse (ETL workload): 80 hours of runtime = 320 credits consumed (4 credits/hour)
- 5 TB of active storage

Three usage charges appear on the January invoice:

- XS Warehouse: `200 Credits x $3.00` = $600.00
- Medium Warehouse: `320 Credits x $3.00` = $960.00
- Active Storage: `5 TB x $23.00` = $115.00

Total BilledCost: $1,675.00

Here is how these charges appear in the data (relevant columns only):

FOCUS Column	XS Warehouse	Medium Warehouse	Active Storage
ChargeCategory	Usage	Usage	Usage
ChargeFrequency	Usage-Based	Usage-Based	Recurring
ConsumedQuantity	200	320	5
ConsumedUnit	Credits	Credits	TB
BilledCost	$600.00	$960.00	$115.00
EffectiveCost	$600.00	$960.00	$115.00
ListCost	$600.00	$960.00	$115.00
ContractedCost	$600.00	$960.00	$115.00
PricingUnit	Credits	Credits	TB
PricingCategory	Standard	Standard	Standard

Key observations:

- Both warehouses share the same ListUnitPrice of $3.00 per credit. The credit price is determined by the service edition, not warehouse size.
- ConsumedUnit is "Credits" for compute and "TB" for storage. The credit is the *service provider*'s unit of compute consumption.
- ChargeFrequency is "Usage-Based" for compute (metered by credit consumption) and "Recurring" for storage (billed per TB per month).
- Storage has no ResourceId or ResourceName because it is an account-level charge, not tied to a specific *resource*.
- BilledCost and EffectiveCost are equal on all rows. These two columns diverge when a purchase *charge* covers future usage (as with a *commitment discount*), because the *cash-based* outflow is invoiced upfront while cost is recognized on an *accrual basis* over time. With no such purchase *charges* here, the two views produce the same cost per row.

CSV Example

8.11.4.2. Host-Based SaaS Monitoring: Monthly On-Demand Usage

A SaaS observability *service provider*, StackLens, offers multiple monitoring *services* billed on different units: host-based pricing for infrastructure and application performance monitoring, and volume-based pricing for log ingestion.

The *service provider*'s on-demand pricing for this example:

Service	SKU	Unit Price	Pricing Unit
Infrastructure Monitoring	Infra Pro	$18.00	Hosts
APM	APM Standard	$36.00	Hosts
Log Management	Log Ingestion	$0.10	GB

A customer uses the *service provider*'s monitoring platform on a month-to-month basis with no annual commitment. During January 2025, the customer's usage is:

- 25 infrastructure hosts monitored
- 10 APM hosts monitored (a subset of the infrastructure hosts, billed independently)
- 150 GB of logs ingested

Three usage charges appear on the January invoice:

- Infrastructure Monitoring: `25 Hosts x $18.00` = $450.00
- APM: `10 Hosts x $36.00` = $360.00
- Log Management: `150 GB x $0.10` = $15.00

Total BilledCost: $825.00

Here is how these charges appear in the data (relevant columns only):

FOCUS Column	Infra Monitoring	APM	Log Management
ChargeCategory	Usage	Usage	Usage
ChargeFrequency	Recurring	Recurring	Usage-Based
ConsumedQuantity	25	10	150
ConsumedUnit	Hosts	Hosts	GB
BilledCost	$450.00	$360.00	$15.00
EffectiveCost	$450.00	$360.00	$15.00
ListCost	$450.00	$360.00	$15.00
ContractedCost	$450.00	$360.00	$15.00
PricingUnit	Hosts	Hosts	GB
PricingCategory	Standard	Standard	Standard

Key observations:

- ChargeFrequency is "Recurring" for the host-based *services* (billed per host per month regardless of utilization within the month) and "Usage-Based" for log ingestion (billed on actual volume consumed).
- APM hosts are a subset of infrastructure hosts but each *service* is billed independently. The 10 APM hosts also appear in the 25 infrastructure host count.
- RegionId and RegionName are null. This *service provider* does not expose region in its billing. Some SaaS and PaaS *service providers* do bill by region, in which case these columns would be populated (see the Multi-Unit PaaS example below).
- ResourceId and ResourceName are null. The *service provider* bills at the *service* level (per host or per GB), not per individual monitored *resource*.
- All three *services* share the ServiceCategory "Management and Governance" per the FOCUS allowed values for logging, monitoring, and observability *services*.
- BilledCost and EffectiveCost are equal on all rows because there are no purchase *charges* covering future usage (see the Credit-Based Consumption example for a full explanation of when these columns diverge).

CSV Example

8.11.4.3. Seat-Based SaaS Subscription: Annual Upfront with Commitment Discount

This example illustrates an annual upfront SaaS subscription where the customer receives a discounted per-user rate by committing to a 12-month term. The discounted rate is only available with the annual commitment, making this a *commitment discount*.

The *service provider*, SprintCanvas, offers a project management platform with the following pricing for 50 users on the Standard plan:

Billing Option	Unit Price	Monthly Cost (50 users)	Annual Cost
Monthly	$9.05/user/month	$452.50	$5,430.00
Annual	$7.58/user/month	$379.00	$4,550.00

The annual option represents a ~16% discount versus monthly billing.

A customer subscribes to the Standard plan for 50 users on April 1, 2025. They choose the annual billing option, paying $4,550.00 upfront for a 12-month term ending April 1, 2026. All 50 seats are occupied in the first month.

Two charges appear in the April 2025 *billing period*:

Here is how these charges appear in the data (relevant columns only):

FOCUS Column	Purchase	Usage (April)
ChargeCategory	Purchase	Usage
ChargeFrequency	One-Time	Recurring
CommitmentDiscountCategory	Spend	Spend
CommitmentDiscountStatus	*(null)*	Used
CommitmentDiscountQuantity	4,550.00	379.00
CommitmentDiscountUnit	USD	USD
ConsumedQuantity	*(null)*	50
ConsumedUnit	*(null)*	Users
BilledCost	$4,550.00	$0.00
EffectiveCost	$0.00	$379.00
ListCost	$5,430.00	$452.50
ContractedCost	$5,430.00	$452.50
PricingUnit	Count	Users
PricingCategory	Standard	Committed

Purchase Charge: ChargeCategory is "Purchase" with ChargeFrequency "One-Time". The full annual amount is invoiced in a single payment.

- The *charge period* spans the entire commitment term: April 1, 2025 through April 1, 2026.
- BilledCost is $4,550.00. This is the *cash-based* invoiced amount for the annual subscription.
- EffectiveCost is $0.00. The purchase covers future usage. Cost is recognized on an *accrual basis* as usage occurs.
- ListCost is $5,430.00 (`50 users x $9.05/user/month x 12 months`). This represents the cost without the annual discount.
- ContractedCost is $5,430.00, equal to ListCost. Because no *negotiated discounts* apply (the annual rate is a published *commitment discount*, not a bilateral negotiation), ContractedUnitPrice defaults to ListUnitPrice per the spec. The difference between ContractedCost ($5,430.00) and BilledCost ($4,550.00) represents the $880.00 in *commitment discount* savings at the point of purchase.
- ResourceId equals CommitmentDiscountId. The *commitment discount* itself is the *resource* being purchased.
- CommitmentDiscountCategory is "Spend" because the commitment is denominated in dollars, not usage units.
- CommitmentDiscountQuantity is 4,550.00 CommitmentDiscountUnit (USD). This is the total spend eligible for consumption over the term.
- PricingCategory is "Standard". The purchase of the *commitment discount* is at the agreed-upon rate.

Usage Charge (April 2025): ChargeCategory is "Usage" with PricingCategory "Committed". The usage is covered by the annual purchase.

- BilledCost is $0.00. No additional invoiced amount. The usage is covered by the purchase charge.
- EffectiveCost is $379.00. This is the *accrual-based* recognized portion of the annual commitment: `$7.58/user x 50 users`.
- ListCost is $452.50 (`50 users x $9.05/user`). The monthly list cost without the annual discount.
- ContractedCost is $452.50, equal to ListCost. No *negotiated discounts* apply, so ContractedUnitPrice defaults to ListUnitPrice.
- The *commitment discount* savings appear in the difference between ContractedCost ($452.50) and EffectiveCost ($379.00): $73.50 per month.
- CommitmentDiscountStatus is "Used". All 50 seats are occupied.
- CommitmentDiscountQuantity is 379.00 USD. This is the amount of spend consumed from the commitment in this *charge period*.
- ResourceId is the actual *resource* (the customer's project management site), not the *commitment discount*.

Rounding Note: The per-user rate of $7.58 is derived from `$4,550 / 12 months / 50 users` = $7.5833, rounded to two decimal places. This means `$7.58 x 50 users x 12 months` = $4,548.00, which is $2.00 less than the $4,550.00 purchase. In a full 12-month dataset, the final month's charge would include a true-up to ensure the sum of EffectiveCost across all usage rows equals the BilledCost of the purchase row.

CSV Example

8.11.4.4. Multi-Unit Usage-Based PaaS: Database-as-a-Service

A PaaS database *service provider*, StoreStack, bills different resource types on different units. The *service provider* offers dedicated database clusters with separate charges for compute, storage, and data transfer.

The *service provider*'s on-demand pricing for this example:

Service	SKU	Unit Price	Pricing Unit
StoreStack Clusters	M10 Cluster	$0.08	Hours
StoreStack Clusters	M30 Cluster	$0.54	Hours
StoreStack Storage	SSD Storage	$0.25	GB
StoreStack Data Transfer	Data Transfer Out	$0.01	GB

A customer runs two dedicated database clusters in the US East region for the full month of January 2025 (744 hours). They also consume 100 GB of SSD storage and 50 GB of outbound data transfer.

- 1x M10 cluster (analytics workload): 744 hours at $0.08/hour
- 1x M30 cluster (production workload): 744 hours at $0.54/hour
- 100 GB of SSD storage
- 50 GB of outbound data transfer

Four usage charges appear on the January invoice:

- M10 Cluster: `744 Hours x $0.08` = $59.52
- M30 Cluster: `744 Hours x $0.54` = $401.76
- SSD Storage: `100 GB x $0.25` = $25.00
- Data Transfer Out: `50 GB x $0.01` = $0.50

Total BilledCost: $486.78

Here is how these charges appear in the data (relevant columns only):

FOCUS Column	M10 Cluster	M30 Cluster	SSD Storage	Data Transfer Out
ChargeCategory	Usage	Usage	Usage	Usage
ChargeFrequency	Usage-Based	Usage-Based	Recurring	Usage-Based
ConsumedQuantity	744	744	100	50
ConsumedUnit	Hours	Hours	GB	GB
BilledCost	$59.52	$401.76	$25.00	$0.50
EffectiveCost	$59.52	$401.76	$25.00	$0.50
ListCost	$59.52	$401.76	$25.00	$0.50
ContractedCost	$59.52	$401.76	$25.00	$0.50
PricingUnit	Hours	Hours	GB	GB
PricingCategory	Standard	Standard	Standard	Standard
RegionId	us-east-1	us-east-1	us-east-1	us-east-1

Key observations:

- Three different PricingUnit values appear within a single *service provider*: Hours (compute), GB (storage), and GB (data transfer). This heterogeneity is typical for PaaS database *services*.
- ChargeFrequency varies by resource type. Compute clusters are "Usage-Based" (metered hourly). Storage is "Recurring" (billed per GB provisioned per month regardless of access patterns). Data transfer is "Usage-Based" (billed on actual volume transferred).
- Compute clusters have ResourceId and ResourceName values because each cluster is a distinct *resource*. Storage and data transfer are account-level charges with no specific *resource*, so these columns are null.
- All four charges share the ServiceCategory "Databases" but are split across three distinct ServiceName values: StoreStack Clusters, StoreStack Storage, and StoreStack Data Transfer.
- RegionId and RegionName are populated. This *service provider* deploys database clusters to specific regions, and pricing varies by region.
- BilledCost and EffectiveCost are equal on all rows because there are no purchase *charges* covering future usage (see the Credit-Based Consumption example for a full explanation of when these columns diverge).

CSV Example

8.11.4.5. Flat-Rate SaaS Licensing: Fixed Monthly Subscription

A team communications *service provider*, CollabChat, offers both per-user and flat-rate subscription tiers. This example uses the flat-rate option, where all features and unlimited users are included for a fixed monthly fee with no per-user pricing.

The *service provider*'s pricing for this example:

Service	SKU	Unit Price	Pricing Unit
CollabChat	Pro Unlimited	$349.00	Subscriptions

The *service provider* also offers an annual billing option at $299.00 per month (billed annually). This example uses the month-to-month option with no annual commitment.

A customer subscribes to the Pro Unlimited plan on a month-to-month basis. During January 2025, the customer's team of 35 people uses the platform.

One charge appears on the January invoice:

- Pro Unlimited: `1 Subscription x $349.00` = $349.00

Total BilledCost: $349.00

Here is how this charge appears in the data (relevant columns only):

FOCUS Column	Pro Unlimited
ChargeCategory	Usage
ChargeFrequency	Recurring
ConsumedQuantity	1
ConsumedUnit	Subscriptions
BilledCost	$349.00
EffectiveCost	$349.00
ListCost	$349.00
ContractedCost	$349.00
PricingUnit	Subscriptions
PricingQuantity	1
PricingCategory	Standard

Key observations:

- ChargeFrequency is "Recurring". The subscription is a fixed monthly fee regardless of usage activity or user count within the *billing period*.
- PricingUnit is "Subscriptions" and PricingQuantity is 1. Unlike per-seat or per-unit models, the entire platform is a single billable unit.
- There is no relationship between ConsumedQuantity and the number of users. The 35-person team does not affect the charge.
- If the customer chose the annual billing option ($299.00/month billed annually), this would follow a *commitment discount* pattern similar to the Seat-Based SaaS Subscription example above, with a Purchase row for the annual payment and monthly Usage rows for amortized EffectiveCost.
- BilledCost and EffectiveCost are equal because there are no purchase *charges* covering future usage (see the Credit-Based Consumption example for a full explanation of when these columns diverge).

CSV Example

8.11.4.6. Annual Commitment Billed Monthly: Seat-Based CRM

A CRM *service provider*, PipelCRM, offers a seat-based sales platform requiring an annual commitment. Unlike prepaid annual subscriptions, one billing option charges monthly throughout the contract term. The monthly billed rate equals the list price, so no *commitment discount* is applied in FOCUS terms.

The *service provider*'s pricing for this example (Professional plan, 10 users):

Billing Option	Unit Price	Monthly Cost (10 users)	Annual Cost
Annual (pay upfront)	$90.00/user/month	$900.00	$10,800.00
Annual (billed monthly)	$100.00/user/month	$1,000.00	$12,000.00

A customer subscribes to the Professional plan for 10 users, choosing the annual commitment with monthly billing. They pay $100.00 per user per month with no upfront payment.

One charge appears on the January 2025 invoice:

- Sales Platform Professional: `10 Users x $100.00` = $1,000.00

Total BilledCost: $1,000.00

Here is how this charge appears in the data (relevant columns only):

FOCUS Column	Sales Platform Professional
ChargeCategory	Usage
ChargeFrequency	Recurring
ConsumedQuantity	10
ConsumedUnit	Users
BilledCost	$1,000.00
EffectiveCost	$1,000.00
ListCost	$1,000.00
ContractedCost	$1,000.00
PricingUnit	Users
PricingCategory	Standard

Key observations:

- BilledCost, EffectiveCost, ListCost, and ContractedCost are all equal at $1,000.00. Monthly billing on an annual contract produces no divergence between *cash-based* and *accrual-based* costs because there is no upfront payment to amortize.
- PricingCategory is "Standard" because the billed rate equals theListUnitPrice. The annual contract is a term commitment, not a pricing discount. No *commitment discount* columns are populated.
- ChargeFrequency is "Recurring" because the charge recurs monthly at a fixed per-seat rate regardless of usage activity within the *billing period*.
- If the customer chose the annual pay-upfront option ($90.00/user/month), the $10.00 per-seat discount would qualify as a *commitment discount*, following the pattern in the Seat-Based SaaS Subscription example with a Purchase row and amortized EffectiveCost.
- This scenario demonstrates that an annual contract does not automatically produce a *commitment discount* in FOCUS. The distinguishing factor is whether the commitment provides a price reduction from the standard rate.

CSV Example

8.11.4.7. Tiered Pricing with Committed Minimum: Email API Platform

A SaaS email API *service provider*, PulseMail, offers tiered plans that include a monthly email allowance. Emails sent within the allowance are covered by the plan fee. Emails exceeding the allowance are billed at a per-email overage rate. The plan minimum functions as a usage-denominated *commitment discount* because the customer pays a fixed fee for a quantity of usage units.

The *service provider*'s pricing for this example (Essentials 50K plan):

Component	Rate	Unit
Plan fee (includes 50,000 emails)	$19.95	Count
Overage	$0.00133	Emails

This example covers two billing periods to show both under-minimum and over-minimum scenarios:

- **January 2025:** The customer sends 30,000 emails, below the 50,000 email allowance. 20,000 emails go unused.
- **February 2025:** The customer sends 65,000 emails, exceeding the allowance by 15,000.

January charges (under minimum):

The monthly plan fee of $19.95 appears as a Purchase charge. Two Usage charges split the commitment between used and unused portions:

- Purchase: $19.95 (plan fee covering 50,000 emails)
- Used: 30,000 emails, EffectiveCost = $11.97 (`30,000 / 50,000 x $19.95`)
- Unused: 20,000 emails, EffectiveCost = $7.98 (`20,000 / 50,000 x $19.95`)

Total BilledCost: $19.95

Here is how the January charges appear in the data (relevant columns only):

FOCUS Column	Purchase	Usage (Used)	Usage (Unused)

FOCUS Column	Purchase	Usage (Used)	Usage (Unused)
ChargeCategory	Purchase	Usage	Usage
ChargeFrequency	Recurring	Usage-Based	Usage-Based
CommitmentDiscountCategory	Usage	Usage	Usage
CommitmentDiscountStatus	*(null)*	Used	Unused
CommitmentDiscountQuantity	50,000	30,000	20,000
CommitmentDiscountUnit	Emails	Emails	Emails
ConsumedQuantity	*(null)*	30,000	*(null)*
ConsumedUnit	*(null)*	Emails	*(null)*
BilledCost	$19.95	$0.00	$0.00
EffectiveCost	$0.00	$11.97	$7.98
ListCost	$19.95	$39.90	$26.60
ContractedCost	$19.95	$39.90	$26.60
PricingUnit	Count	Emails	Emails
PricingCategory	Standard	Committed	Committed

February charges (over minimum):

The customer uses all 50,000 plan emails plus 15,000 overage emails:

- Purchase: $19.95 (plan fee covering 50,000 emails)
- Used: 50,000 emails, EffectiveCost = $19.95 (full allowance consumed)
- Overage: 15,000 emails at $0.00133 = $19.95 (standard pricing, not part of commitment)

Total BilledCost: $39.90

Here is how the February charges appear in the data (relevant columns only):

FOCUS Column	Purchase	Usage (Used)	Usage (Overage)
ChargeCategory	Purchase	Usage	Usage
ChargeFrequency	Recurring	Usage-Based	Usage-Based
CommitmentDiscountCategory	Usage	Usage	*(null)*
CommitmentDiscountStatus	*(null)*	Used	*(null)*
CommitmentDiscountQuantity	50,000	50,000	*(null)*
CommitmentDiscountUnit	Emails	Emails	*(null)*
ConsumedQuantity	*(null)*	50,000	15,000
ConsumedUnit	*(null)*	Emails	Emails
BilledCost	$19.95	$0.00	$19.95
EffectiveCost	$0.00	$19.95	$19.95
ListCost	$19.95	$66.50	$19.95
ContractedCost	$19.95	$66.50	$19.95
PricingUnit	Count	Emails	Emails
PricingCategory	Standard	Committed	Standard

Key observations:

- CommitmentDiscountCategory is "Usage" on all commitment-related rows. The plan minimum is denominated in email quantity (50,000 emails), not a dollar amount. This distinguishes it from the "Spend" category in the Seat-Based SaaS Subscription example.
- CommitmentDiscountStatus shows "Used" and "Unused" on Usage rows. In January, the customer only consumed 30,000 of 50,000 emails, so the remaining 20,000 generate an "Unused" row. The Unused row has BilledCost of $0.00 but EffectiveCost of $7.98, representing waste from the commitment.
- The Purchase row has BilledCost of $19.95 and EffectiveCost of $0.00. This follows the same pattern as the Seat-Based SaaS Subscription: the purchase captures the *cash-based* cost, while *accrual-based* cost is recognized on the Usage rows.
- The overage row in February has no *commitment discount* columns populated. Overage emails are billed at the standard per-email rate and are not part of the commitment.
- ContractedUnitPrice and ListUnitPrice are both $0.00133 on all Usage rows. Because no *negotiated discounts* apply, ContractedUnitPrice defaults to ListUnitPrice per the spec. The *commitment discount* savings are reflected in the difference between ContractedCost and EffectiveCost, not in ContractedUnitPrice.
- ListCost and ContractedCost on Usage rows represent the market value of those emails at the per-email rate: `$0.00133 x 30,000` = $39.90 for the January Used row. These values exceed BilledCost ($0.00) because the usage is covered by the plan fee, not billed individually. The gap between ListCost and EffectiveCost shows the *commitment discount* benefit

per row.
- PricingUnit is "Count" on the Purchase row because the plan fee is a single monthly purchase, not denominated in the usage unit (emails). On Usage rows, PricingUnit is "Emails" because those rows are priced per email.
- ChargeFrequency is "Recurring" on the Purchase rows (the plan fee recurs monthly). On the Usage rows, ChargeFrequency is "Usage-Based" for both Used and Unused rows because the amounts vary based on actual email consumption each period.

CSV Example

8.12. Grouping Constructs for Resources or Services

Service Providers natively support various constructs for grouping *resources* or *services*. These grouping constructs are often used to mimic organizational structures, technical architectures, cost attribution/allocation and access management boundaries, or other customer-specific structures based on requirements.

Service Providers may support multiple levels of resource or service grouping mechanisms. FOCUS supports two distinct levels of groupings that are commonly needed for FinOps capabilities like chargeback, invoice reconciliation and cost allocation.

- *Billing account*: A mandatory container for *resources* or *services* that are billed together in an invoice. *Billing accounts* are commonly used for scenarios like grouping based on organizational constructs, invoice reconciliation and cost allocation strategies.
- Sub account: An optional service-provider-supported construct for organizing *resources* and *services* connected to a *billing account*. *Sub accounts* are commonly used for scenarios like grouping based on organizational constructs, access management needs and cost allocation strategies. *Sub accounts* must be associated with a *billing account* as they do not receive invoices.

The table below highlights key properties of the two grouping constructs supported by FOCUS.

Property	Billing account	Sub account
Requirement level	Mandatory	Optional
Receives an invoice?	Yes	No
Invoiced at	Self	Associated billing account
Examples	AWS: Management Account* GCP: Billing Account Azure MCA: Billing Profile Snowflake: Organizational Account	AWS: Member Account GCP: Project Azure MCA: Subscription Snowflake: Account

* For organizations that have multiple AWS Member Accounts within an AWS Organization, consolidated billing is enabled by default and invoices are received at Management Account level. A Member Account can be removed from AWS consolidated billing whereby the removed account receives independent invoices and is responsible for payments.

8.13. Invoice and Billing Period Handling

8.13.1. Overview

A primary use case for FinOps practitioners is the reconciliation of invoices and usage statements. In FOCUS, this process is supported through *FOCUS datasets*, notably the Cost and Usage, Invoice Detail, and Billing Period datasets.

In the context of FOCUS, successful *invoice reconciliation* relies on all monetary data appearing on an invoice or usage statement (including non-usage charges such as taxes, credits, refunds, support, training, and marketplace transactions) being accurately captured and categorized in these datasets.

Without this fundamental alignment, downstream processes like financial reporting and chargeback become unreliable.

8.13.1.1. Invoice Reconciliation and Issuance

Before an *invoice is issued*, i.e., before the Invoice Issue Status in the Invoice Detail dataset transitions to "Issued", data generators typically perform internal *invoice reconciliation* to ensure consistency between the invoice, the Invoice Detail dataset, and the Cost and Usage dataset.

At the conclusion of this process, a key objective is for the aggregated Billed Costs in the Invoice Detail dataset for a given Invoice Detail ID to align with the payable amounts presented on the corresponding invoice line items. This alignment is performed across Invoice ID, Invoice Detail ID, and Invoice Issuer.

Similarly, practitioners rely on the aggregated Billed Costs in the Invoice Detail dataset to reconcile with the aggregated Billed Costs in the Cost and Usage dataset for the same identifiers, subject to Rounding Variance Tolerance.

Practitioners may perform *invoice reconciliation* independently by verifying that invoice line items align with data delivered in *FOCUS dataset artifacts*, particularly Cost and Usage, Invoice Detail, and Billing Period.

Once an invoice is issued, it becomes an authoritative financial document, and the information it contains is expected not to change.

Corrections related to *issued invoices* (i.e., updates, additions, or omissions of underlying records in Cost and Usage, Invoice Detail, and Billing Period datasets) are permitted in accordance with the Invoice Issue Status requirements, as well as the Billed Cost requirements in both Invoice Detail and Cost and Usage datasets. In other words, such *corrections* are typically performed upon explicit request or approval by the customer, provided that they do not compromise the integrity of the issued invoice.

Corrections to underlying records that do not impact *invoice reconciliation* are allowed regardless of Invoice Issue Status. However, even in this case, they may reduce auditability and traceability or affect downstream processes (e.g., cost allocation, chargeback, reporting).

8.13.1.2. Open and Closed Billing Periods

A *closed billing period* represents a billing period for which all anticipated invoices have been successfully issued by the designated *invoice issuers*, and no additional invoices are generally expected to be associated with that period, except where explicitly requested or approved by the customer. In contrast, an *open billing period* remains subject to ongoing billing activities until it is formally closed.

The Billing Period dataset provides the necessary context to determine the status of each billing period for a specific invoice issuer. Since invoice issuer and *billing period*-related columns are present in all three *FOCUS datasets* (Billing Period, Cost and Usage, and Invoice Detail), records across the three datasets can be consistently associated with the corresponding billing cycles.

Corrections related to *closed billing periods* (i.e., updates; additions; or omissions of underlying records in Cost and Usage, Invoice Detail, and Billing Period *FOCUS datasets*) are permitted in accordance with the Billing Period Status requirements, as well as the Billed Cost requirements in both Invoice Detail and Cost and Usage datasets. In other words, such *corrections* are typically performed upon explicit request or approval by the customer, provided that they do not compromise the integrity of the *closed billing period*.

Corrections to underlying records that do not impact the integrity of the *closed billing period* or *invoice reconciliation*, such as informational or metadata updates, are allowed regardless of Billing Period Status.

If the original *closed billing period* is not reopened, corrections that require issuing additional invoices are best represented by associating them with a subsequent *open billing period*. This approach preserves historical financial accuracy, ensures clear temporal boundaries between billing cycles, and guarantees that all corrections are transparently tracked and auditable in future periods.

8.14. Rounding Variance Tolerance

When performing invoice reconciliation between the Cost and Usage dataset and an invoice (either the payable invoice itself, or the Invoice Detail dataset), the totals may not be perfectly equal due to precision differences (e.g., 6 decimals in detailed cost data vs. 2 decimals in invoice data). The following tolerance formula allows for a maximum rounding error based on the statistical probability of rounding variance, which grows with the square root of the row count.

8.14.1. Tolerance Formula

The tolerance used when comparing aggregated BilledCost values is defined as:

Tolerance = `MAX(100 × Subunit, (SQRT(Rows) × 0.5) × Subunit)`

Where:

- **Rows** — The number of CostAndUsage rows included in the aggregation for the relevant CostAndUsage.InvoiceIssuerName, CostAndUsage.InvoiceId, and (if applicable) CostAndUsage.InvoiceDetailId.
- **Subunit** — The numeric value of the smallest subunit of CostAndUsage.BillingCurrency (for example, `0.01` for USD or `1` for JPY).

The tolerance is the **greater** of the following values:

- **100 × Subunit** — Establishes a fixed minimum tolerance equal to 100 units of the smallest currency subunit.

- **(SQRT(Rows) × 0.5) × Subunit** — Provides a tolerance that increases with the square root of the number of rows to account for rounding accumulation in larger datasets.

8.14.2. Scenario 1: Small Invoice (Pass)

- Data: A small invoice with 5 underlying CostAndUsage rows.
- Values: CostAndUsage sums to $12.50. InvoiceDetail sums to $12.52. Difference is **$0.02**.
- Limit Calculation:
  - Statistical Limit: `SQRT(5) * 0.5 * 0.01` = $0.011.
  - Floor Limit: `100 * 0.01` = $1.00.
  - Effective Tolerance: **$1.00** (Greater of $0.011 and $1.00).
- Result: **Pass** (Difference $0.02 < Tolerance $1.00).

8.14.3. Scenario 2: Small Invoice (Fail)

- Data: A small invoice with 5 underlying CostAndUsage rows where a $5.00 charge is missing from CostAndUsage.
- Values: CostAndUsage sums to $10.00. InvoiceDetail sums to $15.00. Difference is **$5.00**.
- Limit Calculation:
  - Effective Tolerance: **$1.00**.
- Result: **Fail** (Difference $5.00 > Tolerance $1.00).

8.14.4. Scenario 3: Large Invoice (Pass)

- Data: An enterprise invoice with 1,000,000 underlying CostAndUsage rows.
- Values: CostAndUsage sums to $5,000,000.00. InvoiceDetail sums to $5,000,004.50. Difference is **$4.50**.
- Limit Calculation:
  - Statistical Limit: `SQRT(1,000,000) * 0.5 * 0.01` = $5.00.
  - Floor Limit: `100 * 0.01` = $1.00.
  - Effective Tolerance: **$5.00** (Greater of $5.00 and $1.00).
- Result: **Pass** (Difference $4.50 < Tolerance $5.00).

8.14.5. Scenario 4: Large Invoice (Fail)

- Data: An enterprise invoice with 1,000,000 underlying CostAndUsage rows with a systematic rounding error (truncation bias).
- Values: CostAndUsage sums to $5,000,000.00. InvoiceDetail sums to $5,000,015.00. Difference is **$15.00**.
- Limit Calculation:
  - Effective Tolerance: **$5.00**.
- Result: **Fail** (Difference $15.00 > Tolerance $5.00).

9. Credits

The sections below recognize FOCUS Working Group members for their contributions to this release. The recognition convention reflects the role distinctions established in the FOCUS Operating Procedures:

- **Maintainers** are designated only after consistent prior contribution as a Contributor (§2.2.4), so substantive contribution is implicit in the Maintainer role; Maintainers are not separately listed under Contributors.
- **Contributors** are individuals who substantively contributed to this release and are not Maintainers. Steering Committee members who substantively contributed also appear here, since contribution (§2.2.1) is distinct from the Steering Committee's governance role.
- **Steering Committee Members** serve in a governance and release ratification role per the FOCUS Steering Committee charter. Members may also appear in Maintainers or Contributors sections to reflect additional roles.

9.1. Maintainers

Thanks to the following FOCUS Maintainers for their leadership and contributions to the FOCUS Release **v1.4** specification.

- Andrew Quigley (Northwestern Mutual)
- Irena Jurica (Neos)
- Joaquin Prado (FinOps Foundation)
- Karl Kraft (Walmart)
- Larry Advey (CloudZero)
- Matt Cowsert (FinOps Foundation)
- Michael Flanakin (Salesforce)
- Mike Fuller (FinOps Foundation)
- Riley Jenkins (Domo)
- Shawn Alpay (FinOps Foundation)
- Udam Dewaraja (StitcherAI)

9.2. Contributors

Thanks to the following FOCUS members for their contributions to the FOCUS Release **v1.4** specification.

- Alexandra McCoy (A.M. Tech)
- Alfred Francis (thecloudx.co)
- Andrew Qu (Everest)
- Beau Nelford (Anglepoint)
- Ben Olson (Adobe)
- Christopher Harris (Datadog)
- David Earney (American Express)
- Deeja Cruz (Datadog)
- Erik Norman (Caligo)
- George Parker (Salesforce)
- Graham Murphy (TechnologyOne)
- Greg Kroleski (Databricks)
- Justin Grinstead (MongoDB)
- Justin Marks (Amazon Web Services)
- Marc Perreaut (Amadeus)
- Nan Braun (Thavron Solutions)
- Ramkumar Narla (Adobe)
- Rich Kreitz (Grafana)
- Rob Martin (FinOps Foundation)
- Ruben Vander Stockt (Telenet Group)
- Rowdy Voss (Oracle)
- Sai Pydiganta (Amazon Web Services)
- Sanjna Srivatsa (Broadcom)
- Tim Wright (Google)

9.3. Steering Committee Members

Thanks to the following FOCUS Steering Committee members for their leadership on the FOCUS Release **v1.4** specification.

- Ben Olson (Adobe)
- Casey Doran (IBM Cloudability)
- Christopher Harris (Datadog) (term ended 2026/03/30)
- Deeja Cruz (Datadog)
- Jerzy Grzywinski (Capital One) (term ended 2026/03/03)
- John Ely (Capital One)
- Karl Kraft (Walmart)
- Letian Feng (Amazon Web Services)
- Mike Fuller (FinOps Foundation)
- Sarah McMullin (Google)
- Tim O'Brien (Walmart) (term ended 2025/11/10)
- Vikram Desai (Microsoft)