Common Warehouse Metamodel (CWM)

Specification

Volume 2. Extensions

Version 1.0, 2 February 2001

Preface 1

1.1 Introduction

CWM provides interchange for the common portions of data warehouse tool metamodels. However, the CWM metamodel is also intended to be a foundation for tool-specific metamodels and may be extended to be a tool-specific metamodel.

There are two general extension techniques to CWM:

1. Use of the general extension mechanisms provided by the UML Object Model, namely Tagged Values and Stereotypes. This approach is normally used for minor extensions (for example additional attributes of model objects) that are not significant enough to require the production of a tool-specific model.
2. Non-normative model extensions documented as additional metamodel packages that extend the CWM metamodel.

In both cases, tools must share a common understanding of the extension to be able to interchange the extended information.

The chapters provided in this volume represent examples of non-normative model extensions to the CWM metamodel. They should be treated only as examples, and may be changed or updated by the vendors without notice or revision to this volume.

1.2 Guide to Volume 2

Volume 2 consists of the following chapters:

Chapter 1 Preface

Introduces this volume.

Chapter 2 Entity Relationship

Describes the ER package which contains classes and associations that represent metadata of entity-relationship models. This package is an extension of the CWM Foundation package.

Chapter 3 COBOL Data Division

Describes the COBOL Data package which contains classes and associations that represent metadata of COBOL Data Divisions. This package is an extension of the CWM Record package.

Chapter 4 DMS II

Describes the DMS II package which contains classes and associations that represent metadata of DMS II data resources. This package is an extension of the CWM Record package.

Chapter 5 IMS

Describes the IMS Database package which contains classes and associations that represent metadata of IMS data resources. This package is an extension of the CWM Record package.

Chapter 6 Essbase

Describes the Essbase package which contains classes and associations that represent metadata of Essbase data resources. This package is an extension of the CWM Multidimensional package.

Chapter 7 Express

Describes the Express package which contains classes and associations that represent metadata of Express data resources. This package is an extension of the CWM Multidimensional package.

Chapter 8 Information Set

Describes the InformationSet package which contains classes and associations that represent metadata of Information Set tools. This package is an extension of the CWM OLAP package.

Chapter 9 Information Reporting

Describes the InformationReporting package which contains classes and associations that represent metadata of information reporting tools. This package is an extension of the CWM InformationVisualization package.

References

Lists the references used in this volume.

1.3 Organization of the CWM Extensions (CWMX)

The CWMX Metamodel uses the same package structure as CWM to control complexity, promote understanding, and support reuse. The packages are grouped as follows:

- Foundation package
  - Entity Relationship (ER) package
- Resource package
  - COBOL Data package
  - DMS II package
  - IMS Database package
  - Essbase package
  - Express package
- Analysis package
  - Information Set package
  - Information Reporting package

Entity Relationship 2

2.1 Overview

Entity Relationship (ER) models are used frequently as a means of describing business processes and the data on which they operate. The ER model was a precursor of today's object models and is probably the first data model to have the adjective "semantic" applied to it.

Although the ER model is widely used to capture some aspects of application logic and data structure, there have been surprisingly few implementations of the model as data resources. Most development teams have preferred to map their ER models to existing data systems such as relational database management systems. This dearth of physical implementations has meant that modelers have been free to elaborate the basic ER model in any (and all) convenient directions with little or no impact on deployed information systems. Consequently, variants of the ER model abound.

Because of its importance as a design and tool model, the CWM includes a foundational ER model from which individual tool models may derive their specific extensions. Doing so will improve the extent to which ER models can be interchanged between various tooling environments.

2.2 Organization of the Entity Relationship Package

The ER package depends on the following packages:

- org.omg::CWM::Core
- org.omg::CWM::Relationships
- org.omg::CWM::Foundation::Expressions
- org.omg::CWM::Foundation::KeysIndexes

Many ER model concepts map directly onto equivalent CWM concepts, making the ER classes seem to be little more than renamings of CWM classes. However, the renaming provided by deriving ER model classes from appropriate CWM classes is

considered valuable to promote understanding. In such cases, the discussion of the role of ER classes has been kept minimal. However, when important modeling choices were required, they are discussed in the following class descriptions.

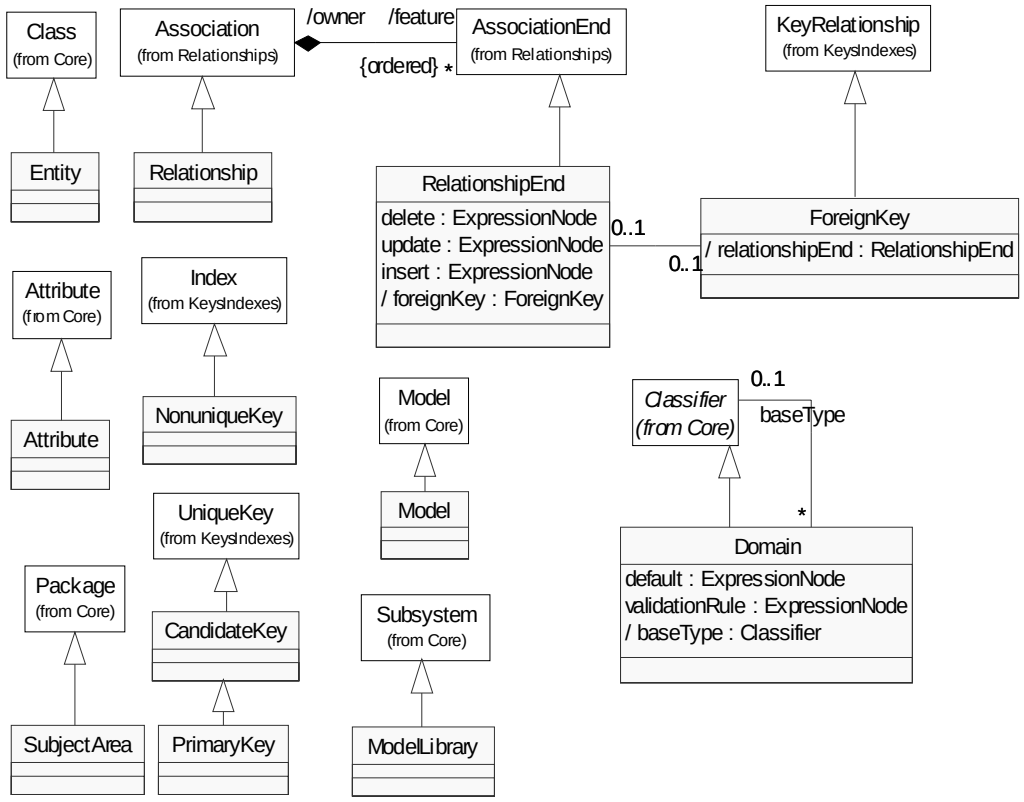

Figure 2-1 Entity Relationship Package

2.3 Entity Relationship Classes

2.3.1 CandidateKey

Candidate keys are keys that meet the requirements for being a primary key. However, only keys that are members of the PrimaryKey subclass have actually been identified as primary keys.

Superclasses

UniqueKey

2.3.2 Attribute

Instances of the ER model Attribute class represent characteristics of some Entity or Relationship instance.

Superclasses

Attribute

2.3.3 Domain

Domain instances represent restrictions on data types declared elsewhere and can be used as the type of Attribute instances. Domains restrict, in a manner described by their validationRule attribute, the values of the type identified via the baseType reference that can be stored in the Attribute. Because the baseType reference is optional, Domains are not required to have a base type; in such cases, the type of the Domain is the type of the default expression.

The following figures illustrate ways that Domains can be used to subset a enumerated and numeric data types.

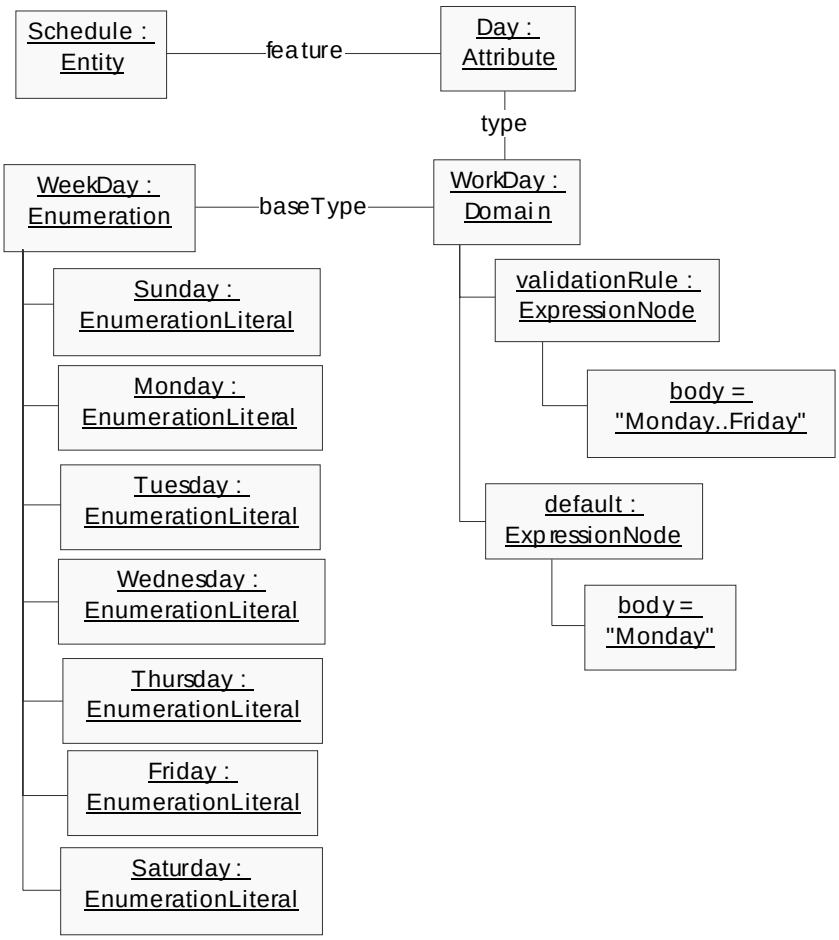

Figure 2-2 Using Domains to subset an enumerated type.

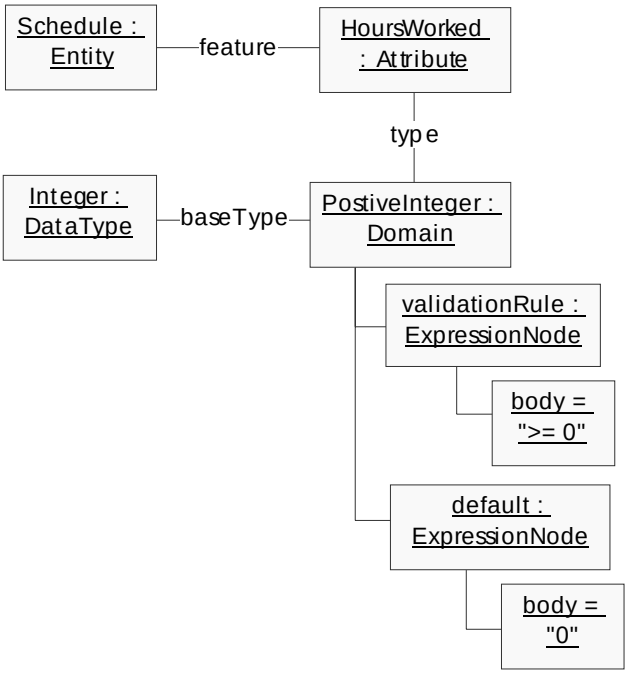

Figure 2-3 Using Domains to subset a numeric data type.

Superclasses

Classifier

Attributes

validationRule

Contains an expression that describes the valid values for this attribute. If the baseType reference is not empty, the expression restricts the values of the base type indicated by it.

type:	ExpressionNode
multiplicity:	exactly one

default

An expression indicating the default value of Attributes for which this Domain serves as the type.

type:	ExpressionNode
multiplicity:	zero or one

References

baseType

Identifies a Classifier instance that represents the type upon which the Domain instance is based.

class:	Classifier
defined by:	DomainBaseType::baseType
multiplicity:	zero or one

Constraints

A Domain instance may not be its own base type. [C-1]

2.3.4 Entity

Instances of the Entity class are the primary objects in a ER model. They represent ideas, processes, and things of interest in an application system or tool model.

Superclasses

Class

Contained Elements

Attribute

CandidateKey

PrimaryKey

ForeignKey

NonUniqueKey

2.3.5 ForeignKey

A ForeignKey instance identifies a set of attributes in one Entity instance that uniquely identifies an instance of another Entity containing a matching primary or candidate key value.

Superclasses

KeyRelationship

References

relationshipEnd

Identifies the RelationshipEnd of the Relationship that is implemented by the current ForeignKey instance.

class:	RelationshipEnd
defined by:	ForeignKeyImplements::relationshipEnd
multiplicity:	zero or one
inverse::	RelationshipEnd::foreignKey

2.3.6 Model

Instances of this class represent ER models. Models are collected together by ModelLibrary instances.

Superclasses

Model

Contained Elements

Domain

Entity

Relationship

SubjectArea

2.3.7 ModelLibrary

A collection of ER models and model libraries. Model libraries can be nested using the inherited CWM ElementOwnership association between Namespace and

ModelElement. The same inherited CWM association is used to define ER package Model instances that reside in the current model library.

Superclasses

Subsystem

Contained Elements

Model

ModelLibrary

2.3.8 NonUniqueKey

A NonUniqueKey in the ER model is equivalent to a CWM Index. Values of the keys are not necessarily either unique or required.

Superclasses

Index

2.3.9 PrimaryKey

PrimaryKey instances identify a key that uniquely identifies each instance of an Entity and that is distinguished by the modeler as the Entity's primary key.

Superclasses

CandidateKey

2.3.10 Relationship

ER Relationship instances represent links between Entity instances. Because they derive from AssociationClass, Relationships can have attributes as allowed by some ER model extensions. This derivation also allows Relationship instances to be used as the end points of other Relationship instances.

Superclasses

Association

Contained Elements

RelationshipEnd

2.3.11 RelationshipEnd

The RelationshipEnd class extends CWM's AssociationEnd class to permit the definition of separate delete, update, and insert rules on each end of a Relationship.

An ER model Relationship instance owns two or more RelationshipEnds via an inherited CWM association between the Association and AssociationEnd classes.

Superclasses

AssociationEnd

Attributes

delete

An expression describing the integrity constraint rule for deletes on this RelationshipEnd instance.

type:	ExpressionNode
multiplicity:	Exactly one

update

An expression describing the integrity constraint rule for updates on this RelationshipEnd instance.

type:	ExpressionNode
multiplicity:	exactly one

insert

An expression describing the integrity constraint rule for inserts on this RelationshipEnd instance.

type:	ExpressionNode
multiplicity:	exactly one

References

foreignKey

Identifies a ForeignKey instance that implements this RelationshipEnd.

class:	ForeignKey
defined by:	ForeignKeyImplements::foreignKey
multiplicity:	zero or one
inverse:	ForeignKey::relationshipEnd

2.3.12 SubjectArea

A meaningful subset of the instances in an ER Model instance. Normally, a SubjectArea instance will consist of the Entity and Relationship instances it owns or imports. The inherited CWM ElementOwnership association between ModelElement and Namespace is used to link SubjectArea instances to their owning Model instances.

Superclasses

Package

2.4 Entity Relationship Associations

2.4.1 DomainBaseType

Identifies a Classifier instance that represents the type upon which the Domain instance is based.

Ends

baseType

Identifies the Classifier instance that is the type upon which the Domain is based.

class:	Classifier
multiplicity:	zero or one

domain

Identifies the Domains for which a Classifier instance acts as the base type.

class:	Domain
multiplicity:	zero or more

2.4.2 ForeignKeyImplements *Protected*

Identifies the ForeignKey that implements a particular RelationshipEnd.

Ends

relationshipEnd

Identifies the RelationshipEnd instance that this ForeignKey instance implements.

class:	RelationshipEnd
multiplicity:	zero or one

foreignKey

Identifies the ForeignKey instance that implements the RelationshipEnd instance.

class:	ForeignKey
multiplicity:	zero or one

2.5 OCL Representation of Entity Relationship Constraints

[C-1] A Domain instance may not be its own base type.

context Domain **inv:**

self.baseType <> self

COBOL Data Division 3

3.1 Overview

The concepts and ideas implicit in the definition of the COBOL language's DATA DIVISION were one of the earliest (if not the first) formalizations of the ubiquitous record model. A COBOL program contains much more than just record descriptions. However, because neither CWM nor UML attempt to describe programming languages directly, only the DATA DIVISION is described here. The model presented here is compliant to the COBOL 85 language standard [COBOL].

The primary purpose of the COBOL DATA DIVISION metamodel extension package in CWM is to allow the structure of DATA DIVISIONs to be captured so that their usage of other model elements (such as RecordDefs and Fields) can be modeled. This allows definition of files and databases created by COBOL programs as well as direct support for tools that attempt to track the lineage and determine the impact of proposed changes to COBOL application programs. The metamodel does not, however, provide sufficient structure to support tools that want to capture the structure of a DATA DIVISION source into a CWM repository and then be able to faithfully reproduce the source on demand.

The COBOL DATA DIVISION metamodel extension also serves as an example of the use of the CWM Record metamodel. The CWM Record package is intended as a foundation upon which many record-oriented programming languages can be described. The COBOL Data Division extension package is provided as example demonstrating appropriate usage of CWM and UML classes in modeling the data structure representation parts of this and similar programming language environments.

3.2 Organization of the COBOL Data Division Package

The COBOL Data Division package depends on the following packages:

- org.omg::CWM::ObjectModel::Core
- org.omg::CWM::Foundation::KeysIndexes

- org.omg::CWM::Resource::Record

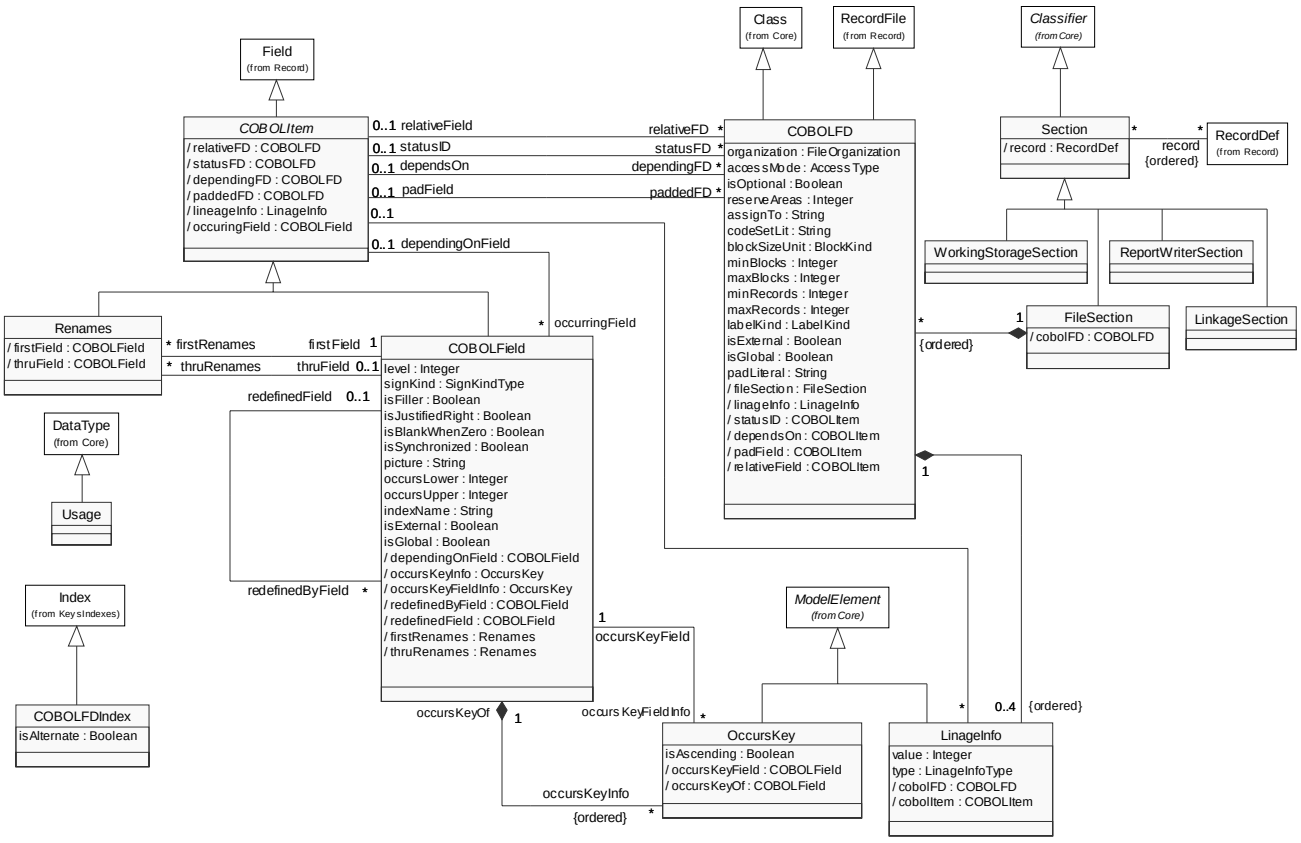

Figure 3-1 COBOLDataDivision Package

3.3 COBOL Data Division Classes

3.3.1 COBOLFD

Represents COBOL FD definitions. FDs describe files that are used in COBOL programs.

The size of COBOLFD records may vary within a range bounded by the contents of the ***minRecords*** and ***maxRecords*** attributes and with a current size given by the field identified by the ***dependsOn*** reference. Two attributes and a reference are used to represent the necessary information. To illustrate the roles they play, the names of the attributes and the reference are substituted into the following COBOL syntax fragment:

RECORD IS VARYING FROM ***minRecords*** TO ***maxRecords*** DEPENDING ON ***dependsOn***

Superclasses

Class

RecordFile

Contained Elements

LinageInfo

Attributes

organization

Contains the physical organization of the file.

type:	FileOrganization (unspecified | indexed | relative | sequential)
multiplicity:	exactly one

accessMode

Contains the access mode of the file.

type:	AccessType (unspecified | dynamic | random | sequential)
multiplicity:	exactly one

isOptional

If True, the file is optional at runtime.

type:	Boolean
multiplicity:	exactly one

reserveAreas

Contains the number of buffer areas reserved for the file.

type: Integer

multiplicity: zero or one

assignTo

Contains the name of the storage medium the file is assigned to.

type: String

multiplicity: exactly one

codeSetLit

Contains the name of the code set.

type: String

multiplicity: exactly one

blockSizeUnit

Contains the unit type for the contents of the minBlocks and maxBlocks fields.

type: BlockKind (records | characters)

multiplicity: exactly one

minBlocks

Contains the minimum number of <units> per block, where <unit> is specified by the ***blockSizeUnit*** attribute.

type: Integer

multiplicity: zero or one

maxBlocks

Contains the maximum number of <units> per block, where <unit> is specified by the ***blockSizeUnit*** attribute.

type: Integer

multiplicity: zero or one

minRecords

Contains the minimum number of characters per record.

type:	Integer
multiplicity:	zero or one

maxRecords

Contains the maximum number of characters per record.

type:	Integer
multiplicity:	zero or one

labelKind

Contains the label kind of the file.

type:	LabelKind (unspecified | standard | omitted)
multiplicity:	exactly one

isExternal

If True, the file is external.

type:	Boolean
multiplicity:	exactly one

isGlobal

If True, the file is global.

type:	Boolean
multiplicity:	exactly one

padLiteral

If not an empty string, contains the pad character. If an empty string, the padField reference may point to a COBOLField instance that contains the pad character.

type:	String
multiplicity:	exactly one

References

fileSection

Identifies the FileSection instances that contain this COBOLFD instance.

class:	FileSection
defined by:	FileSectionFD::fileSection
multiplicity:	exactly one
inverse:	FileSection::cobolFD

linageInfo

Identifies the LinageInfo instances relevant to this COBOLFD instance.

class:	LinageInfo
defined by:	LinageInfoField::linageInfo
multiplicity:	zero to four; ordered
inverse:	LinageInfo::cobolFD

statusID

Identifies the COBOLItem instance containing the status value.

class:	COBOLItem
defined by:	FDStatusID::statusID
multiplicity:	zero or one
inverse:	COBOLItem::statusFD

dependsOn

Identifies the COBOLItem instance that contains the current record size for this COBOLFD instance.

class:	COBOLItem
defined by:	FDDepending::dependsOn
multiplicity:	zero or one
inverse:	COBOLItem::dependingFD

padField

Identifies the COBOLItem instance that contains the pad character.

class:	COBOLItem

defined by:	PaddingField::padField
multiplicity:	zero or one
inverse:	COBOLItem::paddedFD

relativeField

Identifies the COBOLItem instance containing the current relative record offset in the file represented by the COBOLFD instance.

class:	COBOLItem
defined by:	RelativeOffsetField::relativeField
multiplicity:	zero or one
inverse:	COBOLItem::relativeFD

Constraints

The presence of a padding character can be indicated either by a constant (in the padLiteral attribute) or by a reference to another field via the padField reference but not by both. [C-1]

3.3.2 COBOLFDIndex

A COBOLFDIndex instance represents a RECORD KEY or ALTERNATE RECORD KEY for an INDEXED file.

Superclasses

Index

Attributes

isAlternate

If True, this is an alternate index.

type:	Boolean
multiplicity:	exactly one

3.3.3 COBOLField

Represents fields that appear in COBOL record descriptions. COBOLField instances are associated with their owning RecordDef or Group instances via the UML owner/feature association between Feature and Classifier.

The VALUE IS clause for a COBOLField instance is stored in the initialValue attribute inherited from the UML Attribute superclass.

The “occurs-depending” syntax that may be attached to a COBOLField instance is addressed by a collection of attributes (***occursLower*** and ***occursUpper***) and a reference (***dependingOnField***). To illustrate the roles these attributes and references play, their names can be substituted into the following COBOL syntax fragment:

OCCURS ***occursLower*** TO ***occursUpper*** TIMES DEPENDING ON ***dependingOnField***

Superclasses

Field

Contained Elements

OccursKey

Attributes

level

The level number of a COBOLField.

type:	Integer
multiplicity:	exactly one

signKind

The type of sign for the field.

type:	SignKindType (unspecified | leadingSign | trailingSign | leadingSepSign | trailingSepSign)
multiplicity:	exactly one

isFiller

If True, the field is a filler field.

type:	Boolean
multiplicity:	exactly one

isJustifiedRight

If True, the content of the field is right justified.

type:	Boolean
multiplicity:	exactly one

isBlankWhenZero

If True, the field is interpreted as having the numeric value zero when the field contains blanks.

type:	Boolean
multiplicity:	exactly one

isSynchronized

If True, the field is synchronized.

type:	Boolean
multiplicity:	exactly one

picture

Contains the picture specification for the field.

type:	String
multiplicity:	exactly one

occursLower

If the field occurs, contains the lower bound of the number of possible occurrences.

type:	Integer
multiplicity:	zero or one

occursUpper

If the field occurs, contains the upper bound of the number of possible occurrences.

type:	Integer
multiplicity:	zero or one

indexName

A list of strings that are the names obtained from the INDEXED BY clause.

type:	String
multiplicity:	zero or more

isExternal

If True, the field is external.

type:	Boolean
multiplicity:	exactly one

isGlobal

If True, the field is global.

type:	Boolean
multiplicity:	exactly one

References

dependingOnField

Identifies the COBOLField instance that contains the number of occurrences of this field.

class:	COBOLItem

defined by:	OccursDependingOn::dependingOnField
multiplicity:	zero or one
inverse:	COBOLItem::occurringField

occursKeyInfo

Identifies the OccursKey instances describing the fields that make up the “occurs” key for this field.

class:	OccursKey
defined by:	OccursKeyField::occursKeyInfo
multiplicity:	zero or more; ordered
inverse:	OccursKey::occursKeyOf

occursKeyFieldInfo

Identifies the OccursKey instances that describe how this field participates in the “occurs” keys of other fields.

class:	OccursKey
defined by:	OccuringKeyInfo::occursKeyFieldInfo
multiplicity:	zero or more
inverse:	OccursKey::occursKeyField

redefinedByField

Identifies the COBOLField instances that redefine this field.

class:	COBOLField
defined by:	Redefines::redefinedByField
multiplicity:	zero or more
inverse:	COBOLField::redefinedField

redefinedField

Identifies the COBOLField instances that this field redefines.

class:	COBOLField
defined by:	Redefines::redefinedField
multiplicity:	zero or one
inverse:	COBOLField::redefinedByField

firstRenames

Identifies the Renames instances in which this COBOLField instance is the first renamed field.

class:	Renames
defined by:	RenamesFirst::firstRenames
multiplicity:	zero or more
inverse:	Renames::firstField

thruRenames

Identifies the Renames instances in which this COBOLField instance is the last renamed field.

class:	Renames
defined by:	RenamesThru::thruRenames
multiplicity:	zero or more
inverse:	Renames::thruField

Constraints

Level 77 fields must be owned by the Working Storage or the Linkage sections and may not have children. [C-2]

Field level must be 01 to 49, 66, 77 or 88. [C-3]

A COBOLField can only be redefined by fields at the same level. [C-4]

3.3.4 COBOLItem *Abstract*

The COBOLItem class is an abstract metaclass that represents the commonalities shared by both the COBOLField and Renames metaclasses.

Superclasses

Field

References

relativeFD

Identifies the COBOLFD instances for which this COBOLItem instance acts as a relative record offset.

class:	COBOLFD
defined by:	RelativeOffsetField::relativeFD
multiplicity:	zero or more
inverse:	COBOLFD::relativeField

statusFD

Identifies COBOLFD instances for which this COBOLItem acts as the statusID.

class:	COBOLFD
defined by:	FDStatusID::statusFD
multiplicity:	zero or more
inverse:	COBOLFD::statusID

dependingFD

Identifies the COBOLFD instance for which this COBOLItem determines the record size.

class:	COBOLFD
defined by:	FDDepending::dependingFD
multiplicity:	zero or more
inverse:	COBOLFD::dependsOn

paddedFD

Identifies the COBOLFD instances for which this COBOLItem contains the pad character.

class:	COBOLFD
defined by:	PaddingField::paddedFD
multiplicity:	zero or more
inverse:	COBOLFD::padField

linageInfo

Identifies the LinageInfo instances in which this COBOLItem participates.

class:	LinageInfo
defined by:	LinageField::linageInfo
multiplicity:	zero or more
inverse:	LinageInfo::cobolField

occurringField

Identifies the COBOLField instances for which this COBOLItem contains the number of occurrences.

class:	COBOLField
defined by:	OccursDependingOn::occuringField
multiplicity:	zero or more
inverse:	COBOLField::dependingOnField

3.3.5 FileSection

Represents the File section of a COBOL Data Division.

Superclasses

Section

Contained Elements

COBOLFD

References

cobolFD

Associates a COBOL File section with the COBOLFD instances that it contains.

class:	COBOLFD
defined by:	FileSectionFD::cobolFD
multiplicity:	zero or more; ordered
inverse:	COBOLFD::fileSection

Constraints

The RecordDef instances defined within each COBOLFD in a FileSection instance must belong to the FileSection instance. [C-5]

3.3.6 LinageInfo

LinageInfo instances are used to record the individual components of a LINAGE clause for a COBOLFD. A LINAGE clause is used to specify a page layout for a sequential file.

Superclasses

ModelElement

Attributes

value

Contains the value of the LinageInfo. If the value is empty, the cobolField reference must identify a COBOLField instance that contains the value.

type:	Integer
multiplicity:	zero or one

type

Contains the type of the linage information in this LinageInfo instance.

type:	LinageInfoType (linage | linageFooting | linageTop | linageBottom)
multiplicity:	exactly one

References

cobolItem

Identifies the COBOLItem instance that contains the linage information. If this reference is empty, the value attribute must contain the linage information.

class:	COBOLItem
defined by:	LinageField::cobolField
multiplicity:	zero or one
inverse:	COBOLItem::linageInfo

cobolFD

Identifies the COBOLFD instance for which this LinageInfo reference is valid.

class:	COBOLFD
defined by:	LinageInfoField::cobolFD
multiplicity:	exactly one
inverse:	COBOLFD::linageInfo

Constraints

LinageInfo must either have a value or reference a COBOLItem, but not both. [C-6]

3.3.7 LinkageSection

Represents the Linkage section of a COBOL Data Division.

Superclasses

Section

3.3.8 OccursKey

This intersection class identifies the COBOLField instances that are parts of occurs keys and contains attributes relevant to the fields' roles in the occurs key.

Superclasses

ModelElement

Attributes

isAscending

If True, the COBOLField on the occursKeyField end is maintained in an ascending order in the occurs key. If False, the occursKeyField is maintained in descending order.

type:	Boolean
multiplicity:	exactly one

References

occursKeyOf

Identifies the COBOLField instance that owns the occurs key.

class:	COBOLField
defined by:	OccursKeyField::occursKeyOf
multiplicity:	exactly one
inverse:	COBOLField::occursKeyInfo

occursKeyField

Identifies the COBOLField instance that this OccursKey instance represents in the occurs key.

class:	COBOLField
defined by:	OccuringKeyInfo::occursKeyField
multiplicity:	exactly one
inverse:	COBOLField::occursKeyFieldInfo

3.3.9 Renames

Renames instances define alternate identifiers for one or more contiguous COBOLField instances. Although they are not truly COBOL fields, Renames must be ordered in a record along with true COBOL fields. Because they are ObjectModel Features, they can be ordered among COBOL fields via the ClassifierFeature association.

Superclasses

COBOLItem

References

firstField

Identifies the first field that is renamed.

class:	COBOLField
defined by:	RenamesFirst::firstField
multiplicity:	exactly one
inverse:	COBOLField::firstRenames

thruField

Identifies the last field in a range of features that is renamed.

class:	COBOLField
defined by:	RenamesThru::thruField
multiplicity:	zero or one
inverse:	COBOLField::thruRenames

3.3.10 ReportWriterSection

Represents the Report Writer section of a COBOL Data Division. Although a metaclass for this section remains in the standard, the section itself appears to be deprecated.

Superclasses

Section

3.3.11 Section

Instances of Section describe the various sections of a COBOL Data Division. Section instances are use primarily as collection containers for the RecordDefs that a Section may contain.

Superclasses

Classifier

References

record

Identifies the RecordDef instances that describe the structure of 01 level records defined in the section.

class:	RecordDef
defined by:	SectionRecord::record
multiplicity:	zero or more; ordered

3.3.12 Usage

A subclass of UML's DataType class representing valid COBOL usage types. The type attribute that a COBOLField inherits from StructuralFeature should point to one of the instances of Usage defined here.

Defined M1 instances of the Usage class have the names: COMP, DISPLAY, and INDEX. Common extension data types (such as BINARY, PACKEDDECIMAL, COMP-1, COMP-2, COMP-3, etc.) can be easily added by creating addition M1 instances of Usage without the need to change the metamodel change.

Superclasses

DataType

3.3.13 WorkingStorageSection

Represents the Working Storage section of a COBOL Data Division.

Superclasses

Section

3.4 COBOLData Associations

3.4.1 FDDepending *Protected*

Associates COBOLFD instances with the COBOLItem instance that contains the current size of the FD's record.

Ends

dependsOn

Identifies the COBOLItem instance that contains the current size of the COBOLFD instance's records.

class:	COBOLItem
multiplicity:	zero or one

dependingFD

Identifies the COBOLFD instances for which the COBOLItem instance contains the current record size.

class:	COBOLFD
multiplicity:	zero or more

3.4.2 FDStatusID *Protected*

Associates COBOLFD instances with the COBOLItem instance that contains the status ID of the COBOLFD.

Ends

statusID

Identifies the COBOLItem instance that contains the status ID for this COBOLFD instance.

class:	COBOLItem
multiplicity:	zero or one

statusFD

Identifies the COBOLFD instance for which this COBOLItem instance contains status ID information.

class:	COBOLFD
multiplicity:	zero or more

3.4.3 FileSectionFD *Protected*

Associates a COBOL File section with the COBOLFD instances that it contains.

Ends

cobolFD

Identifies the COBOLFD instances that this FileSection instance contains.

class:	COBOLFD
multiplicity:	zero or more; ordered

fileSection

Identifies the FileSection instances that contain this COBOLFD instance.

class:	FileSection
multiplicity:	exactly one
aggregation:	composite

3.4.4 LinageField *Protected*

Associates COBOLField instances with LinageInfo instances that pertain to them.

Ends

cobolItem

Identifies the COBOLItem instance to which the LinageInfo instance applies.

class:	COBOLItem
multiplicity:	zero or one

linageInfo

Identifies the linageInfo instances that reference a COBOLItem instance.

class:	LinageInfo
multiplicity:	zero or more

3.4.5 LinageInfoField *Protected*

Associates a COBOLFD instance with the LinageInfo intersection instances containing information about the FD's linage displays.

Ends

linageInfo

Identifies the LinageInfo instances for this COBOLFD instance.

class:	LinageInfo
multiplicity:	zero to four; ordered

cobolFD

Identifies the COBOLFD instance that owns this LinageInfo instance.

class:	COBOLFD
multiplicity:	exactly one
aggregation:	composite

3.4.6 OccursDependingOn *Protected*

Associates occurring COBOLItem instances with the fields that contain the current number of occurrences.

Ends

dependingOnField

Identifies the COBOLField instance that contains the number of occurrences.

class:	COBOLItem
multiplicity:	zero or one

occurringField

Identifies the field that occurs (i.e., the array).

class:	COBOLField
multiplicity:	zero or more

3.4.7 OccuringKeyInfo *Protected*

Associates COBOLField instances with the OccursKey instances that describe how the fields participate in occurs keys.

Ends

occursKeyField

Identifies the COBOLField instance that participates in this occurs key.

class:	COBOLField
multiplicity:	exactly one

occursKeyFieldInfo

Identifies the OccursKey instances relevant to this COBOLField instance.

class:	OccursKey
multiplicity:	zero or more

3.4.8 OccursKeyField *Protected*

Identifies the usage information showing how subfields of an occurring item are used to order the occurring items.

Ends

occursKeyInfo

Identifies the OccursKey instances relevant for this field.

class:	OccursKey
multiplicity:	zero or more; ordered

occursKeyOf

Identifies the COBOLField instance that owns the occurs key.

class:	COBOLField
multiplicity:	exactly one
aggregation:	composite

3.4.9 PaddingField *Protected*

Associates a COBOLFD instance with a COBOLField instance that contains the FD's pad character.

Ends

paddedFD

Identifies the COBOLFD instance that is padded by this field.

class: COBOLFD

multiplicity: zero or more

padField

Identifies the COBOLField instance that contains the pad character for this COBOLFD.

class: COBOLItem

multiplicity: zero or one

3.4.10 Redefines *Protected*

If this association is non null, the COBOLField instance on the redefinedField end is redefined by the COBOLField instance on the redefinedByField end. For example, in the following COBOL fragment

02 X
02 Y REDEFINES X

X is the redefinedField instance, and Y is the redefinedByField instance.

Ends

redefinedField

Identifies the COBOLField instance that is redefined.

class: COBOLField

multiplicity: zero or one

redefinedByField

Identifies the COBOLField instances that redefine this field.

class: COBOLField

multiplicity: zero or more

3.4.11 RelativeOffsetField *Protected*

Associates COBOLField instances that contains the current relative record offset values with the COBOLFD instances they serve.

Ends

relativeFD

Identifies the COBOLFD instances for which this COBOLField instance acts as a relative record offset.

class: COBOLFD

multiplicity: zero or more

relativeField

Identifies the COBOLField instance containing the current relative record offset in the file represented by the COBOLFD instance.

class: COBOLField

multiplicity: zero or one

3.4.12 RenamesFirst *Protected*

Identifies the COBOLField instance that is the first renamed field.

Ends

firstField

Identifies the COBOLField instance that is the first renamed field.

class: COBOLField

multiplicity: exactly one

firstRenames

Identifies the Renames instances in which this COBOLField instance is the first renamed field.

class: Renames

multiplicity: zero or more

3.4.13 RenamesThru *Protected*

Associates COBOLField instances with the last field in a renamed range of fields.

Ends

thruField

Identifies the COBOLField instance that is the last field in a range of renamed fields.

class:	COBOLField
multiplicity:	zero or one

thruRenames

Identifies the Renames instances in which this COBOLField instance is the last renamed field in a range of renamed fields.

class:	Renames
multiplicity:	zero or more

3.4.14 SectionRecord

Identifies the Section instances in which the RecordDef instance is defined.

Ends

section

Identifies the Section instances in which the RecordDef is used.

class:	Section
multiplicity:	zero or more

record

Identifies the RecordDef instances that are defined in the Section instance.

class:	RecordDef
multiplicity:	zero or more; ordered

3.5 OCL Representation of COBOLData Constraints

[C-1] The presence of a padding character can be indicated either by a constant (in the padLiteral attribute) or by a reference to another field via the padField reference but not by both.

context COBOLFD **inv:**

self.padLiteral <> "" **implies** self.padField->isEmpty

[C-2] Level 77 fields must be owned by the Working Storage or the Linkage sections and may not have children.

context COBOLField **inv:**

self.level = 77 **implies** (self.classifier.oclIsKindOf(WorkingStorageSection) **or** self.classifier.oclIsKindOf(LinkageSection) and self.type.feature->isEmpty

[C-3] Field level must be 01 to 49, 77.

context COBOLField **inv:**

(self.level >= 1 **and** self.level <= 49) **or** self.level = 77

[C-4] A COBOLField can only be redefined by fields at the same level.

context COBOLField **inv:**

self.redefinedByField->NotEmpty **implies** self.level = self.redefinedByField.level

[C-5] The RecordDef instances defined within each COBOLFD in a FileSection instance must belong to the FileSection instance.

context FileSection **inv:**

self.cobolFD.record->exists(p | p = self.record)

[C-6] LinageInfo must either have a value or reference a COBOLItem, but not both.

context LinageInfo **inv:**

self.value->isEmpty **implies** not self.cobolItem->isEmpty

DMS II 4

4.1 Overview

The CWM DMSII extension package contains classes supporting the description of DMS II database schemata and their deployment. DMS II is a database system available on Unisys ClearPath NX servers. DMS II is a non-CODASYL, network model database management system. The DMS II extension package is provided as example demonstrating appropriate usage of CWM classes in modeling this and similar DBMS environments.

Because DMSII database schemas are normally stored in record-based source files written in a data definition language called DASDL, the CWM DMSII extension package contains constructs allowing the declaration-order sequence of the DASDL source file to be preserved in the model. The goal of this is to allow a DASDL source to be stored into the DMSII model and subsequently regenerated from the model by a suitably designed utility program. To achieve this ordering, the DMSII model represents ownership using the CWM ClassifierFeature association which is ordered, rather than the alternate technique using the ElementOwnership association which is unordered. A side-effect of this choice is that any DMSII class that can be ordered must be subclass of the CWM ObjectModel's Feature class; this is the reason for the multiple inheritance required to define the SetStructure, DataSet, Database and Remap classes.

For convenience, utilities may chose to store the text of the DASDL file from which the database was created in a CWM Description instance attached to particular instances of DMSII model classes. The names of DMSII model elements are stored in the name attribute that every DMSII instance inherits from CWM's ModelElement class.

4.2 Organization of the DMSII Package

The DMS II package depends on the following packages:

- org.omg::CWM::ObjectModel::Core

- org.omg::CWM::Foundation::BusinessInformation
- org.omg::CWM::Foundation::Expressions
- org.omg::CWM::Foundation::KeyIndexes
- org.omg::CWM::Foundation::Record

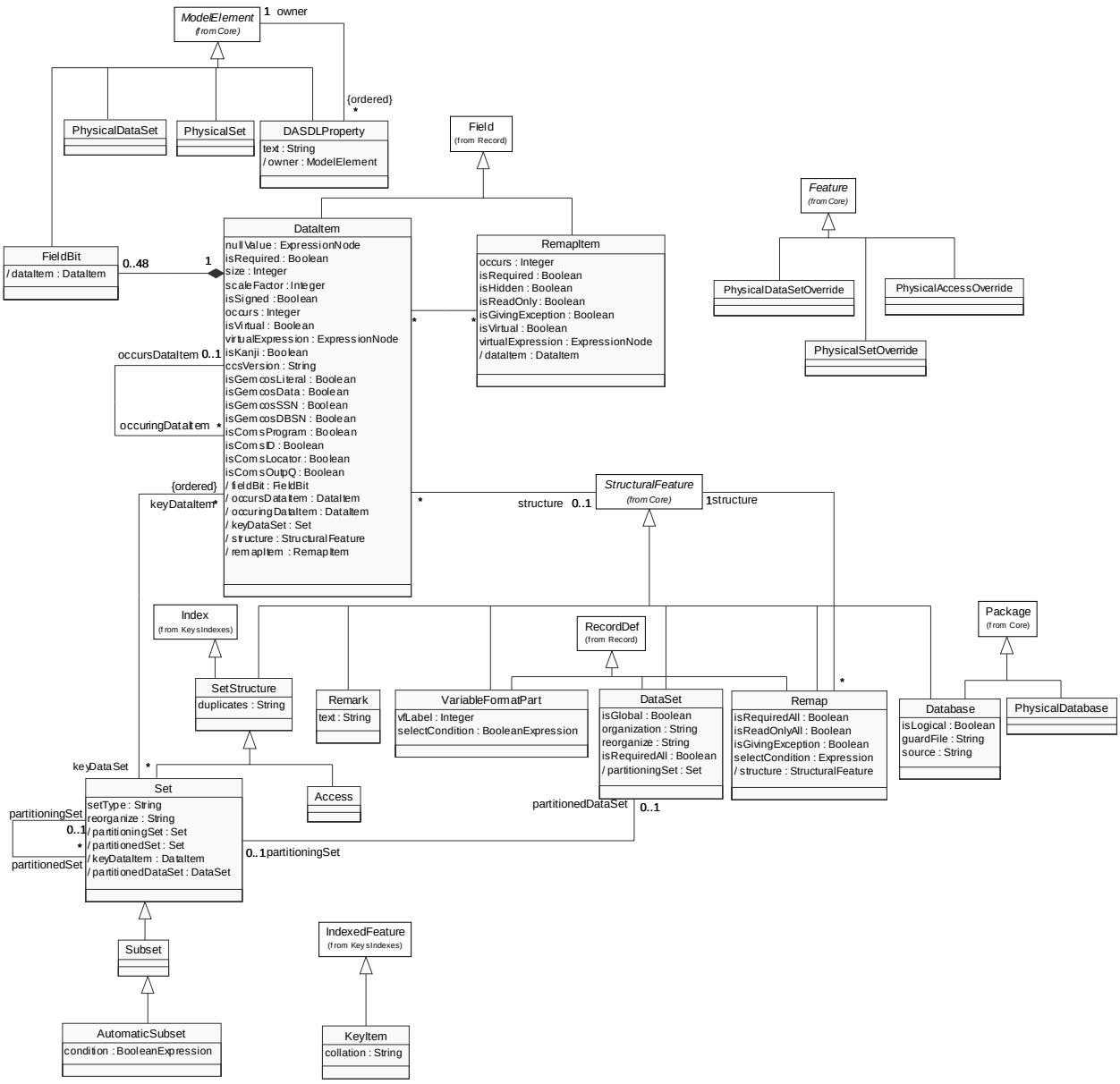

Figure 4-1 DMSII Package

4.3 DMSII Classes

4.3.1 Access

Represents a DMSII Access which is used to specify the physical ordering of records in the DataSet that the Access spans. Unlike a Set instance, there is no separate physical file associated with an Access in a deployed database.

Superclasses

SetStructure

Constraints

An Access must span a DataSet. [C-1]

4.3.2 AutomaticSubset

A Subset that has a membership expression. Records in the spanned DataSet instance are part of the AutomaticSubset instance if the expression in the condition attribute evaluates to True.

Superclasses

Subset

Attributes

condition

Contains the text of an expression that determines membership of the spanned DataSet's records in this AutomaticSubset instance.

type:	BooleanExpression
multiplicity:	exactlyxactly one

4.3.3 DASDLComment

Contains the text of the single DASDL <comment> that nearly every object in a DASDL source may have. These comments differ from Remarks in that there is only one instance per DASDL object, their position in the source is constrained, and they are stored in the database description files. Remarks are not tied to particular database objects, can occur anywhere in a DASDL source, and exist only in the DASDL source itself.

For all DASDLComments, the language attribute contains the string “DASDL”.

Superclasses

Description

4.3.4 DASDLProperty

The DASDL language source file from which DMS II databases are built contains a large number (>100) of options related primarily to the physical characteristics or deployment of DMS II databases and the data and set structures they contain. Generally, these “DASDL properties” are of the form <name> = <string>, where the meaning of the contents of <string> is specific to the property that is being described (i.e., knowing the content of <name>). Also, new DASDL properties are added from time to time. Capturing these DASDL properties as <name>/<string> pairs has several important side-effects, including

- a much simplified DMS II model overall,
- addition of new properties without having to change the model, and
- maintenance of the order (because of the ordered nature of the association to ModelElement) in which the property values were supplied in the DASDL source.

Note that allowing a DASDL remark (i.e., a %-comment) to be a DASDL property in this case allows preservation of the order remarks with respect to other DASDL properties.

Superclasses

ModelElement

Attributes

text

Contains the text of the DASDL property. The precise content of the string is dependent upon the name of the DASDL property defined in the name attribute inherited from ModelElement.

type:	String
multiplicity:	exactly one

References

owner

Identifies the ModelElement for which this DASDLProperty instance is relevant.

class:	ModelElement
defined by:	DASDLPropertyOwner::owner
multiplicity:	exactly one

Constraints

The types of ModelElements that may own DASDLProperties is limited to DataSet, SetStructure, Database, PhysicalDatabase, PhysicalDataset, PhysicalSet, PhysicalDatasetOverride, PhysicalSetOverride, and PhysicalAccessOverride[C-2].

4.3.5 Database

For a given DASDL source, there can be at most one Database instance with ***isLogical*** = False (representing an independent, free standing database) and zero or more with ***isLogical*** = True (each representing a Logical Database declared within the physical database). A logical database can be owned by at most one database.

Superclasses

Subsystem

StructuralFeature

Attributes

isLogical

If True, this Database instance is a logical database.

type:	Boolean
multiplicity:	exactly one

guardFile

Contains the name of the database guard file that contains access control information for the database.

type:	String
multiplicity:	exactly one

source

Contains the text of the DASDL source from which the database was created.

type:	String
multiplicity:	exactly one

Constraints

An independent database may not be owned. [C-11]

A logical database must be owned by an independent database. [C-12]

A database can own SetStructure, DataSet, Remark, and Database, Remap, PhysicalDataSetOverride, PhysicalSeOverride, and PhyiscalAccessOverride instances. [C-13]

4.3.6 DataItem

Instances of DataItem represent the individual data fields within a DataSet. The Group class in the CWM Foundation's DataTypes package is also available for constructing collections of fields in a DataSet.

The interpretation of the contents of some attributes of a DataItem instance are dependent upon the DataItem's type. For example, the size attribute represents the maximum number of characters in ALPHA and KANJI items, the number of digits of precision in a NUMERIC or REAL items, and the number of bits in a FIELD item. Refer to the definition of individual attributes for specifics.

Superclasses

Field

Contained Elements

FieldBit

Attributes

nullValue

Identifies a value of a data item that is treated as representing a null value.

type:	ExpressionNode
multiplicity:	exactly one

isRequired

If True, the data item must have a value when the corresponding Dataset record is stored.

type: Boolean

multiplicity: exactly one

size

Contains the declared size of a data type. The precise meaning of the attribute depends on the type of the DMS II data type being declared.

type: Integer

multiplicity: exactly one

scaleFactor

Contains the <scale factor> value for DMS II data types.

type: Integer

multiplicity: exactly one

isSigned

Contains the state of the signed indication (“S”) for REAL and NUMERIC data types. Not relevant for other data types.

type: Boolean

multiplicity: exactly one

occurs

Indicates the number of times the data item occurs in the DataSet record. The occurs attribute is optional, existing only for data items that have an OCCURS clause in their definition.

type: Integer

multiplicity: exactly one

isVirtual

If True, the DataItem instance is calculated when accessed using the expression stored in the virtualExpression attribute.

type: Boolean

multiplicity: exactly one

virtualExpression

The expression used to calculate the value of a virtual DataItem.

type:	ExpressionNode
multiplicity:	exactly one

isKanji

True if the USAGE=KANJI clause was used. Otherwise USAGE=EBCDIC is assumed. Relevant only for ALPHA data items.

type:	Boolean
multiplicity:	exactly one

ccsVersion

Identifies the CCSVersion specification of a data item.

type:	String
multiplicity:	exactly one

isGemcosLiteral

If True, the DataItem instance was defined with the GEMCOS LITERAL clause.

type:	Boolean
multiplicity:	exactly one

isGemcosData

If True, the DataItem instance was defined with the GEMCOS DATA clause.

type:	Boolean
multiplicity:	exactly one

isGemcosSSN

If True, the DataItem instance was defined with the GEMCOS SSN clause.

type:	Boolean
multiplicity:	exactly one

isGemcosDBSN

If True, the DataItem instance was defined with the GEMCOS DBSN clause.

type: Boolean

multiplicity: exactly one

isComsProgram

If True, the DataItem instance was defined with the COMS PROGRAM clause.

type: Boolean

multiplicity: exactly one

isComsID

If True, the DataItem instance was defined with the COMS ID clause.

type: Boolean

multiplicity: exactly one

isComsLocator

If True, the DataItem instance was defined with the COMS LOCATOR clause.

type: Boolean

multiplicity: exactly one

isComsOutputQ

If True, the DataItem instance was defined with the COMS OUTPQ clause.

type: Boolean

multiplicity: exactly one

References

fieldBit

Identifies the FieldBit instance owned by the DataItem instance.

class: FieldBit

defined by: FieldBits::fieldBit

multiplicity: zero to forty eight

inverse: FieldBit::dataItem

keyDataSet

Identifies the Set instances in whose key data parts this DataItem instance participates.

class:	Set
defined by:	KeyDataItem::keyDataSet
multiplicity:	zero or more
inverse:	Set::keyDataItem

occursDataItem

Identifies the DataItem instance that contains the number of occurrences for this occurring DataItem instance.

class:	DataItem
defined by:	OccursDepending::occursDataItem
multiplicity:	zero or one
inverse:	DataItem::occuringDataItem

occuringDataItem

Identifies the DataItem instances whose number of occurrences is defined by the value of this DataItem instance.

class:	DataItem
defined by:	OccursDepending::occuringDataItem
multiplicity:	zero or more
inverse:	DataItem::occursDataItem

remapItem

Identifies the RemapItem instances that remap this DataItem instance.

class:	RemapItem
defined by:	RemapItems::remapItem
multiplicity:	zero or one
inverse:	RemapItem::dataItem

structure

Identifies the DataSet or SetStructure instance to which this DataItem instance refers.

class:	StructuralFeature
defined by:	DataItemStructure::structure
multiplicity:	zero or one

Constraints

A DataItem that owns FieldBit instances must have a data type of FIELD. [C-3]

A reference to a StructuralFeature must refer to a DataSet or a SetStructure instance. [C-4]

A DataItem may not be its own occursDataItem. [C-5]

A DataItem may not be its own occuringDataItem. [C-6]

4.3.7 DataSet

A DataSet is the primary container of data records in a DMS II database. A DataSet instance that is owned by another DataSet instance is embedded in its owner.

If the isGlobal attribute is True, the DataSet instance is the global data set -- a special data set with a single instance that contains database global data items.

Superclasses

RecordDef

StructuralFeature

Contained Elements

None

Attributes

isGlobal

If ***isGlobal*** = True, the DataSet instance represents the database's global data record. There can be at most one DataSet instance with ***isGlobal*** = True for a given database, but multiple with ***isGlobal*** = False.

type:	Boolean
multiplicity:	exactly one

organization

Identifies the structural organization of the DataSet.

type:	String
multiplicity:	exactly one

reorganize

Contains the reorganization clause attached the DataSet instance.

type:	String
multiplicity:	exactly one

isRequiredAll

If True, the REQUIRED ALL clause was specified in the DASDL for this DataSet instance.

type:	Boolean
multiplicity:	exactly one

References

partitioningSet

Identifies the Set instance that serves as the partitioning set for this DataSet instance.

class:	Set
defined by:	DataSetPartitionSet::partitioningSet
multiplicity:	zero or one
inverse:	Set::partitionedDataSet

Constraints

A DataSet may have one of the following organizations. [C-7]

The reorganize attribute, if present, must be one of the allowed values from the DASDL manual. [C-8]

The partitioningSet, if present, must span the DataSet. [C-9]

If the DataSet has VariableFormatParts, it must also have an attribute of the type “RECORD TYPE”. [C-10]

4.3.8 FieldBit

FieldBit instances name the individual bits in a DMS II field data item.

Superclasses

ModelElement

References

dataItem

Identifies the DataItem instance that owns this FieldBit instance.

class:	DataItem
defined by:	FieldBits::dataItem
multiplicity:	exactly one
inverse:	DataItem::fieldBit

4.3.9 KeyItem

KeyItem instances correspond to DASDL's <key item> construct. Every Key instance has the inherited attribute ***isSorted*** = True and the inherited ***isAscending*** attribute set as indicated in the DASDL <key item>. By default, ***isAscending*** = True.

Superclasses

IndexedFeature

Attributes

collation

Identifies the value of the collation clause specified for a KeyItem instance.

type:	String
multiplicity:	exactly one

Constraints

The collation clause, if present, must be one of the allowed values from the DASDL manual. [C-14]

4.3.10 PhysicalAccessOverride

Collects together DASDLProperty instances associate with a physical access specification in the DASDL source.

Superclasses

Feature

Constraints

PhysicalAccessOverride instances must be owned by an Access instance. [C-15]

4.3.11 PhysicalDatabase

Instances represent deployed physical DMS II databases.

The INITIALIZE and MODEL statements are directives to the DASDL compiler and do not need to be modeled here. Rather, they cause specific actions to happen within a CWM repository. INITIALIZE causes the creation of the first PhysicalDatabase instance; MODEL causes the creation of additional PhysicalDatabase instances after the first.

Superclasses

Package

Constraints

A PhysicalDatabase instance must be owned by a Database instance. [C-16]

4.3.12 PhysicalDataSet

Identifies a physical deployment of a DMS II DataSet.

Superclasses

ModelElement

Constraints

A PhysicalDataSet instance must be owned by a DataSet instance. [C-17]

4.3.13 PhysicalDataSetOverride

Collects together DASDLProperty instances associate with a physical data set specification in the DASDL source.

Superclasses

Feature

Constraints

PhysicalDataSetOverride instances must be owned by a DataSet instance. [C-18]

4.3.14 PhysicalSet

Identifies a physical deployment of a DMS II Set.

Superclasses

ModelElement

Constraints

A PhysicalSet instance must be owned by a Set instance. [C-19]

4.3.15 PhysicalSetOverride

Collects together DASDLProperty instances associate with a physical set specification in the DASDL source.

Superclasses

Feature

Constraints

PhysicalSetOverride instances must be owned by a Set instance. [C-20]

4.3.16 Remap

Contains information identifying a Remap of a DMSII DataSet or Set. The features of a Remap instance must be RemapItem instances.

Superclasses

RecordDef

StructuralFeature

Contained Elements

RemapItem

Attributes

isRequiredAll

If True, all items in the remap are required to be non-null.

type:	Boolean
multiplicity:	exactly one

isReadOnlyAll

If True, the READONLY ALL clause was specified.

type:	Boolean
multiplicity:	exactly one

isGivingException

The ***isGivingException*** boolean is meaningful only if ***isReadOnlyAll*** = True. If the ***isGivingException*** boolean is absent, no exception clause was specified. If it is present, False indicates that the NO EXCEPTION clause was specified whereas True indicates the GIVING EXCEPTION clause.

type:	Boolean
multiplicity:	exactly one

selectCondition

Contains the expression specified in a remap's SELECT clause.

type:	ExpressionNode
multiplicity:	exactly one

References

structure

Identifies the StructuralFeature instance that represents the remapped structure.

class:	StructuralFeature
defined by:	RemappedItem::structure
multiplicity:	exactly one

Constraints

The features of a Remap must be RemapItem instances. [C-21]

Remap instances may remap only DataSet and Set instances. [C-22]

The GIVING EXCEPTION clause is valid only if READONLY ALL was specified. [C-23]

4.3.17 RemapItem

Maps a Remap instance's field to its source, which may be some DataItem or an expression.

The name attribute of a RemapItem instance defaults to the name attribute of the associated DataItem instance. If changed in the Remap definition by a "<identifier> =" clause, the name attribute of the Remap instance is simply set to <identifier>.

The RemapItem instance's initial value is stored in the ***initialValue*** attribute inherited from the CWM ObjectModel's Attribute class.

Superclasses

Feature

Attributes

occurs

If specified, overrides the occurs attribute of the associated DataItem instance.

type: Integer

multiplicity: exactly one

isRequired

If True, overrides the isRequired attribute of the associated DataItem instance.

type: Boolean

multiplicity: exactly one

isHidden

If True, the corresponding DataItem is not visible to the user of the Remap.

type: Boolean

multiplicity: exactly one

isReadOnly

If True, the RemapItem is readonly.

type: Boolean

multiplicity: exactly one

isGivingException

The ***isGivingException*** boolean is meaningful only if ***isReadOnly*** = True. If the ***isGivingException*** boolean is absent, no exception clause was specified. If it is present, False indicates that the NO EXCEPTION clause was specified whereas True indicates the GIVING EXCEPTION clause.

type:	Boolean
multiplicity:	exactly one

isVirtual

If True, the RemapItem instance is calculated when accessed using the expression stored in the virtualExpression attribute.

type:	Boolean
multiplicity:	exactly one

virtualExpression

The expression used to calculate the value of a virtual RemapItem.

type:	ExpressionNode
multiplicity:	exactly one

References

dataItem

Identifies the DataItem instance that this RemapItem instance remaps.

class:	DataItem
defined by:	RemapItems::dataItem
multiplicity:	zero or more
inverse:	DataItem::remapItem

Constraints

[C-24] The GIVING EXCEPTION clause is valid only if READONLY was specified.

4.3.18 Remark

Contains the text of a % comment embedded anywhere within a DASDL source (except at places where the Comment class should be used). Making Remarks a

subtype of StructuralFeature allows their location and order within a DASDL source to be preserved.

Superclasses

StructuralFeature

Attributes

text

Contains the text of the Remark.

type:	String
multiplicity:	exactly one

4.3.19 Set

Represents a DMS II Set that spans some DataSet. Sets are represented by a physical file in a deployed DMSII database.

Superclasses

SetStructure

Attributes

setType

Contains the set organization for this Set instance.

type:	String
multiplicity:	exactly one

reorganize

Contains the content of the reorganization clause, if any, that was specified for the Set instance.

type:	String
multiplicity:	exactly one

References

keyDataItem

Identifies the DataItem instances that participate in the Set instance's key data.

class:	DataItem
defined by:	KeyDataItem::keyDataItem
multiplicity:	zero or more; ordered
inverse:	DataItem::keyDataSet

partitionedDataSet

Identifies the DataItem instances that make up this Set instance's key data.

class:	DataSet
defined by:	DataSetPartitionSet::partitionedDataSet
multiplicity:	exactly one
inverse:	DataSet::partitioningSet

partitionedSet

Identifies the Set instances partitioned by this Set instance.

class:	Set
defined by:	SetPartitionSet::partitionedSet
multiplicity:	exactly one
inverse:	Set::partitioningSet

partitioningSet

Identifies the Set instance that act as a partitioning set for this Set instance.

class:	Set
defined by:	SetPartitionSet::partitioningSet
multiplicity:	zero or more
inverse:	Set::partitionedSet

Constraints

The setType attribute must contain one of the allowed values for set organization from DASDL. [C-25]

The reorganize clause, if present, must be one of the allowed values from the DASDL manual. [C-26]

The items in the Set's key data must be owned by the DataSet that the Set spans. [C-27]

A Set may not partition itself. [C-28]

A Set may not be partitioned by itself. [C-29]

4.3.20 SetStructure

SetStructure instances represent access paths in DMS II. SetStructures are connected to the DataSet instances that they span via the ElementOwnership association inherited from the Index class in the CWM Foundation's KeysIndexes package.

Superclasses

Index

StructuralFeature

Attributes

duplicates

Indicates the duplicates clause associated with the SetStructure instance.

type:	String
multiplicity:	exactly one

Constraints

A SetStructure must span one and only one DataSet instance. [C-30]

Value of the duplicates attribute must be one of the allowed values from the DASDL manual. [C-31]

4.3.21 Subset

Represents a subset of a DataSet instance. Because Subset instances have no membership expression, they are equivalent to DMS II's notion of a "manual" subset.

Superclasses

Set

4.3.22 VariableFormatPart

Represents the VariableFormatParts that can be associated with a DataSet.

Superclasses

StructuralFeature

Attributes

vfLabel

Contains the value of <integer label> identifying this variable format part. <integer label>s are the values of the DMS II data item in the fixed part of the owning DataSet instance with the type RECORD TYPE.

type:	Integer
multiplicity:	exactly one

selectCondition

A boolean expression determining which records appear in the Remap.

type:	BooleanExpression
multiplicity:	exactly one

4.4 DMSII Associations

4.4.1 DASDLPropertyOwner

Associates DASDLProperties with the model elements that own them. The ordered attribute preserves the order in which individual DASDLProperty instances were found in the DASDL source file.

Ends

owner

Identifies the owning ModelElement.

class:	ModelElement
multiplicity:	exactly one

dasdlProperty

Identifies the DASDLProperties that apply to a ModelElement.

class:	DASDLProperty
multiplicity:	zero or more; ordered

4.4.2 DataItemStructure

Links DataItems to StructuralFeatures they may reference. Used to represent the Set or DataSet referenced by various link-type dataItems and to identify the Set or DataSet to which POPULATION dataItems apply.

Ends

structure

Identifies the feature referenced by the data item.

class:	StructuralFeature
multiplicity:	zero or one

dataItem

Identifies the dataItems which a Set or DataSet instance references.

class:	DataItem
multiplicity:	zero or more

4.4.3 DataSetPartitionSet *Protected*

In DASDL, the partitioning set is specified as a physical data set option. However, it is more correctly modeled as a relationship between a DataSet instance and a Set instance.

Ends

partitioningSet

Identifies the partitioning set for this DataItem instance.

class:	Set
multiplicity:	zero or one

partitionedDataSet

Identifies the DataSet instances which the Set instance partitions.

class:	DataSet
multiplicity:	zero or one

4.4.4 FieldBits *Protected*

Associates a BIT data item with the labels for it individual bits.

Ends

dataItem

Identifies the FIELD data item for which a FieldBit is relevant.

class:	DataItem
multiplicity:	exactly one
aggregation:	composite

fieldBit

Identifies the FieldBits for a dataItem whose type is BIT.

class:	FieldBit
multiplicity:	zero to forty eight

4.4.5 KeyDataItem *Protected*

Indicates the DataItem instances that participate in this set's <key data> clause. Note that the referenced DataItem instances must be owned by the DataSet instance that the Set spans.

Ends

keyDataItem

Identifies the DataItem instances that participate in the Set instance's key data.

class:	DataItem
multiplicity:	zero or more; ordered

keyDataSet

Identifies the Set instances in which this DataItem instance participates in the Set's key data.

class:	Set
multiplicity:	zero or more

4.4.6 OccursDepending *Protected*

Identifies the data item that another data item depends upon for its number of occurrences. The occuringDataItem end specifies the DataItem that occurs (i.e., the “array” itself) whereas the occursDataItem end specifies the DataItem whose value is the number of occurrences (i.e., number of elements in the “array”). Observe that this works for BOTH scalar data items and group data items!

Ends

occuringDataItem

Identifies the occurring DataItem (i.e., the array).

class:	DataItem
multiplicity:	zero or more

occursDataItem

Identifies the DataItem that contains the number of elements in the array.

class:	DataItem
multiplicity:	zero or one

4.4.7 RemapItems *Protected*

Identifies the dataItem that a remapItem remaps.

Ends

dataItem

Identifies the dataItem that is remapped.

class:	DataItem
multiplicity:	zero or one

remapItem

Identifies the RemapItem instances that remap this data item.

class: RemapItem

multiplicity: zero or more

4.4.8 RemappedStructure

Identifies the structure which a Remap instance remaps.

Ends

structure

Identifies the structure that is remapped by the Remap instance.

class: StructuralFeature

multiplicity: exactly one

remap

Identifies the Remap instances that remap this structure.

class: RemapItem

multiplicity: zero or more

4.4.9 SetPartitionSet *Protected*

Associates a partitioned Set with its partitioning Set.

Ends

partitionedSet

Identifies the partitioned set.

class: Set

multiplicity: zero or more

partitioningSet

Identifies the partitioning set.

class:	Set
multiplicity:	zero or one

4.5 OCL Representation of DMSII Constraints

[C-1] An Access must span a DataSet.

context Access **inv:**

self.spannedClass->size = 1 **and** self.spannedClass.oclIsKindOf(DataSet)

[C-2] The types of ModelElements that may own DASDLProperties is limited by the following OCL.

context DASDLProperty **inv:**

self.owner.oclIsKindOf(DataSet) **or** self.owner.oclIsKindOf(SetStructure) or self.owner.oclIsKindOf(Database) **or** self.owner.oclIsKindOf(PhysicalDatabase) or self.owner.oclIsKindOf(PhysicalDataset) **or** self.owner.oclIsKindOf(PhysicalSet) or self.owner.oclIsKindOf(PhysicalDatasetOverride) **or** self.owner.oclIsKindOf(PhysicalSetOverride) **or** self.owner.oclIsKindOf(PhysicalAccessOverride)

[C-3] An independent database may not be owned.

context Database **inv:**

not self.isLogical **implies** self.classifier->isEmpty

[C-4] A logical database must be owned by an independent database.

context Database **inv:**

self.isLogical **implies** (self.classifier->size = 1 **and** self.classifier.oclIsTypeOf(Database) **and** not self.classifier.isLogical)

[C-5] A database can own SetStructure, DataSet, Remark, and Database, Remap, PhysicalDataSetOverride, PhysicalSetOverride, and PhysicalAccessOverride instances.

context Database **inv:**

self.feature->forAll(x | x.oclIsTypeOf(SetStructure) **or** x.oclIsTypeOf(DataSet) **or** x.oclIsTypeOf(Remark) **or** x.oclIsTypeOf(Database) **or** x.oclIsTypeOf(PhysicalDataSetOverride) **or** x.oclIsTypeOf(Remap) **or** x.oclIsTypeOf(PhysicalSetOverride) **or** x.oclIsTypeOf(PhysicalAccessOverride))

[C-6] A DataItem that owns FieldBit instances must have a data type of FIELD.

context DataItem **inv:**

self.fieldBit.notEmpty **implies** self.type.name = “FIELD”

[C-7] A reference to a StructuralFeature must refer to a DataSet or a SetStructure instance.

context DataItem **inv:**

self.structure.notEmpty **implies** self.structure.type.oclIsKindOf(DataSet) **or** self.structure.type.oclIsKindOf(SetStructure)

[C-8] A DataItem may not be its own occursDataItem.

context DataItem **inv:**

self.occursDataItem <> self

[C-9]A DataItem may not be its own occuringDataItem.

context DataItem **inv:**

self.occuringDataItem <> self

[C-10] A DataSet may have one of the following organizations.

context DataItem **inv:**

self.organization = “COMPACT” **or** self.organization = “DIRECT” **or** self.organization = “ORDERED” **or** self.organization = “RANDOM” **or** self.organization = “RESTART” **or** self.organization = “STANDARD” **or** self.organization = “UNORDERED”

[C-11] The reorganize attribute, if present, must be one of the allowed values from the DASDL manual.

context DataSet **inv:**

self.reorganize <> “” **implies** self.reorganize = “ITEMS SAME” **or** self.reorganize = “ITEMS CHANGED”

[C-12] The partitioningSet, if present, must span the DataSet.

context DataSet **inv:**

self.partitioningSet->size = 1 **implies** self.partitioningSet.namespace = self

[C-13] If the DataSet has VariableFormatParts, it must also have an attribute of the type “RECORD TYPE”.

context DataSet **inv:**

self.ownedElement->exists(oclIsKindOf(VariableFormatPart)) **implies** self.feature.oclAsType(StructuralFeature)->exists(type.name = “RECORD TYPE”)

[C-14] The collation clause, if present, must be one of the allowed values from the DASDL manual.

context KeyItem **inv:**

self.collation = “BINARY” **or** self.collation = “EQUIVALENT” **or** self.collation = “LOGICAL”

[C-15] PhysicalAccessOverride instances must be owned by an Access instance.

context PhysicalAccessOverride **inv:**

self.namespace.oclIsKindOf(Access)

[C-16] A PhysicalDatabase instance must be owned by a Database instance.

context PhysicalDatabase **inv:**

self.namespace->size = 1 **and** self.namespace.oclIsKindOf(Database)

[C-17] A PhysicalDataSet instance must be owned by a DataSet instance.

context PhysicalDataSet **inv:**

self.namespace->size = 1 **and** self.namespace.oclIsKindOf(DataSet)

[C-18] A PhysicalDataSetOverride instance must be owned by a DataSet instance.

context PhysicalDataSetOverride **inv:**

self.namespace.oclIsKindOf(DataSet)

[C-19] A PhysicalSet instance must be owned by a Set instance.

context PhysicalSet **inv:**

self.namespace->size = 1 **and** self.namespace.oclIsKindOf(Set)

[C-20] PhysicalSetOverride instances must be owned by a Set instance.

context PhysicalSetOverride **inv:**

self.namespace.oclIsKindOf(Set)

[C-21] The features of a Remap must be RemapItem instances.

context Remap **inv:**

self.feature.oclIsKindOf(RemapItem)

[C-22] Remap instances may remap only DataSet and Set instances.

context Remap **inv:**

self.structure.oclIsKindOf(DataSet) **or** self.structure.oclIsKindOf(Set)

[C-23] The GIVING EXCEPTION clause is valid only if READONLY ALL was specified.

context Remap **inv:**

self.isGivingException->notEmpty **implies** self.isReadOnlyAll

[C-24] The GIVING EXCEPTION clause is valid only if READONLY was specified.

context RemapItem **inv:**

self.isGivingException->notEmpty **implies** self.isReadOnly

[C-25] The setType attribute must contain one of the allowed values for set organization from DASDL.

context Set **inv:**

self.setType = “BITVECTOR” **or** self.setType = “UNORDERED LIST” **or** self.setType = “INDEX RANDOM” **or** self.setType = “INDEX SEQUENTIAL” **or** self.setType = “ORDERED LIST”

[C-26] The reorganize clause, if present, must be one of the allowed values from the DASDL manual.

context Set **inv:**

self.reorganize <> “” **implies** self.reorganize = ”KEY CHANGED” **or** self.reorganize = ”KEY SAME”

[C-27] The items in the Set's key data must be owned by the DataSet that the Set spans.

context Set **inv:**

self.keyDataItem->forAll(self.keyDataItem.namespace = self.spannedClass)

[C-28] A Set may not partition itself.

context Set **inv:**

self.partitionedSet <> self

[C-29] A Set may not be partitioned by itself.

context Set **inv:**

self.partitioningSet <> self

[C-30] A SetStructure must span one and only one DataSet instance.

context SetStructure **inv:**

self.spannedClass->size = 1 **and** self.spannedClass.oclIsKindOf(DataSet)

[C-31] Value of the duplicates attribute must be one of the allowed values from the DASDL manual.

context SetStructure **inv:**

self.duplicates = "DUPLICATES" **or** self.duplicates = "DUPLICATES FIRST" **or** self.duplicates = "DUPLICATES LAST" **or** self.duplicates = "NO DUPLICATES" **or** self.duplicates = "NO DUPLICATES KEY CHANGE OK"

IMS 5

5.1 Overview

This package contains a model for IMS database definitions that is an extension of the Record package. This package also uses classes found in the ObjectModel Core package.

The fundamental objects in IMS databases are DBD (Data Base Definition), PCB (Process Contol Block) and PSB (Program Specification Block).

PSBs are the connection between an IMS system and application programs. PSBs contain PCBs which come in three varieties:

- TP (Teleprocessing) PCBs describe a connection to a terminal
- GSAM PCBs connect a PSB to a input or output file.
- DB PCBs connect a PSB to the data defined by a DBD.

DBDs describe the organization of data and the pathways by which an application program can retrieve or store Records. A Record within a DBD is called a Segment. Segments are connected by parent-child relationships to create the information hierarchy.

A Segment can be fully described through the Fields contained within it. However, it is also valid for the Segments within a DBD to contain only a single key field. In this case, the detailed layout of information within Records is described by data structures used by the application program.

Most Data Warehouse applications are concerned only with Segments and Fields. This model contains classes to cover the rest of IMS to support potential tools that might export more of the IMS structure.

5.2 Organization of the IMS Package

The IMS package depends on the following packages:

- org.omg::CWM::ObjectModel::Core
- org.omg::CWM::Resource::Record

The DBD part of the model is shown in Figure 5-1. The PSB portion is shown in Figure 5-2. The way IMS classes inherit from Record classes and from ObjectModel classes is shown in Figure 5-3.

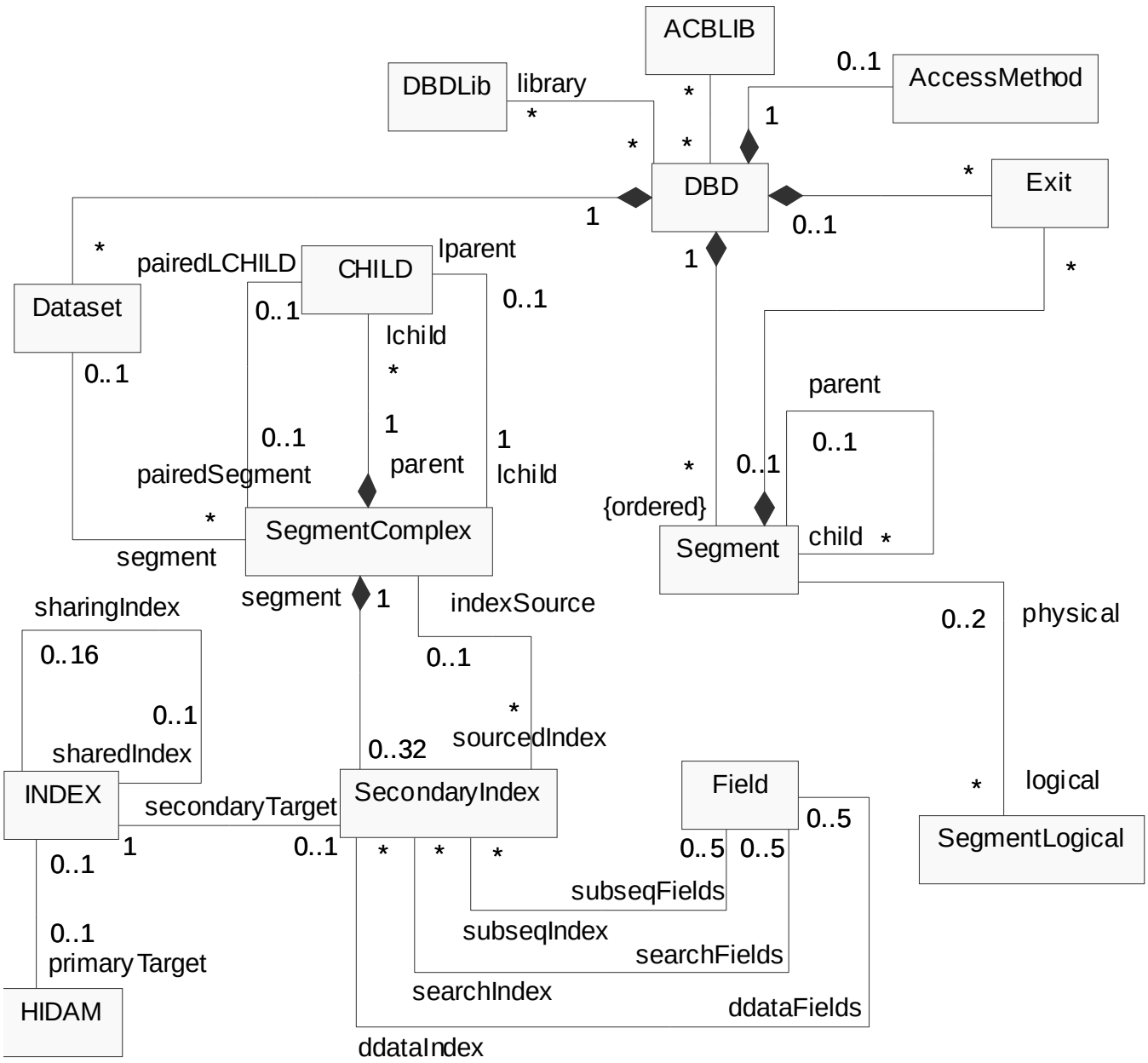

Figure 5-1 IMS Package - DBD

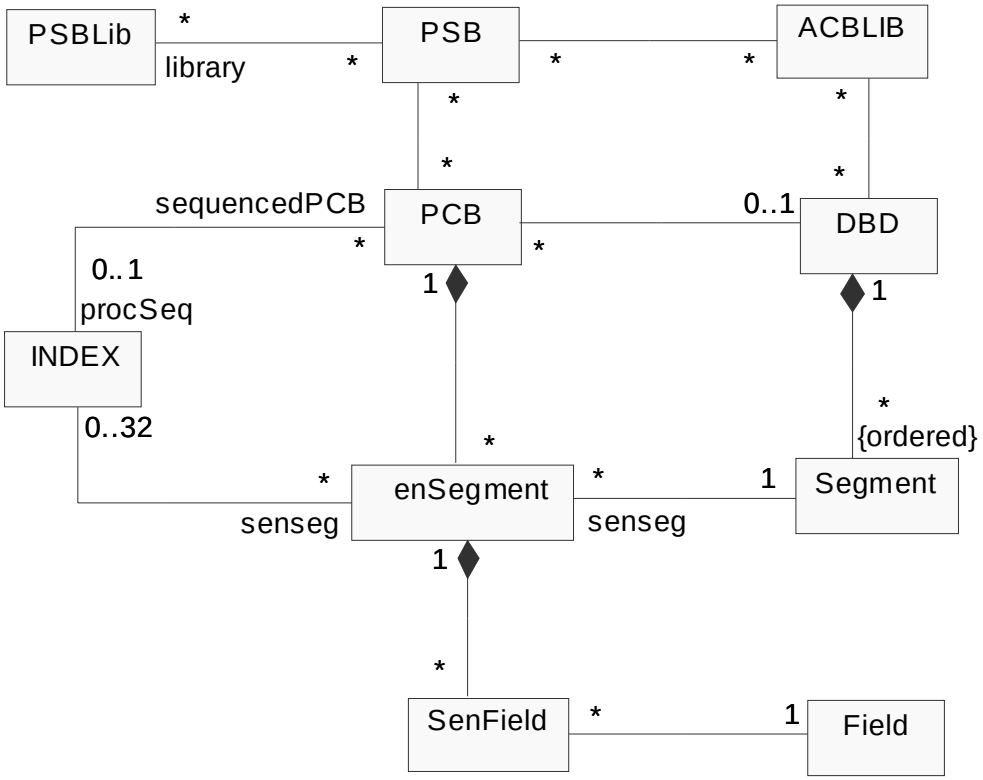

Figure 5-2 IMS Package - PSB and PCB

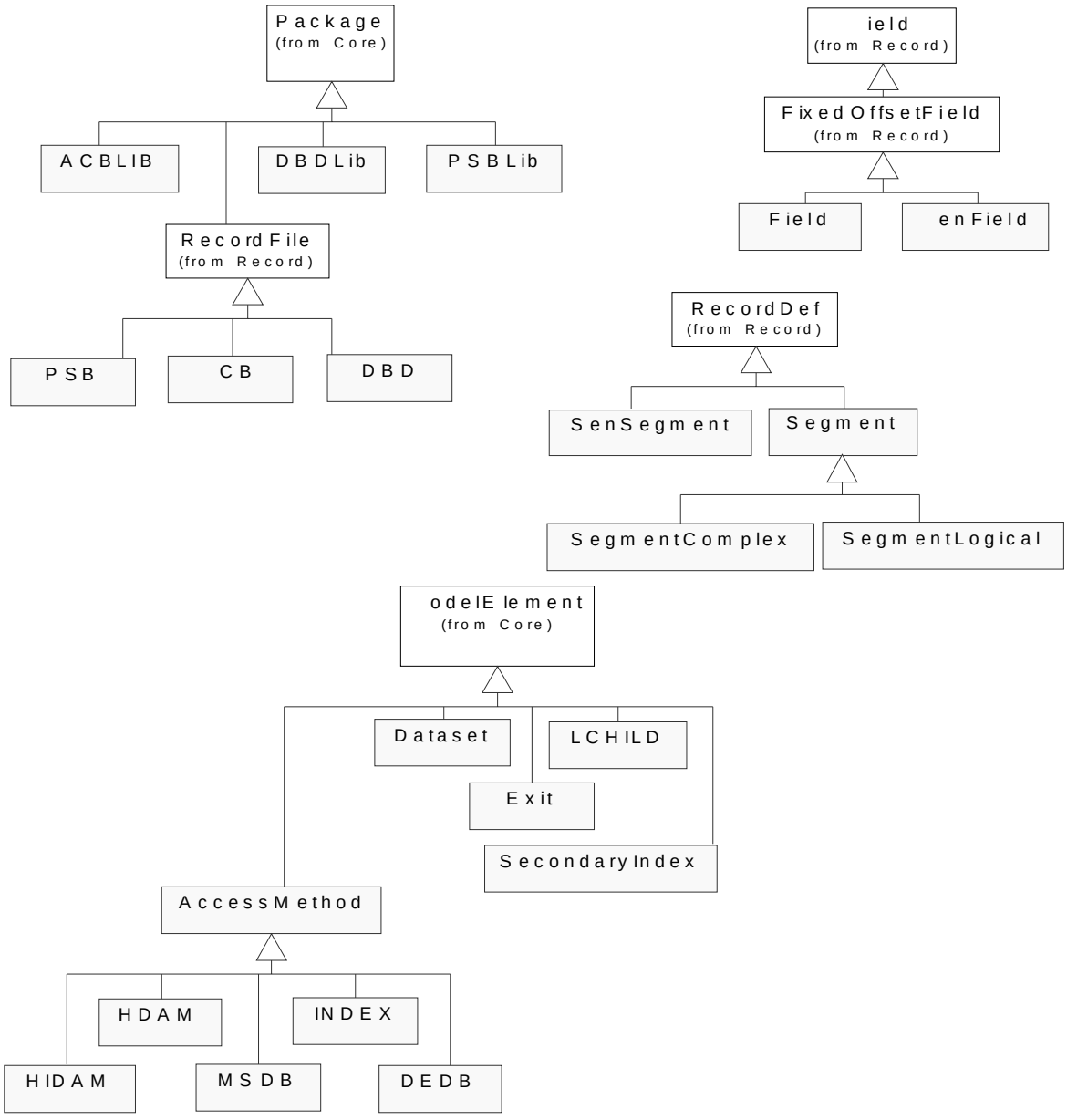

Figure 5-3 Showing inheritance from Record Oriented and ObjectModel classes

5.3 IMS Classes

5.3.1 ACBLIB

This class represents the collection of components needed for an IMS ACB (Application Control Block).

An IMS application will use one or more PSBs. For an application to be compiled successfully, all of the PSBs and all of the DBDs referenced by those PSBs must be collected into an ACBLIB.

Superclasses

Package

References

dbd

The DBDs used in this ACBLIB.

class:	DBD
defined by:	ContainsDBD::dbd
multiplicity:	zero or more
inverse:	DBD::acblib

psb

The PSBs used in this ACBLIB

class:	PSB
defined by:	Contains PSB::psb
multiplicity:	zero or more
inverse:	PSB::acblib

5.3.2 AccessMethod

An instance of a subtype of this virtual class holds access-method-specific attributes of a DBD user object. DBDs with access methods MSDB, INDEX, HIDAM, DEDB or HDAM will use instances of the subclasses of this object class.

Superclasses

ModelElement

References

dbd

DBD extended by this access method instance.

class:	DBD
defined by:	ExtendedByAccessMethod::dbd
multiplicity:	exactly one
inverse:	dbd::accessMethod

5.3.3 DBD

An instance of this object class represents an IMS Data Base Description, which is the Root entity for a DBD object.

DBDs describe the organization of data and the pathways by which an application program can retrieve or store Records. A Record within a DBD is called a Segment. Segments are connected by parent-child relationships to create the information hierarchy.

Superclasses

RecordFile

Contained Elements

AccessMethod

Dataset

Exit

Segment

Attributes

dliAccess

This attribute holds the access method of the DBD.
PSINDEX, PHDAM, and PHIDAM are new valid values added for IMS V6.

type:	AccessMethodType Valid values: (DEDB | GSAM | HDAM | HIDAM | HISAM | HSAM | INDEX | LOGICAL | MSDB | PSINDEX | PHDAM | PHIDAM | SHSAM | SHISAM)
multiplicity:	exactly one

isVSAM

This attribute indicates whether the operating system access method for the DBD is VSAM. It affects the string in the ACCESS keyword in the generated DBD when dliAccess=GSAM, HDAM,or HIDAM.

type:	Boolean
multiplicity:	zero or one

passwordFlag

This attribute is a flag to indicate whether PASSWD=YES should be specified on the DBD macro.

type:	Boolean
multiplicity:	zero or one

versionString

This is a 255-character string that is generated with the VERSION keyword to serve as a descriptive label on the DBD.

type:	String
multiplicity:	zero or one

References

acblib

The ACBLIB(s) in which the DBD is used.

class:	ACBLIB
defined by:	ContainsDBD::acblib
multiplicity:	zero or more
inverse:	ACBLIB::dbd

accessMethod

Connection to additional attributes and relationships that apply to a specific access method.

class:	AccessMethod
defined by:	ExtendedByAccessMethod::accessMethod
multiplicity:	zero or one
inverse:	AccessMethod::dbd

dataset

Dataset information for this DBD.

class:	Dataset
defined by:	ContainsDataset::dataset
multiplicity:	zero or more
inverse:	Dataset::dbd

exit

Data capture exit used by this DBD.

class:	Exit
defined by:	Captures::exit
multiplicity:	zero or more
inverse:	Exit::dbd

library

DBDLIB(s) in which DBD is stored.

class:	DBDLib

defined by:	IsInDBDLib::library
multiplicity:	zero or more
inverse:	DBDLib::dbd

pcb

The PCBs that are based on this DBD.

class:	PCB
defined by:	PcbToDbd::pcb
multiplicity:	zero or more
inverse:	PCB::dbd

segment

Segments that are part of this DBD.

class:	Segment
defined by:	ContainsSegment::segment
multiplicity:	zero or more; ordered
inverse:	Segment::dbd

5.3.4 DBDLib

A DBDLib is a collection of DBDs, comparable to a COPYlib for data structures.

Superclasses

Package

References

dbd

DBDs stored in the DBDLIB.

class:	DBD
defined by:	IsInDBDLib::dbd
multiplicity:	zero or more
inverse:	DBD::library

5.3.5 DEDB

An instance of this object class represents a DBD user object that has access=DEDB. A DEDB DBD is a Fast Path DBD designed for very fast transactions. It must have a randomizing module name to be valid. STAGE and XCI (Extended Call Interface) were new parameters added with IMS 5.

Superclasses

AccessMethod

Attributes

rmName

This attribute is the name of the executable module used to randomize the database.

type:	String
multiplicity:	zero or one

stage

This attribute specifies whether or not the randomizer is a 1 or 2 stage process (default is 1).
Valid Values: 1 or 2

type:	Integer
multiplicity:	zero or one

extendedCall

This attribute specifies whether or not the ramdomizer should use the extended call interface.

type:	Boolean
multiplicity:	zero or one

5.3.6 Dataset

Instances of this object type are used to hold attributes for the DATASET and AREA macro statements.

DATASET and AREA macro statements describe the physical storage of the DBD in MVS datasets that are connected to an application by use of DD Statements in the JCL.

Superclasses

ModelElement

Attributes

dd1name

Name of the primary or input dataset

type:	String
multiplicity:	exactly one

dd2name

Name of overflow or output dataset.

type:	String
multiplicity:	zero or one

device

This attribute is the DEVICE specified on the DATASET statement.

type:	DeviceType Valid values: (2305 | 2319 | 3330 | 3350 | 3375 | 3380 | 3390 | 2400 | 3400 | TAPE)
multiplicity:	zero or one

model

This attribute holds the model part of the DEVICE attribute on the DATASET statement. The value is not relevant for most device types.

type:	ModelType Valid values: (1 | 2 | 11)
multiplicity:	zero or one

size1

Size of area or primary or input dataset.
For DEDB databases, valid values are: 512, 1024, 2048, 4096, 8192, 12288, 16384, 20480, 24576, 28672, and the Default value is 4096

type:	Integer
multiplicity:	zero or one

size2

Size of overflow or output dataset

type: Integer

multiplicity: zero or one

recordLength1

Record length in primary or input dataset.

type: Integer

multiplicity: zero or one

recordLength2

Record length in overflow or ioutput dataset.

type: Integer

multiplicity: zero or one

blockingFactor1

Blocking factor in primary or input dataset

type: Integer

multiplicity: zero or one

blockingFactor2

Blocking factor in overflow or output dataset.

type: Integer

multiplicity: zero or one

datasetLabel

This attribute holds a label used for reverse referencing of datasets in a HDAM or HIDAM database.

type: String

multiplicity: zero or one

freeBlockFrequency

This attribute describes the frequency of free blocks in the initial dataset layout.
Valid Values are 0, 2-100, null

type:	Integer
multiplicity:	zero or one

freeSpacePercentage

This attribute describes the percentage of free space in the initial dataset layout.
Valid Values are 0-99, null

type:	Integer
multiplicity:	zero or one

recordFormat

This attribute describes the record format for a GSAM database.
Valid Values are F, FB, V, VB, U, null

type:	RECFMType Valid values: (F | FB | V | VB | U)
multiplicity:	zero or one

scanCylinders

This attribute describes the number of cylinders to be scanned to find space for new data.
Valid values are 0-255

type:	Integer
multiplicity:	zero or one

searchAlgorithm

This attribute specifies where should IMS look for space in which to put new data.
Valid Values are 0, 1, 2, null

type:	AlgorithmType Valid values: (0 | 1 | 2)
multiplicity:	zero or one

root

This attribute holds total space allocated to the root addressable part of the AREA in terms of UOWs. A value of 0 represents a null value.
Valid Values are 0 or 2-32767

type:	Integer
multiplicity:	zero or one

rootOverflow

This attribute holds the amount of space reserved for independent overflow in terms of units of work. A value of 0 represents a null value.
Valid values are 0 or 1-32767
Constraints:
root and rootOverflow must be specified together.
The value in rootOverflow must be less than the value in root.

type:	Integer
multiplicity:	zero or one

uow

This attribute holds the number of control intervals in a unit of work. A value of 0 represents a null value.
Valid values: 0 or 2-32767

type:	Integer
multiplicity:	zero or one

uowOverflow

This attribute holds the number of control intervals in overflow section of a unit of work. A value of 0 represents a null value.
Valid values: 0 or 1-32767

type:	Integer
multiplicity:	zero or one

References

dbd

DBD that uses this DATASET statement.

class:	DBD
defined by:	ContainsDataset::dbd
multiplicity:	exactly one
inverse:	DBD::dataset

segment

Segment whose physical data is stored in the physical dataset represented by the DATASET statement.

class:	SegmentComplex
defined by:	StoresSegment::segment
multiplicity:	zero or more
inverse:	SegmentComplex::dataset

5.3.7 Exit

This class represents a Data Capture exit routine, which is specified to enable DB2 applications and end users to access updated IMS data. Data Capture exits can apply to an entire DBD or to a specific segment.

Superclasses

ModelElement

Attributes

key

Specifies whether the exit routine is passed the physical concatenated key. This key identifies the physical segment updated by the application. A value of TRUE makes to KEY, a value of FALSE maps to NOKEY.

type: Boolean

multiplicity: zero or one

data

This attribute specifies whether the physical segment data is passed to the exit routine for updating. When DATA is specified and a Segment Edit/Compression exit routine is also used, the data passed is expanded data. A value of TRUE maps to DATA, a value of FALSE maps to NODATA.

type: Boolean

multiplicity: zero or one

path

This attribute specifies whether the data from each segment in the physical root's hierarchical path must be passed to the exit routine for an updated segment. A value of TRUE maps to PATH; a value of FALSE maps to NOPATH.

type: Boolean

multiplicity: zero or one

log

This attribute specifies whether the data capture control blocks and data should be written to the IMS system log.
A value of TRUE maps to LOG; a value of FALSE maps to NOLOG.

type: Boolean

multiplicity: zero or one

cascade

This attribute specifies whether the exit routine is called when DL/I deletes this segment because the application deleted a parent segment. Using CASCADE ensures that data is captured for the defined segment.
A value of TRUE maps to CASCADE; a value of FALSE maps to NOCASCADE.

type: Boolean

multiplicity: zero or one

cascadeKey

This attribute specifies whether to pass the physical concatenated key to the exit. This key identifies the segment being deleted by a cascade delete.
A value of TRUE maps to PATH; a value of FALSE maps to NOPATH.

type: Boolean

multiplicity: zero or one

cascadeData

The attribute specifies whether to pass segment data to the exit routine for a cascade delete. DATA also identifies the segment being deleted when the physical concatenated key is unable to do so.
A value of TRUE maps to DATA; a value of FALSE maps to NODATA.

type: Boolean

multiplicity: zero or one

cascadePath

This attribute specifies whether to allow an application to separately access several segments for a cascade delete.
A value of TRUE maps to PATH; a value of FALSE maps to NOPATH.

type: Boolean

multiplicity: zero or one

References

dbd

The DBD instance that uses this data capture exit.

class: DBD

defined by:	Captures::dbd
multiplicity:	zero or one
inverse:	DBD::exit

segment

The Segment instance that uses this data capture exit.

class:	Segment
defined by:	CapturesExit::segment
multiplicity:	zero or one
inverse:	Segement::exit

5.3.8 Field

This sub-type is used to add the attributes that apply to a Record Field only within the context of a specific DBD. One or more Fields within a Segment can be defined as a Sequence Field, which is the Key for the segment. A Sequence Field can be defined as Unique or Multiple. A Segment description does not have to include all of the Fields defined within the data structure.

Superclasses

FixedOffsetField

Attributes

sequenceField

This attribute is a flag to indicate whether the dataItem instance should be generated as a sequence field or not.

type:	Boolean
multiplicity:	zero or one

uniqueSequence

This attribute is a flag to indicate whether the dataItem instance should be generated as a unique sequence field.

type:	Boolean
multiplicity:	zero or one

fieldLength

The attribute holds the length of the data item. This value should come from the data item itself.

type:	Integer
multiplicity:	exactly one

generated

Indicates if the field has been generated by analyzing the "copybook" associated with the structure/segment, or if the field is coming directly from the IMS catalog

type:	Boolean
multiplicity:	exactly one

References

ddataIndex

The index that uses this field as duplicate data.

class:	SecondaryIndex
defined by:	IsDuplicateData::ddataIndex
multiplicity:	zero or more
inverse:	SecondaryIndex::ddataFields

searchIndex

The index that uses this field for a search field.

class:	SecondaryIndex
defined by:	Searched::searchIndex
multiplicity:	zero or more
inverse:	SecondaryIndex::searchFields

subseqIndex

The index that uses this field as a subsequence field.

class:	SecondaryIndex
defined by:	Subsequenced::subseqIndex
multiplicity:	zero or more
inverse:	SecondaryIndex::subseqFields

senField

Instances of this relationship type are used to connect a SenField instance to the Field instance that represents a field to which the sensitiveSegment is sensitive.

class:	SenField
defined by:	SenfldToFiled::senfield
multiplicity:	zero or more
inverse:	SenField::field

5.3.9 HDAM

An instance of this object class represents a DBD user object that has access=HDAM. These attributes are part of the randomizing module information that a valid HDAM DBD must have.

HDAM is a full-function DBD with indexing and logical relationships.

Superclasses

AccessMethod

Attributes

rmName

This attribute is the name of the executable module used to randomize the database.

type:	String
multiplicity:	zero or one

relativeBlockNumber

The maximum relative block number that the user wishes to allow a randomizing module to produce for a database. This attributes determines the number of control intervals or blocks in the root-addressable area of a HDAM database.
Valid Values: 0 or 1-16777215

type:	Integer
multiplicity:	zero or one

rootAnchorPoints

The number of root anchor points desired in each control interval of block in the root addressable area.
Valid Values: 0 or 1-255

type:	Integer
multiplicity:	zero or one

rootMaxBytes

The maximum number of bytes of database record that can be stored in the root-addressable area in a series of inserts unbroken by a call to another database record.

type:	Integer
multiplicity:	zero or one

5.3.10 HIDAM

An instance of this object class represents a DBD user object with an access method of HIDAM. A HIDAM DBD must have a primary index relationship to be valid. The relationship maps to an LCHILD statement under the root segment that has POINTER=INDX and no associated XDFLD statement.

A HIDAM DBD is a full function DBD with indexing and logical relationships.

Superclasses

AccessMethod

References

index

The primary index for this HIDAM DBD

class:	INDEX
defined by:	PrimaryIndex::index
multiplicity:	zero or one
inverse:	INDEX::primaryTarget

5.3.11 INDEX

An instance of the DBDindex object class represents a DBD user object that can be used to index a HIDAM database or a segment in a HDAM, HIDAM or HISAM database. The indexing relationship maps to the LCHILD statement in the macro language description of an index DBD.

An INDEX DBD can also be treated as a normal single-segment DBD.

Superclasses

AccessMethod

Attributes

dosCompatibility

This attribute indicates whether the index DBD was created with DLI/DOS with a segment code as part of the prefix.

type:	Boolean
multiplicity:	zero or one

protect

This attribute is a flag for data integrity in index pointer segments.

type:	Boolean
multiplicity:	zero or one

References

primaryTarget

The HIDAM DBD for which this Index is the primary index.

class:	HIDAM
defined by:	PrimaryIndex::primaryTarget
multiplicity:	zero or one
inverse:	HIDAM::index

secondaryTarget

The secondary index relationship to a complex segment.

class:	SecondaryIndex
defined by:	Indexes::secondaryTarget
multiplicity:	zero or one
inverse:	SecondaryIndex::index

sharedIndex

The first DBD that defines the Dataset shared by the rest of the index DBDs.

class:	INDEX

defined by:	IndexShares::sharedIndex
multiplicity:	zero or one
inverse:	INDEX::sharingIndex

sharingIndex

The second and later Index DBDs that share a dataset.

class:	INDEX
defined by:	IndexShares::sharingIndex
multiplicity:	zero or one
inverse:	INDEX::sharedIndex

sequencedPCB

The PCB(s) that use this secondary index for processing sequence.

class:	PCB
defined by:	SequencedBy::sequencedPCB
multiplicity:	zero or more
inverse:	PCB::procSeq

senseg

The sensitive segments indexed by this Index DBD

class:	SenSegment
defined by:	Indices::senseg
multiplicity:	zero or more
inverse:	SenSegment::index

5.3.12 LCHILD

This type holds the attributes that apply to the relationship used to connect a SegmentComplex instance to the SegmentComplex instances for which it is the logical parent (maps to LCHILD statement and to the PARENT keyword on the SEGM statement).

Superclasses

ModelElement

Attributes

counter

This attribute holds a flag for whether COUNTER keyword is to be used in the POINTER= parameter on the child segment.

type:	Boolean
multiplicity:	zero or one

lcPointer

This attribute holds a value used in the POINTER keyword on the LCHILD macro to specify amount of pointer fields to be reserved in the logical parent segment.

type:	ChildPointerType Valid values: (SNGL | DBLE | NONE)
multiplicity:	zero or one

lparentFlag

This attribute holds a flag for whether LPARNT keyword is to be used in the POINTER= parameter on the child segment.

type:	Boolean
multiplicity:	zero or one

ltwin

This attribute holds a value to be used in the POINTER= parameter on the child segment in order to specify logical twin pointers.

type:	LPointerType Valid values: (LTWIN | LTWINBWD)
multiplicity:	zero or one

rules

This attribute holds a value used in the RULES keyword on the LCHILD macro to control the logical twin sequence.
Valid Values are FIRST, LAST, HERE

type:	RulesType Valid values: (FIRST | LAST | HERE)
multiplicity:	zero or one

virtualParent

virtualParent
This attribute holds a value used in the PARENT parameter on the logical child segment to specify whether the concatenated key of the logical parent segment is stored with each logical child segment.

type:	ParentType Valid values: (VIRTUAL | PHYSICAL)
multiplicity:	zero or one

References

lparent

The segment that represents the parent in a logical parent relationship.

class:	SegmentComplex
defined by:	IsLChild::lparent
multiplicity:	exactly one
inverse:	SegmentComplex::lchild

lchild

The child segment in the logical parent relationship.

class:	SegmentComplex
defined by:	IsLParent::lchild
multiplicity:	exactly one
inverse:	SegmentComplex::lparent

pairedSegment

The pair relationship to a physical child of the logical parent segment.

class:	SegmentComplex
defined by:	IsPaired::pairedSegment
multiplicity:	zero or one
inverse:	SegmentComplex::pairedLCHILD

5.3.13 MSDB

A DBD with access=MSDB (Mass Storage Data Base) has msdbField and msdbType information instead of physical datasaet information.

Superclasses

AccessMethod

Attributes

msdbField

This attribute holds a search field name for a Mass Storage Data Base.

Update Constraints
A string is required in msdbField when msdbType=FIXED, TERM,or DYNAMIC for the DBD user object to be valid.
The string in msdbField must not be the same as the name on any FIELD statement in this DBD.
msdbField must be null when msdbType=NO.

type:	String
multiplicity:	zero or one

msdbType

This attribute specifies the type of Mass Storage Data Base. It may be NO (nonterminal-related without terminal-related keys which has key and sequence field as part of the segment), FIXED (terminal-related fixed), TERM (nonterminal-related with terminal-related keys), DYNAMIC (terminal-related dynamic) or null

type:	MSDBtype Valid values: (NO | TERM | FIXED | DYNAMIC)
multiplicity:	exactly one

5.3.14 PCB

A PCB is a series of macro instructions contained in a PSB. PCBs which come in three varieties:

- TP (Teleprocessing) PCBs describe a connection to a terminal
- GSAM PCBs connect a PSB to a input or output file.
- DB PCBs connect a PSB to the data defined by a DBD.

Superclasses

RecordFile

Contained Elements

SenSegment

Attributes

pcbType

The type of PCB - whether GSAM, DB or TP

type:	PCBType Valid values: (DB | GSAM | TP)
multiplicity:	exactly one

list

This attribute specifies whether a named PCB is included in the PCB list passed to the application program at entry. TRUE includes the PCB in the PCB list, FALSE excludes it from the PCB list.

type:	Boolean
multiplicity:	zero or one

keyLength

The value specified in bytes of the longest concatenated key for a hierarchic path of sensitive segments used by the application program in the logical data structure.

type:	Integer
multiplicity:	zero or one

processingOptions

This attribute holds a string that represents the processing options on either the sensitive segments or the data set declared in this PCB and which can be used in an associated application program.

type:	String
multiplicity:	zero or one

positioning

This attribute specifies whether single or multiple positioning is desired for the logical data structure. Single or multiple positioning provides a functional variation in the call. Multiple positioning is not supported by HSAM.

type:	PositioningType Valid values: (S | M)
multiplicity:	zero or one

sequentialBuffering

The value in this attribute specifies if this PCB will be buffered using sequential buffering (SB).
True means the SB should be activated conditionally (COND);
False means that SB should not be used for this DB PCB (NO).

type: Boolean

multiplicity: zero or one

alternateResponse

This attribute specifies whether this PCB can be used instead of the I/O PCB for responding to terminal in response mode, conversational mode, or exclusive mode.

type: Boolean

multiplicity: zero or one

express

This attribute specifies whether messages from this alternate PCB are to be sent (TRUE) or are to be backed out (FALSE) if the application program should terminate abnormally.

type: Boolean

multiplicity: zero or one

modify

This attribute specifies whether the alternate PCB is modifiable. This feature allows for the dynamic modification of the destination name associated with this PCB.

type: Boolean

multiplicity: zero or one

sameTerminal

This attribute specifies whether IMS should verify that the logical terminal named in the response alternate PCB is assigned to the same physical terminal as the logical terminal that originated the input message.

type: Boolean

multiplicity: zero or one

destinationType

The attribute specifies whether the ltermName attribute signifies a logical terminal (LTERM) or a transaction code (NAME). This attribute maps to the LTERM or NAME keyword on the PCB macro statement.

type:	LTermType Valid values: (LTERM | NAME)
multiplicity:	zero or one

ltermName

This attribute specifies the name of the actual destination of the message and is either a logical terminal name or a transaction-code name. When the name is a transaction-code name, output messages to this PCB are queued for input to the program used to process the transaction-code named by the NAME attribute. The name must be specified in the user's IMS/VS system definition as a logical terminal name or transaction code. This attribute maps to the LTERM/NAME keyword on the PCB macro statement.

type:	String
multiplicity:	zero or one

References

dbd

The DBD on which this PCB is based.

class:	DBD
defined by:	PcbToDbd::dbd
multiplicity:	zero or one
inverse:	DBD::pcb

psb

The PSB(s) that use this PCB.

class:	PSB
defined by:	PsbToPcb::psb
multiplicity:	zero or more
inverse:	PSB::pcb

senSegment

The sensitive segments included in this PCB.

class:	SenSegment
defined by:	PcbToSenSegment::senSegment
multiplicity:	zero or more
inverse:	SenSegment::pcb

procSeq

The secondary index that this PCB uses as a processing sequence.

class:	INDEX
defined by:	SequencedBy::procSeq
multiplicity:	zero or one
inverse:	INDEX::sequencedPCB

5.3.15 PSB

An instance of this object class represents the root entity of a PSB user object. Within IMS, a PSB (Program Specification Block) is a series of PCB macro instructions that describe an application program's I/O operations and its view and use of segments and fields in IMS databases. The types of PCBs are TP PCB which describes interactions with logical terminals, GSAM PCB which is based on a GSAM DBD used as an input or output dataset, and DB PCB which can relate to segments and fields in its base DBD.

Superclasses

RecordFile

Attributes

compatibility

The value in this attribute provides for compatibility between BMP or MSG and Batch-DL/I parameter lists. When TRUE, the PSB is always treated as if there were an I/O PCB, no matter how it is used. When FALSE, the PSB has an I/O PCB added only when run in a BMP or MSG region.

type:	Boolean
multiplicity:	zero or one

ioErrorOption

The value in this attribute represents the condition code returned to the operating system when IMS/VS terminates normally and one or more input or output errors occurred on any data base during the application program execution.

type:	Integer
multiplicity:	zero or one

ioaSize

This attribute holds the size of the largest I/O area to be used by the application program. The size specification is used to determine the amount of main storage reserved in the PSB pool to hold the control region's copy of the user's I/O area data during scheduling of this application program. If this value is not specified, the ACB utility program calculates a maximum I/O area size to be used as a default. The size calculated is the total length of all sensitive segments in the longest possible path call. The value specified is in bytes.

type:	Integer
multiplicity:	zero or one

language

This attribute holds the language label used on the PSBGEN statement.

type:	PSBLanguageType Valid values: (ASSEM | C | COBOL | PL/I | PASCAL)
multiplicity:	zero or one

lockMaximum

The value in this attribute indicates the maximum number of locks an application program can get at one time. The value is specified in units of 1000. For example, a lockMaximum value of 5 indicates a maximum of 5000 locks at one time. A value of 0 turns off the limit.

type: Integer

multiplicity: zero or one

maximumQxCalls

The value in this attribute represents the maximum number of data base calls with Qx command codes which may be issued between synchronization points. If this number is exceeded, the application program will abend.

type: Integer

multiplicity: zero or one

onlineImageCopy

This attribute specifies whether the user of this PSB is authorized to execute the Online Data Base Image Copy utility or the Surveyor utility feature run as a BMP against a data based named in this PSB. When TRUE, use of the Online Image Copy and the Surveyor utility feature is allowed; When FALSE, use of the Online Image copy and the Surveyor utility feature is prohibited.

type: Boolean

multiplicity: zero or one

ssaSize

The value in this attribute represents the maximum total length of all SSAs to be used by the application program. The size specification is used to determine the amount of main storage reserved in the PSB pool to hold the control region's copy of the user's SSA string during scheduling of this application program. If not specified, the ACB utility program calculates the maximum SSA size to be used as a default. The size calculated is the maximum number of levels in any PCB within this PSB times 280. The value specified is in bytes.

type: Integer

multiplicity: zero or one

writeToOperator

This attributes holds a subparameter of the IOEROPN parameter. It is tied to the "write-to-operator-with-reply" function in the Utility Control facility. When TRUE, a WTOR for the DFS0451A I/O error message is issued, and DL/I waits for the operator to respond before continuing.

type:	Boolean
multiplicity:	zero or one

References

acblib

The ACBLIB(s) that use this PSB.

class:	ACBLIB
defined by:	ContainsPSB::acblib
multiplicity:	zero or more
inverse:	ACBLIB::psb

library

The PSBLIB(s) in which this PSB is stored.

class:	PSBLib
defined by:	IsInPSBLib::library
multiplicity:	zero or more
inverse:	PSBLib::psb

pcb

The PCBs used by this PSB.

class:	PCB
defined by:	PsbToPcb::pcb
multiplicity:	zero or more
inverse:	PCB::psb

5.3.16 PSBLib

A collection of PSBs - comparable to a COPYlib for data structures.

Superclasses

Package

References

psb

The PSBs stored in this PSBLIB.

class:	PSB
defined by:	IsInPSBLib::psb
multiplicity:	zero or more
inverse:	PSB::library

5.3.17 SecondaryIndex

This type holds the attributes on the relationships between a SegmentComplex instance and the INDEX instances that act as secondary indexes for the DBD (maps to combination of LCHILD and XDFLD statements).

Superclasses

ModelElement

Attributes

constant

This attribute holds a character string that defines a one-byte self-defining term with which every index pointer segment in a particular secondary index is identified. It is used to identify index pointer segments for a specific secondary index when multiple secondary indexes reside in the same database.

type:	String
multiplicity:	zero or one

exitRoutine

This attribute is the name of the executable module that suppresses creation of index pointer segments.

type:	String
multiplicity:	zero or one

nullValue

This attribute holds a character string that is a one-byte self-defining term. The creation of index pointer segments is suppressed when the specified value is contained in the search field of an index pointer segment.

type:	String
multiplicity:	zero or one

References

index

The index used in the secondary index relationship.

class:	INDEX
defined by:	Indexes::index
multiplicity:	exactly one
inverse:	INDEX::secondaryTarget

segment

The segment that is being indexed.

class:	SegmentComplex
defined by:	IsIndexedBy::segment
multiplicity:	exactly one
inverse:	SegmentComplex::secondaryIndex

indexSource

The segment that contains the Search, Subsequence and Duplicate data fields for the index relationship.

class:	SegmentComplex
defined by:	HasIndexSource::indexSource
multiplicity:	zero or one
inverse:	SegmentComplex::sourcedIndex

ddataFields

Fields used for duplicate data in the index relationship.

class:	Field
defined by:	IsDuplicateData::ddataFields
multiplicity:	0..5
inverse:	Field::ddataIndex

searchFields

The fields used for search fields by the secondary index.

class:	Field
defined by:	Searched::searchFields
multiplicity:	0..5
inverse:	Field::searchIndex

subseqFields

The fields used as subsequence fields by the secondary index.

class:	Field

defined by:	Subsequenced::subseqFields
multiplicity:	0..5
inverse:	Field::subseqIndex

5.3.18 Segment

An instance of this object class represents a segment within a DBD user object.

A segment is the IMS-view of a data structure that maps the fields in the segment.

Superclasses

Record

Contained Elements

Field

Exit

Attributes

directDependent

This attribute indicates whether the segment is direct dependent or sequential. A value of TRUE specifies use of DIR as the segment type on the generated DBD. A value of FALSE specifies use of SEQ on the generated DBD. This attribute is ignored for the root segment of the DBD user object.

type: Boolean

multiplicity: zero or one

exitFlag

This attribute is a flag to indicate whether a segment will use the data capture exits specified on the DBD. A valid of FALSE maps to use of EXIT=NONE parameter on the SEGM macro. This flag has no meaning when exits points to any instances of PropagatedBy.

type: Boolean

multiplicity: zero or one

frequency

This attribute holds estimated number of times that this segment will occur for each occurrence of its physical parent.

type: String

multiplicity: zero or one

maximumLength

This attribute holds the length of a fixed-length segment, or the maximum length of a variable length segment.

type: Integer

multiplicity: zero or one

minimumLength

This attribute holds the minimum length of a variable length segment.

type: Integer

multiplicity: zero or one

pcPointer

This attribute describes the type of physical child pointer to be stored in the prefix area of the segment in the DBD.
Valid Values are SNGL, DBLE, null

type:	ChildPointerType Valid values: (SNGL | DBLE | NONE)
multiplicity:	zero or one

rules

This attributes holds the value that indicates where to place new occurrences of this segment type in the physical database.

type:	RulesType Valid values: (FIRST | LAST | HERE)
multiplicity:	zero or one

subsetPointers

This attribute holds the number of subset pointers in a direct dependent segment in a DEDB DBD. Valid values are 0-8.

type:	Integer
multiplicity:	zero or one

References

dbd

DBD that owns this segment.

class:	DBD
defined by:	ContainsSegment::dbd
multiplicity:	exactly one
inverse:	DBD::segment

exit

Data capture exit used by this segment.

class:	Exit

defined by:	CapturesExit::exit
multiplicity:	zero or more
inverse:	Exit::segment

logical

The logical segment that is based on this physical segment.

class:	SegmentLogical
defined by:	HasSource::logical
multiplicity:	zero or more
inverse:	SegmentLogical::physical

child

The physical child segment.

class:	Segment
defined by:	ParentChild::child
multiplicity:	zero or more
inverse:	Segment::parent

parent

The physical parent segment.

class:	Segment
defined by:	ParentChild::parent
multiplicity:	zero or one
inverse:	Segment::child

senseg

The sensitive segments that use this segment.

class:	SenSegment
defined by:	SensegMapsTo::senseg
multiplicity:	zero or more
inverse:	SenSegment::segment

5.3.19 SegmentComplex

This subclass of Segment supports the full-function features that are limited to HDAM, HIDAM and HISAM databases: specifically logical children and secondary indexes.

Superclasses

Segment

Contained Elements

SecondaryIndex

LCHILD

Attributes

deleteFlag

This attribute holds the value used for the delete rule.

type:	FlagsType Valid values: (P | L | V | B)
multiplicity:	zero or one

dsGroup

This is used to arrange the segments in a partitioned database in a manner comparable to arranging them within datasets in earlier versions of IMS

type:	String
multiplicity:	zero or one

insertFlag

This attribute holds the value used for the insert rule. bidirectional is not used for the insert flag.

type:	FlagsType Valid values: (P | L | V)
multiplicity:	zero or one

replaceFlag

This attribute holds the value used for the replace rule. bidirectional is not used for the replace flag.

type:	FlagsType Valid values: (P | L | V)
multiplicity:	zero or one

segmPointer

This attribute holds the string used for pointer keyword value.

type:	PointerType Valid values: (NOTWIN | TWIN | HIER | TWINBWD | HIERBWD)
multiplicity:	zero or one

References

dataset

Reference to a physical dataset in which this segments physical data is stored.

class:	Dataset
defined by:	StoresSegment::dataset
multiplicity:	zero or one
inverse:	Dataset::segment

lchild

The relationship to the logical children relationships.

class:	LCHILD
defined by:	IsLChild::lchild
multiplicity:	zero or more
inverse:	LCHILD::lparent

lparent

The relationship to the logical parent relationship instance.

class:	LCHILD
defined by:	IsLParent
multiplicity:	zero or one
inverse:	lchild The child segment in the logical parent relationship.

pairedLCHILD

The pair relationship to a logical child of the physical parent segment

class:	LCHILD
defined by:	IsPaired::pairedLCHILD
multiplicity:	zero or one
inverse:	LCHILD::pairedSegment

secondaryIndex

The secondary index relationships that would be represented as LCHILD/XDFLD statement sets.

class:	SecondaryIndex

defined by:	IsIndexedBy::secondaryIndex
multiplicity:	0..32
inverse:	SecondaryIndex::segment

sourcedIndex

The index that uses fields in this segment.

class:	SecondaryIndex
defined by:	HasIndexSource::sourcedIndex
multiplicity:	zero or more
inverse:	SecondaryIndex::indexSource

5.3.20 SegmentLogical

An instance of this object class represents a segment in a DBD user object that has access method of LOGICAL. Segments in a logical DBD use segments in other DBDs instead of defining physical data.

Superclasses

Segment

Attributes

keyData1

This attribute indicates how segment data will be handled when the logical DBD is processed.
A value of TRUE specifies use of "DATA" as the type in the first parameter of the SOURCE keyword in the generated DBD, which directs the segment key to be placed in the key feedback area and the segment data to be placed in the user's I/O area.
A value of FALSE specified use of "KEY" as the type, which directs only the key to be placed in the key feedback area.

type:	Boolean
multiplicity:	zero or one

keyData2

This attribute indicates how segment data will be handled when the logical DBD is processed.
A value of TRUE specifies use of "DATA" as the type in the second parameter of the SOURCE keyword in the generated DBD, which directs the segment key to be placed in the key feedback area and the segment data to be placed in the user's I/O area.
A value of FALSE specified use of "KEY" as the type, which directs only the key to be placed in the key feedback area.
A value of NULL indicates that there is no second SOURCE parameter.

type:	Boolean
multiplicity:	zero or one

References

physical

The real segment that is the basis of the logical segment.

class:	Segment
defined by:	HasSource::physical
multiplicity:	0..2
inverse:	Segment::logical

5.3.21 SenField

This relationship associates a SensitiveSegment instance to the Field instances that

represent the fields in the segment to which the PCB must be sensitive.

Field level sensitivity provides an increased level of data independence by isolating application programs from changes in the arrangement of fields within a segment and addition or deletion of data within a segment.

Additionally, it enhances data security by limiting an application program to a subset of fields within a segment and controlling replace operations at the field level.

Superclasses

FixedOffsetField

Attributes

replace

The value of this attribute specifies whether this field may be altered on a replace call.

type:	Boolean
multiplicity:	zero or one

References

senSegment

The segment that is sensitive to this field.

class:	SenSegment
defined by:	SensegToSenfld::senSegment
multiplicity:	exactly one
inverse:	SenSegment::senField

field

The field to which the PCB is sensitive.

class:	Field
defined by:	SenfldToField::field
multiplicity:	exactly one
inverse:	Field::senField

5.3.22 SenSegment

The type holds the attributes that apply to a PCB's use of a specific segment within a DBD. Application programs using a PCB can only access the segments to which that PCB is sensitive, protecting an hiding some of the data covered by the DBD.

Superclasses

Record

Contained Elements

SenField

Attributes

procoptSENSEG

This attribute holds the processing options allowable for use of this sensitive segment by an associated application program. It has the same meaning as the same attribute in the PCB, plus other options may be specified here which are not allowed on the PCB. This PROCOPT overrides the PCB PROCOPT.

type:	String
multiplicity:	zero or one

subsetPointers

This attribute specifies sensitivity to the array of subset pointers, each of which may be R (read sensitive), U (update sensitive), or N (not sensitive).

type:	String
multiplicity:	zero or one

References

index

The indices of this sensitive segment

class:	INDEX
defined by:	Indices::index
multiplicity:	0..32
inverse:	INDEX::senseg

pcb

The PCB that includes this sensitive segment.

class:	PCB
defined by:	PcbToSenSegment::pcb
multiplicity:	exactly one
inverse:	PCB::senSegment

segment

One segment to which the PCB is sensitive.

class:	Segment
defined by:	SensegMapsTo::segment
multiplicity:	exactly one
inverse:	Segment::senseg

senField

The field to which the segment is sensitive.

class:	SenField
defined by:	SensegToSenfld::senField
multiplicity:	zero or more
inverse:	SenField::senSegment

5.4 IMS Associations

5.4.1 Captures *protected*

This relationship connects a DBD to data capture exits.

Ends

exit

Data capture exit used by this DBD.

class:	Exit
multiplicity:	zero or more

dbd

DBD that uses and owns this data capture exit.

class:	DBD
multiplicity:	zero or one

5.4.2 CapturesExit *protected*

This relationship connects a Segment to data capture exits.

Ends

exit

Data capture exit used by this Segment.

class:	Exit
multiplicity:	zero or more

segment

Segment that uses this data capture exit.

class:	Segment
multiplicity:	zero or one

5.4.3 ContainsDataset *protected*

This relationship connects a DBD to the set of Datasets it uses.

Ends

dbd

The DBD instance that owns this Dataset instance.

class:	DBD
multiplicity:	1..1

dataset

The set of Dataset instances used by this DBD.

class:	Dataset
multiplicity:	0..*

5.4.4 ContainsDBD *protected*

This relationship collects DBDs in an ACBLIB.

Ends

acblib

The ACBLIB(s) in which the DBD is used.

class:	ACBLIB
multiplicity:	zero or more

dbd

The DBDs used in the ACBLIB.

class:	DBD
multiplicity:	zero or more

5.4.5 ContainsPSB *protected*

This relationship collects PSBs in an ACBLIB.

Ends

acblib

The ACBLIB(s) that use this PSB.

class:	ACBLIB
multiplicity:	zero or more

psb

The PSBs used in this ACBLIB.

class:	PSB
multiplicity:	zero or more

5.4.6 ContainsSegment *protected*

Instances of this relationship type are used to connect a DBD instance to the Segment instances that are used in the DBD user object.

Ends

segment

Segments that are part of this DBD.

class:	Segment
multiplicity:	zero or more; ordered

dbd

DBD that owns this segment.

class:	DBD
multiplicity:	exactly one

5.4.7 ExtendedByAccessMethod *protected*

An instance of this relationship type is used to connect an instance of DBD to an instance of AccessMethod or one of its subclasses in order to hold the access-specific attributes of a DBD. Using an extension instead of subtyping DBD directly allows a tool to change the access method of a DBD (or decide the access method later) without having to change the type of the DBD instance.

Ends

dbd

DBD extended by this access method instance.

class:	DBD
multiplicity:	exactly one

accessMethod

Connection to additional attributes and relationships that apply to a specific access method.

class:	AccessMethod
multiplicity:	zero or one

5.4.8 HasIndexSource *protected*

An instance of this relationship type is used to connect a SecondaryIndex instance to the SegmentComplex instance that represents the index source segment for the secondary index (SEGMENT keyword on XDFLD statement).

Ends

indexSource

The segment that contains the Search, Subsequence and Duplicate data fields for the index relationship.

class:	SegmentComplex
multiplicity:	zero or one

sourcedIndex

The index that uses fields in this segment.

class:	SecondaryIndex
multiplicity:	zero or more

5.4.9 HasSource *protected*

This relationship connects a logical segment to the one or two physical segments that contain the data.

This relationship maps to the SOURCE keyword on the SEGM statement in a DBD with access=LOGICAL.

Ends

logical

The logical segment that is based on this physical segment.

class:	SegmentLogical
multiplicity:	zero or more

physical

The real segment that is the basis of the logical segment.

class:	Segment
multiplicity:	0..2

5.4.10 Indexes *protected*

Instances of this relationship type are used to connect each SecondaryIndex instance to the INDEX DBD that acts as a secondary index for the DBD (this information maps to the combination of LCHILD and XDFLD statements).

Ends

secondaryTarget

The secondary index relationship to a complex segment.

class:	SecondaryIndex
multiplicity:	zero or one

index

The index used in the secondary index relationship.

class:	INDEX
multiplicity:	exactly one

5.4.11 IndexShares *protected*

This relationship connects an Index DBD to the other index DBDs sharing the dataset. This models the shared secondary index type of DBD.

Ends

sharingIndex

The second and later Index DBDs that share a dataset.

class:	INDEX
multiplicity:	0..16

sharedIndex

The first DBD that defines the Dataset shared by the rest of the index DBDs.

class:	INDEX
multiplicity:	zero or one

5.4.12 Indices *protected*

Instances of this relationship type are used to connect a SensitiveSegment instance to the INDEX DBDs that are used as INDICES by the SENSEG.

Ends

index

The indices of this sensitive segment

class: INDEX

multiplicity: 0..32

senseg

The sensitive segments indexed by this Index DBD

class: SenSegment

multiplicity: zero or more

5.4.13 IsDuplicateData *protected*

Instances of this relationship type are used to connect a SecondaryIndex instance to the Field instances that are used by the secondary index as duplicate data fields (DDATA keyword on XDFLD statement).

Ends

ddataFields

Fields used for duplicate data in the index relationship.

class: Field

multiplicity: 0..5

ddataIndex

The index that uses this field as duplicate data.

class: SecondaryIndex

multiplicity: zero or more

5.4.14 IsInDBDLib *protected*

DBDs are stored in a DBDlib

Ends

dbd

DBDs stored in the DBDLIB.

class:	DBD
multiplicity:	zero or more

library

DBDLIB(s) in which DBD is stored.

class:	DBDLib
multiplicity:	zero or more

5.4.15 IsIndexedBy *protected*

Instances of this relationship type are used to connect a SegmentComplex instance to the SecondaryIndex instances that act as secondary indexes for the DBD user object (maps to combination of LCHILD and XDFLD statements).

Ends

secondaryIndex

The secondary index relationships that would be represented as LCHILD/XDFLD statement sets.

class:	SecondaryIndex
multiplicity:	0..32

segment

The segment that is being indexed.

class:	SegmentComplex
multiplicity:	exactly one

5.4.16 IsInPSBLib *protected*

PSBs are stored in a PSBlib

Ends

library

The PSBLIB(s) in which this PSB is stored.

class:	PSBLib
multiplicity:	zero or more

psb

The PSBs stored in this PSBLIB.

class:	PSB
multiplicity:	zero or more

5.4.17 IsLChild *protected*

Instances of this relationship type are used to connect a SegmentComplex to its logical children.

Ends

lchild

The relationship to the logical children relationships.

class:	LCHILD
multiplicity:	zero or more

lparent

The segment that represents the parent in a logical parent relationship.

class:	SegmentComplex
multiplicity:	exactly one

5.4.18 IsLParent *protected*

Instances of this relationship connect a SegmentComplex to its logical parent.

Ends

lparent

The relationship to the logical parent relationship instance.

class: LCHILD

multiplicity: zero or one

lchild

The child segment in the logical parent relationship.

class: SegmentComplex

multiplicity: exactly one

5.4.19 IsPaired *protected*

This relationship connects a logical child to the bidirectionally paired segment.

Ends

pairedLCHILD

The pair relationship to a logical child of the physical parent segment

class: LCHILD

multiplicity: zero or one

pairedSegment

The pair relationship to a physical child of the logical parent segment.

class: SegmentComplex

multiplicity: zero or one

5.4.20 ParentChild *protected*

An instance of this relationship type is used to connect an Segment instance to the hierarchical parent Segment instance in the same DBD user object (maps to PARENT keyword on SEGM macro statement).

Ends

child

The physical child segment.

class:	Segment
multiplicity:	zero or more

parent

The physical parent segment.

class:	Segment
multiplicity:	zero or one

5.4.21 PcbToDbd *protected*

An instance of this relationship type is used to connect a PCB to the DBD on which the PCB is based. The base DBD contains all of the segments that the PCB can access.

Ends

dbd

The DBD on which this PCB is based.

class:	DBD
multiplicity:	zero or one

pcb

The PCBs that are based on this DBD.

class:	PCB
multiplicity:	zero or more

5.4.22 PcbToSenSegment *protected*

Instances of this relationship type are used to connect a PCB to the SenSegments that reference the segments to which the PCB is sensitive.

Ends

senSegment

The sensitive segments included in this PCB.

class:	SenSegment
multiplicity:	zero or more

pcb

The PCB that includes this sensitive segment.

class:	PCB
multiplicity:	exactly one

5.4.23 PrimaryIndex *protected*

An instance of this relationship type is used to connect a HIDAM DBD to the INDEX DBD that serves as the primary index for the HIDAM database.

Ends

primaryTarget

The HIDAM DBD for which this Index is the primary index.

class:	HIDAM
multiplicity:	zero or one

index

The primary index for this HIDAM DBD

class:	INDEX
multiplicity:	zero or one

5.4.24 PsbToPcb *protected*

The relationship connects a PSB to the PCBs used within it.

Ends

pcb

The PCBs used by this PSB.

class:	PCB
multiplicity:	zero or more

psb

The PSB(s) that use this PCB.

class:	PSB
multiplicity:	zero or more

5.4.25 Searched *protected*

Instances of this relationship type are used to connect a SecondaryIndex instance to the Field instances that are used by the secondary index as search fields (SEARCH keyword on XDFLD statement).

Ends

searchFields

The fields used for search fields by the secondary index.

class:	Field
multiplicity:	0..5

searchIndex

The index that uses this field for a search field.

class:	SecondaryIndex
multiplicity:	zero or more

5.4.26 SenfldToField *protected*

Instances of this relationship type are used to connect a SensitiveField instance to the Field instance that represents a field to which the sensitiveSegment is sensitive.

Ends

field

The field to which the PCB is sensitive.

class:	Field
multiplicity:	exactly one

senField

The sensitive field that depends on this field.

class:	SenField
multiplicity:	zero or more

5.4.27 SensegMapsTo *protected*

Instances of this relationship type are used to connect a SensitiveSegment instance to the Segment instance that represents a segment to which the PCB is sensitive.

Ends

segment

One segment to which the PCB is sensitive.

class:	Segment
multiplicity:	exactly one

senseg

The sensitive segments that use this segment.

class:	SenSegment
multiplicity:	zero or more

5.4.28 SensegToSenfld *protected*

Instances of this relationship type are used to connect a SensitiveField instance to the SensitiveSegment that is sensitive to a field.

Ends

senSegment

The segment that is sensitive to this field.

class:	SenSegment
multiplicity:	exactly one

senField

The field to which the segment is sensitive.

class:	SenField
multiplicity:	zero or more

5.4.29 SequencedBy *protected*

An instance of this relationship type is used to connect a PCB instance to the INDEX DBD that defines the processing sequence (PROCSEQ parameter) for the PCB.

Ends

procSeq

The secondary index that this PCB uses as a processing sequence.

class:	INDEX
multiplicity:	zero or one

sequencedPCB

The PCB(s) that use this secondary index for processing sequence.

class:	PCB
multiplicity:	zero or more

5.4.30 StoresSegment *protected*

In a full-function database, a segment can be assigned to a specific dataset.

Ends

dataset

Reference to a physical dataset in which this segments physical data is stored.

class:	Dataset
multiplicity:	zero or one

segment

Segment whose physical data is stored in the physical dataset represented by the DATASET statement.

class:	SegmentComplex
multiplicity:	zero or more

5.4.31 Subsequenced *protected*

Instances of this relationship type are used to connect a SecondaryIndex instance to the Field instances that are used by the secondary index as subsequence fields (SUBSEQ keyword on XDFLD statement).

Ends

subseqIndex

The index that uses this field as a subsequence field.

class:	SecondaryIndex
multiplicity:	zero or more

subseqFields

The fields used as subsequence fields by the secondary index.

class:	Field
multiplicity:	0..5

5.5 OCL Representation of IMS Constraints

None

Essbase *6*

6.1 Overview

The Hyperion Essbase package represents the physical data model for a Hyperion Essbase Database. This package extends the Multidimensional package and provides a specific metadata representation for Hyperion's Essbase multidimensional database.

The classes in this package can be used as either sources or targets of data in the data warehouse, and are available to provide a physical implementation of the OLAP data model.

6.2 Organization of the Essbase Package

The Essbase package depends on the following packages:

org.omg::CWM::ObjectModel::Core

org.omg::CWM::Foundation::Expressions

org.omg::CWM::Foundation::SoftwareDeployment

org.omg::CWM::Resource::Multidimensional

org.omg::CWM::Analysis::OLAP

Figure 6-1 shows the Essbase server metamodel. The inheritance of Essbase classes from classes of the Multidimensional, SoftwareDeployment, OLAP, and ObjectModel packages is shown in Figure 6-2. Note that the Essbase metamodel is primarily dependent on classes of the Multidimensional model.

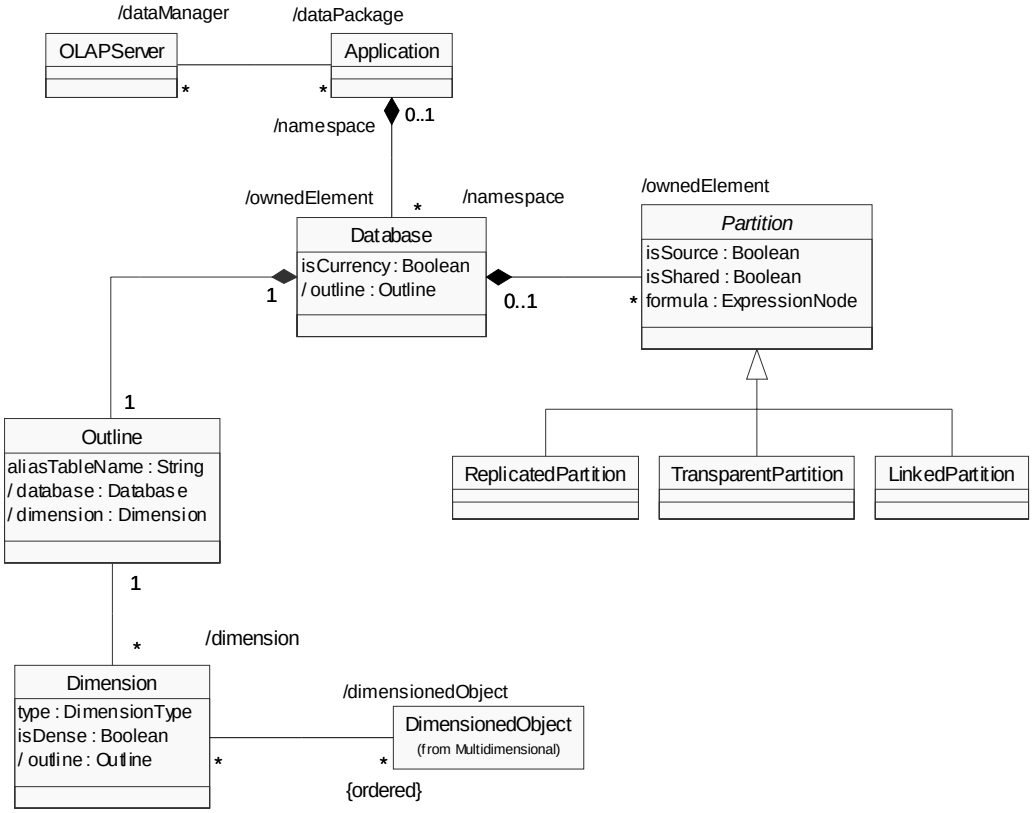

Figure 6-1 Essbase Server Metamodel

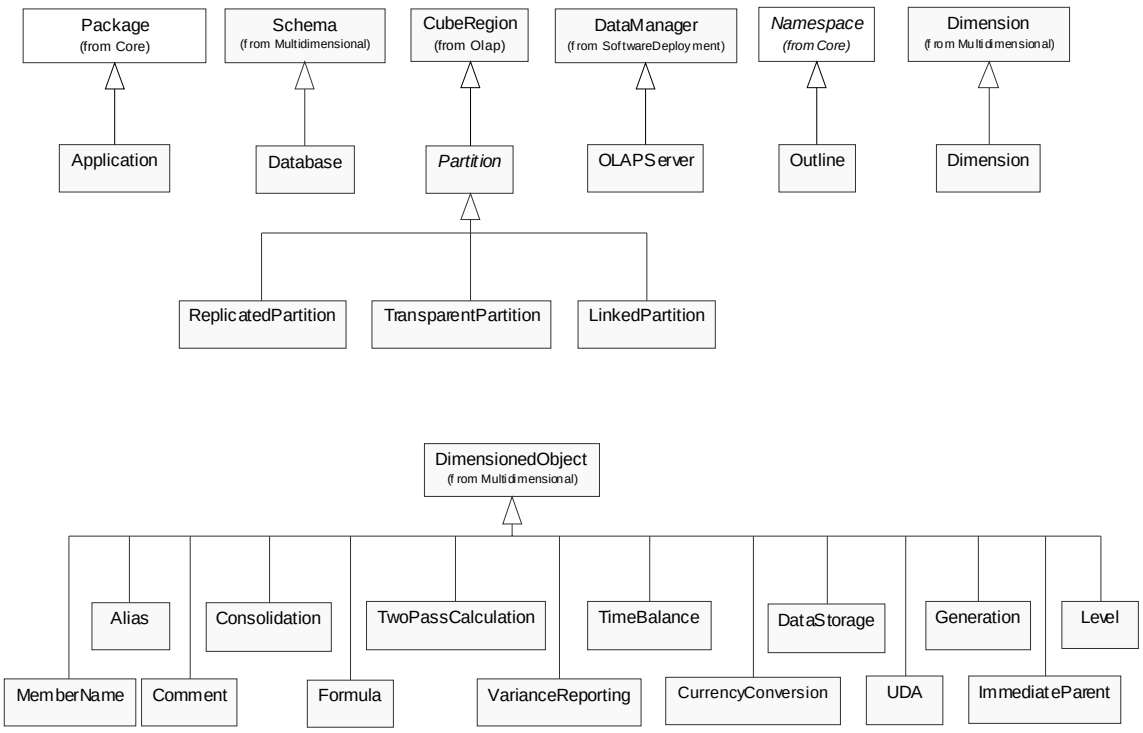

Figure 6-2 Essbase Metamodel Inheritance

6.3 Essbase Classes

6.3.1 Alias

An alias name for an Essbase Dimension member.

Superclasses

DimensionedObject

6.3.2 Application

An Essbase Application is a named container of one or more Databases and their related files. In addition to the Database, an Application may include scripts that are used to load data into the database, calculate derived values, and prepare reports.

Superclasses

Package

Contained Elements

Database

6.3.3 Comment

This is a user-defined comment that can be attached to an Essbase Dimension member.

Superclasses

DimensionedObject

6.3.4 Consolidation

Specifies how this Essbase Dimension member is to roll-up into its parent (e.g., attribute instance values include: +, -, *, /, %, ~).

Superclasses

DimensionedObject

6.3.5 CurrencyConversion

Currency conversion tag for an Essbase Currency Dimension member (e.g., attribute instance values include: "None", "NoConversion", "Category").

Superclasses

DimensionedObject

6.3.6 DataStorage

Data storage tag for an Essbase Dimension member (e.g., attribute instance values include: "StoreData", "DynamicCalc&Store", "DynamicCalc", "NeverShare", "LabelOnly", "SharedMember").

Superclasses

DimensionedObject

6.3.7 Database

An Essbase Database is a unique, named multidimensional database implemented by an Essbase server.

Superclasses

Schema

Contained Elements

- Outline
- Partition

Attributes

isCurrency

If true, then this Database is a Currency Database.

type:	Boolean
multiplicity:	exactly one

References

outline

Reference the Outline owned by the Database.

class:	Outline
defined by:	DatabaseOwnsOutline::outline
multiplicity:	exactly one
inverse:	Outline::database

Constraints

Restrict the cardinality of the namespace role to 1 so that a Database must always be owned by an Application. [C-1]

6.3.8 Dimension

An Essbase Dimension is the primary physical object used in the construction of Essbase Databases.

Superclasses

Dimension

Contained Elements

None

Attributes

type

The type of the Essbase Dimension.

type:	DimensionType (ess_none | ess_accounts | ess_time | ess_country | ess_currencyPartition | ess_attribute)
multiplicity:	exactly one

isDense

Specifies if this Essbase Dimension is sparse or dense.

type:	Boolean
multiplicity:	exactly one

References

outline

References the Outline that organizes this Dimension.

class:	Outline
defined by:	OutlineReferencesDimensions::outline
multiplicity:	exactly one
inverse:	Outline::dimension

Constraints

Essbase Dimensions are not composed from other Essbase dimensions. [C-2]

The inclusion of certain DimensionedObjects is valid only for certain DimensionTypes. [C-3]

6.3.9 Formula

Formula used to calculate the value of an Essbase Dimension member.

Superclasses

DimensionedObject

6.3.10 Generation

Common name/identifier for members occupying the same generation in the Dimension hierarchy, as defined by the Outline.

Superclasses

DimensionedObject

6.3.11 ImmediateParent

Represents the immediate parent of an Essbase Dimension member in the Dimension hierarchy defined by the Outline.

Superclasses

DimensionedObject

6.3.12 Level

Common name/identifier for members occupying the same level in the Dimension hierarchy, as defined by the Outline.

Superclasses

DimensionedObject

6.3.13 LinkedPartition

Subclass of Essbase Partition representing Linked Partitions.

Superclasses

Partition

6.3.14 MemberName

Name for an Essbase Dimension member.

Superclasses

DimensionedObject

6.3.15 OLAPServer

A software process that implements one or more Essbase Databases.

Superclasses

DataManager

6.3.16 Outline

An Essbase Outline defines the structure of an Essbase Database, including the dimensional hierarchies, members, tags, types, consolidations, and mathematical relationships. Data is stored in the Database according to the structure defined in the Outline.

Superclasses

Namespace

Attributes

aliasTableName

The name of the Alias Table to be used by this instance of Outline.

type:	String
multiplicity:	exactly one

References

database

References the Database owning this Outline.

class:	Database
defined by:	DatabaseOwnsOutline::database
multiplicity:	exactly one
inverse:	Database::outline

dimension

References the collection of Dimensions that this Outline organizes.

class:	Dimension
defined by:	OutlineReferencesDimensions::dimension
multiplicity:	zero or more
inverse:	Dimension::outline

Constraints

The Outline name must be the same as the Database name. [C-4]

6.3.17 Partition *abstract*

Defines an abstract Essbase partition class (the superclass of all Essbase partition types).

Superclasses

CubeRegion

Attributes

isSource

If true, then this Partition is a source Partition (i.e., a source of data values for some other target Partition).

type:	Boolean
multiplicity:	exactly one

isShared

If true, then this Partition is a shared source Partition (i.e., shared by several targets).

type:	Boolean
multiplicity:	exactly one

formula

Expression specifying the mapping of source Partition data cells to target Partition data cells.

type:	ExpressionNode
multiplicity:	exactly one

Constraints

Restrict the cardinality of the namespace role to 1 so that a Partition must always be owned by a Database. [C-5]

Only a source Partition can be shared. [C-6]

6.3.18 ReplicatedPartition

Subclass of Essbase Partition representing Replicated Partitions.

Superclasses

Partition

6.3.19 TimeBalance

Time balance tag for an Essbase Accounts Dimension member (e.g., attribute instance values include: "None", "First", "Last", "Average").

Superclasses

DimensionedObject

6.3.20 TransparentPartition

Subclass of Essbase Partition representing Transparent Partitions.

Superclasses

Partition

6.3.21 TwoPassCalculation

This is a tag specifying that a derived (calculated) Accounts member needs to be re-computed following the global calculation of an Essbase Database

This is done to provide a derived Accounts member that's dependent on other account members with a final, correct value, following the sequential calculation of both the Accounts and Time Dimensions (e.g., Profit % Sales).

Superclasses

DimensionedObject

6.3.22 UDA

An Essbase user-defined attribute.

Superclasses

DimensionedObject

6.3.23 VarianceReporting

Variance reporting tag for an Essbase Accounts Dimension (e.g., attribute instance values include: "NonExpense", "Expense").

Superclasses

DimensionedObject

6.4 Essbase Associations

6.4.1 DatabaseOwnsOutline

A Database has exactly one Outline.

Ends

database

The Database that owns the Outline.

class:	Database
multiplicity:	exactly one
aggregation:	composite

outline

The Outline owned by the Database.

class:	Outline
multiplicity:	exactly one

6.4.2 OutlineReferencesDimensions

An Outline organizes the Dimensions contained in its Database.

Ends

outline

The Outline that organizes the Dimensions.

class:	Outline
multiplicity:	exactly one

dimension

Dimensions organized by the Outline.

class:	Dimension
multiplicity:	zero or more

6.5 OCL Representation of Essbase Constraints

[C-1] Restrict the cardinality of the namespace role to 1 so that a Database must always be owned by an Application.

context Database **inv:**

self.namespace->notEmpty and self.namespace.oclIsKindOf(Application)

[C-2] Essbase Dimensions are not composed from other Essbase dimensions.

context Dimension

inv: self.component->isEmpty

inv: self.composite->isEmpty

[C-3] The inclusion of certain DimensionedObjects is valid only for certain DimensionTypes.

context Dimension

inv: self.dimensionedObject->includes(TimeBalance) implies self.type = #ess_accounts

inv: self.dimensionedObject->includes(VarianceReporting) implies self.type = #ess_accounts

inv: self.dimensionedObject->includes(CurrencyConversion) implies self.type = #ess_currencyPartition

inv: self.dimensionedObject->includes(UDA) implies self.type = #ess_attribute

[C-4] The Outline name must be the same as the Database name.

context Outline **inv:**

self.name = self.database.name

[C-5] Restrict the cardinality of the namespace role to 1 so that a Partition must always be owned by a Database.

context Partition **inv:**

self.namespace->notEmpty and self.namespace.oclIsKindOf(Database)

[C-6] Only a source Partition can be shared.

context Partition **inv:**

self.isShared implies self.isSource

Express 7

7.1 Overview

The Oracle Express package is an extension of the Multidimensional package. It represents the physical data model for an Oracle Express Database.

The classes in this package can be used as either sources or targets of data in the data warehouse, and are available to provide a physical implementation of the OLAP data model.

7.2 Organization of the Express Package

The Express package depends on the following packages:

- org.omg::CWM::ObjectModel::Core
- org.omg::CWM::Foundation::SoftwareDeployment
- org.omg::CWM::Resource::Multidimensional

The model for the Express package is shown in five diagrams. Figure 7-1 shows the Express Database class and its containment relationships to other Express classes. Figure 7-2 shows the main Express classes and associations, while Figure 7-3 shows the associations for Express Aggregation Maps and Figure 7-4 shows the associations for Express Worksheets. Finally, Figure 7-5 shows the inheritance relationships for all the Express classes.

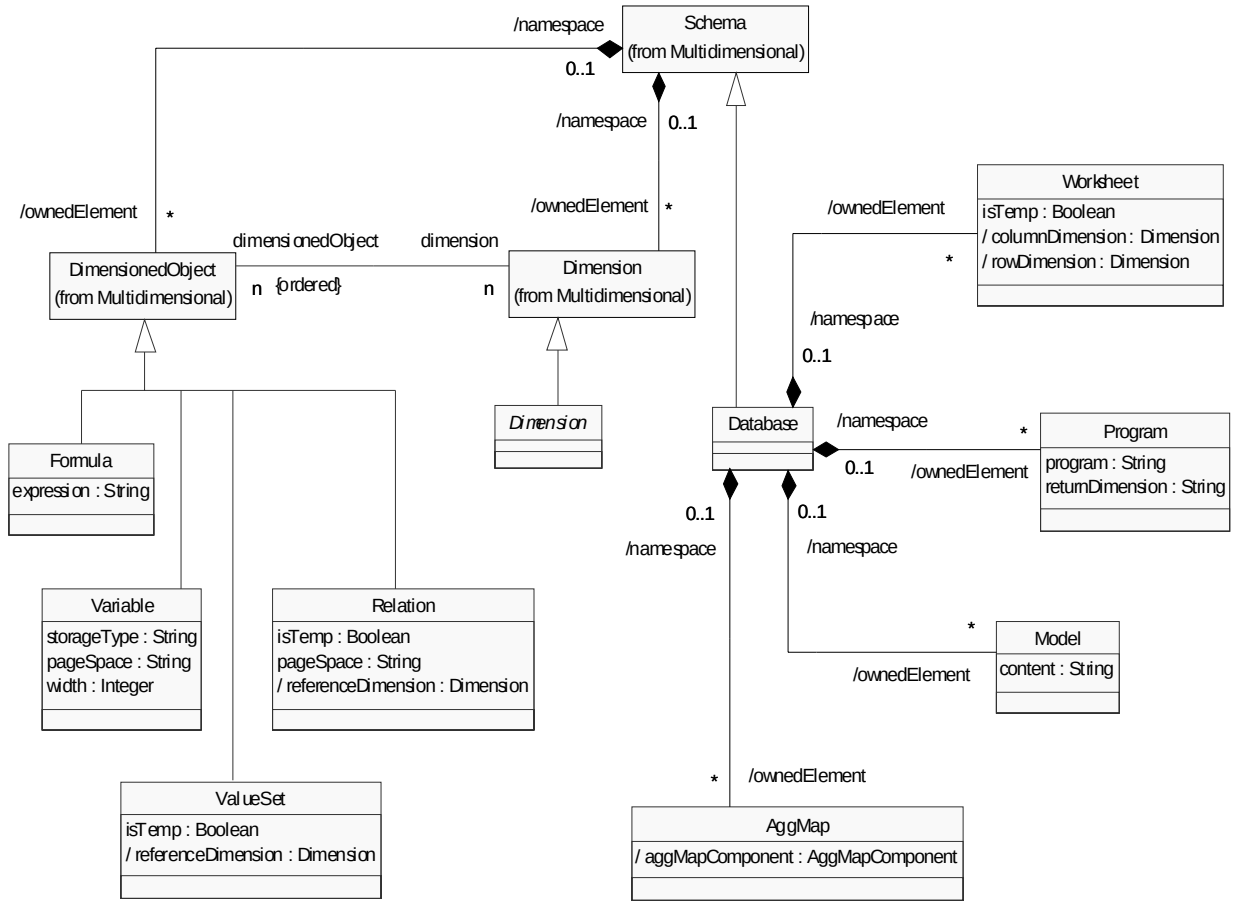

Figure 7-1 Express Database and its containment relationships

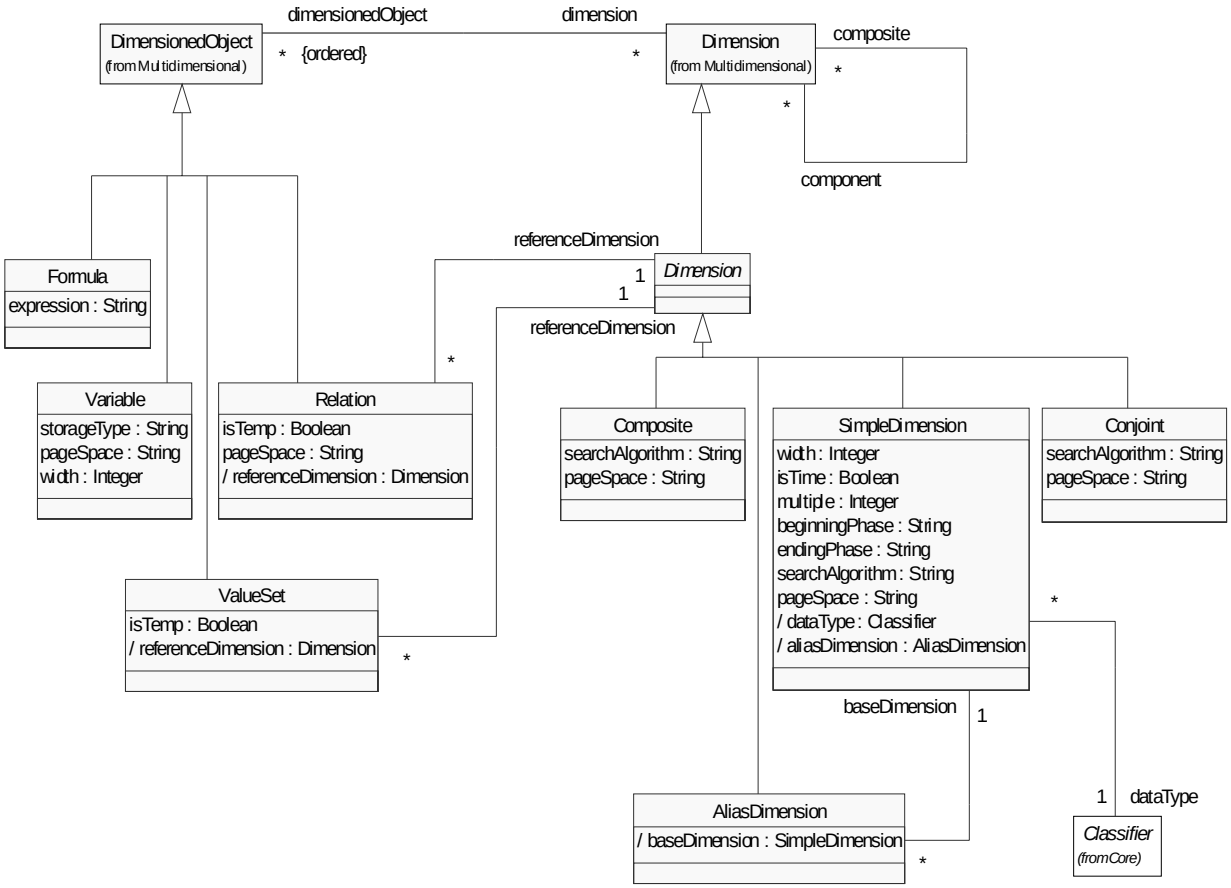

Figure 7-2 Main Express classes and associations

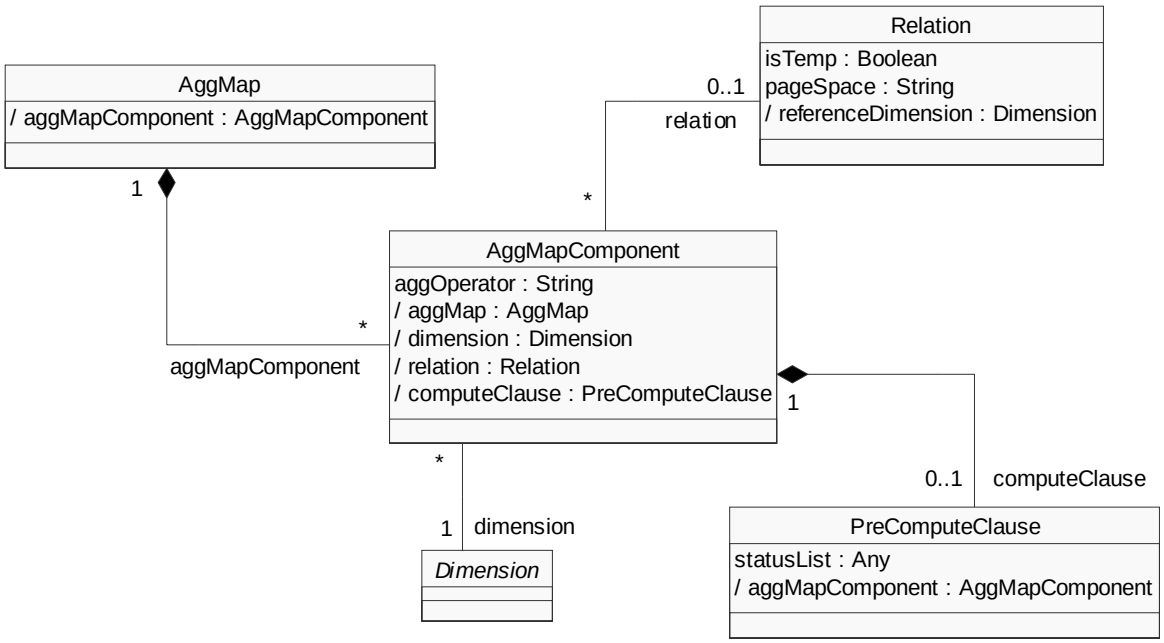

*Figure 7-3*Express Aggregation Maps

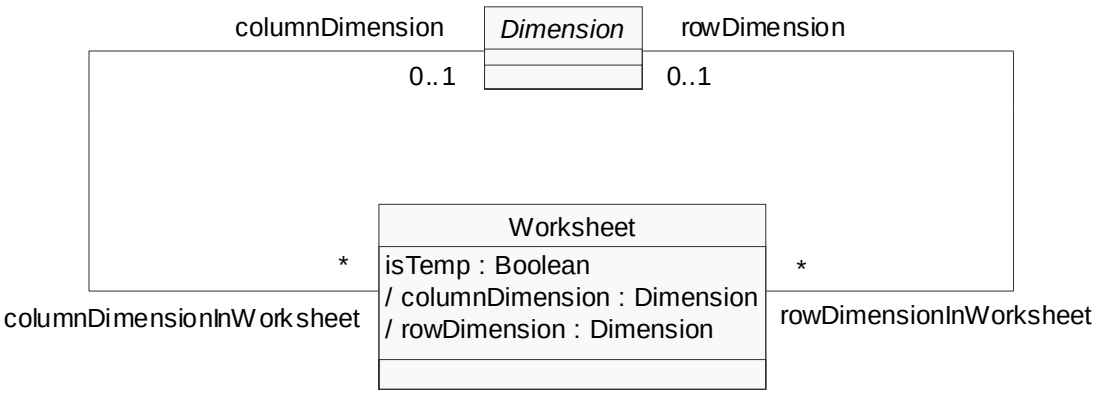

*Figure 7-4*Express Worksheets

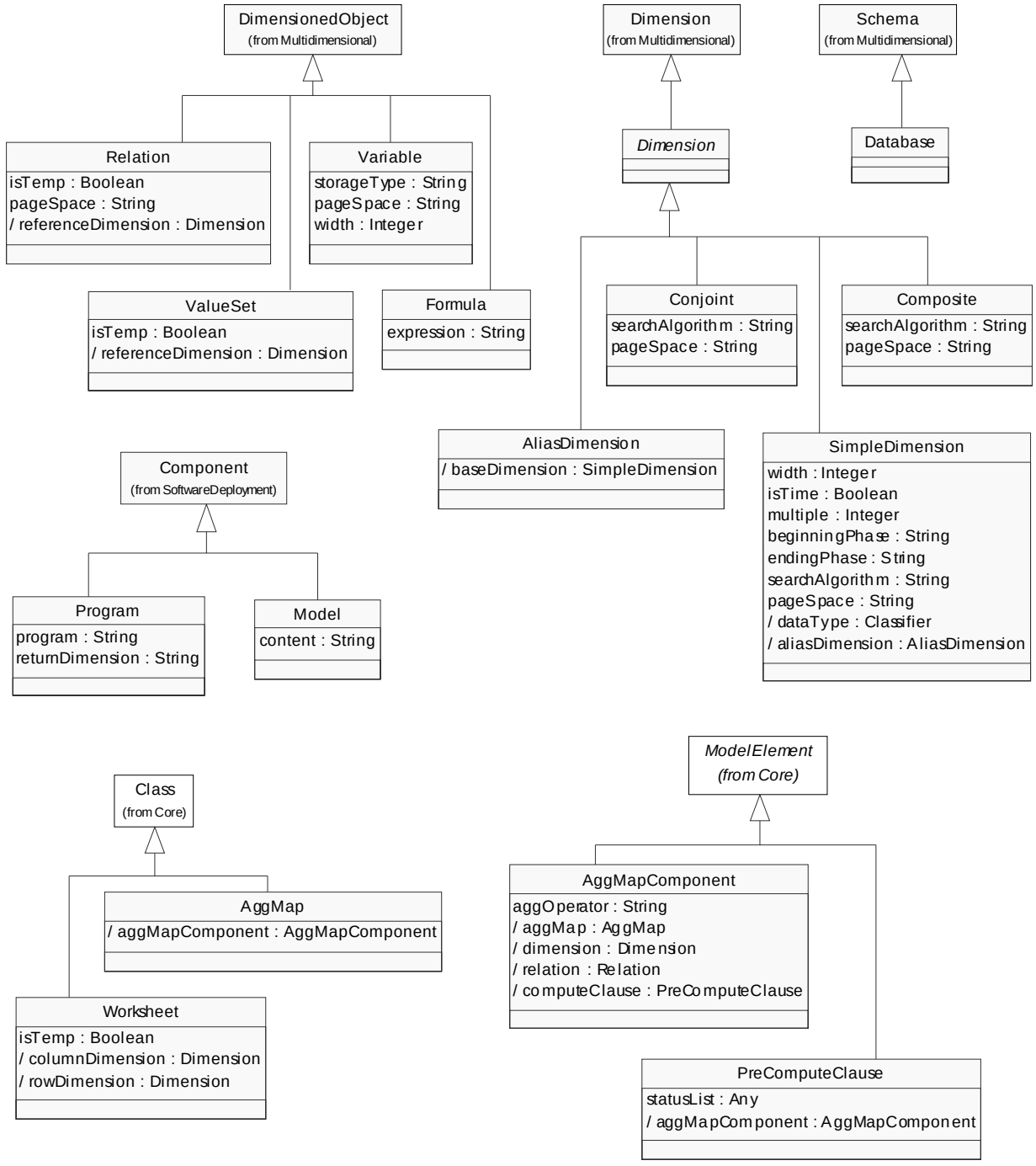

*Figure 7-5*Express inheritances

7.3 Express Classes

The Oracle Express package contains the following classes, in alphabetical order:

- AggMap
- AggMapComponent
- AliasDimension
- Composite
- Conjoint
- Database
- Dimension
- Formula
- Model
- PreComputeClause
- Program
- Relation
- SimpleDimension
- ValueSet
- Variable
- Worksheet

7.3.1 AggMap

This represents an Express aggregation map.

Superclasses

Class (from Core)

Contained Elements

AggMapComponent

References

aggMapComponent

Identifies the AggMapComponents that constitute the AggMap.

class:	AggMapComponent
defined by:	AggMapComponents::aggMapComponent
multiplicity:	zero or more
inverse:	AggMapComponent::aggMap

Constraints

An AggMap must be owned by a Database. [C-1]

7.3.2 AggMapComponent

This represents a component of an Express aggregation map.

Superclasses

ModelElement (from Core)

Contained Elements

PreComputeClause

Attributes

aggOperator

A text expression indicating the type of aggregate operation.

type:	String
multiplicity:	exactly one

References

aggMap

Identifies the AggMap that includes the AggMapComponent.

class:	AggMap
defined by:	AggMapComponents::aggMap
multiplicity:	exactly one
inverse:	AggMap::aggMapComponent

dimension

Identifies the Dimension associated with the AggMapComponent.

class:	Dimension
defined by:	AggMapComponentDimension::dimension
multiplicity:	exactly one

relation

Identifies the Relation used by the AggMapComponent.

class:	Relation
defined by:	AggMapComponentRelation::relation
multiplicity:	zero or one

computeClause

Identifies a PreComputeClause associated with the AggMapComponent.

class:	PreComputeClause
defined by:	ComputeClause::computeClause
multiplicity:	zero or one
inverse:	PreComputeClause::aggMapComponent

7.3.3 AliasDimension

This represents an Express alias dimension.

Superclasses

Dimension (from Express)

References

baseDimension

Identifies the SimpleDimension for which this is an alias.

class:	SimpleDimension
defined by:	AliasDimensionBaseDimension::baseDimension
multiplicity:	exactly one
inverse:	SimpleDimension::aliasDimension

7.3.4 Composite

This represents a physical Express composite.

Superclasses

Dimension (from Express)

Attributes

searchAlgorithm

Indicates the type of algorithm Express should use for loading and accessing the values of the Composite Dimension. The valid values are HASH, BTREE.

type: String

multiplicity: exactly one

pageSpace

If specified, this defines the type of page space to be allocated to data relating specific values of the Composite to values of its base Dimensions:
OWNSPACE specifies that the data will be stored in private page space.
SHAREDSPACE specifies that the data will be stored in the database's global page space.

type: String

multiplicity: zero or one

7.3.5 Conjoint

This represents a physical Express conjoint. This is a type of physical dimension that may be used to provide more efficient storage for sparse cubes.

Superclasses

Dimension (from Express)

Attributes

searchAlgorithm

Indicates the type of algorithm Express should use for loading and accessing the values of the Conjoint Dimension. Valid values are HASH, BTREE, NOHASH.

type: String

multiplicity: exactly one

pageSpace

If specified, this defines the type of page space to be allocated to data relating specific values of the Conjoint to values of its base Dimensions:
OWNSPACE specifies that the data will be stored in private page space.
SHAREDSPACE specifies that the data will be stored in the database's global page space.

type:	String
multiplicity:	zero or one

7.3.6 Database

This represents a physical Express database.

Superclasses

Schema (from Multidimensional)

Contained Elements

AggMap
AliasDimension
Composite
Conjoint
Formula
Model
Program
Relation
SimpleDimension
ValueSet
Variable
Worksheet

7.3.7 Dimension *abstract*

This represents a physical Express dimension.

Superclasses

Dimension (from Multidimensional)

7.3.8 Formula

This represents a physical Express formula.

Superclasses

DimensionedObject (from Multidimensional)

Attributes

expression

The calculation to be performed to produce values when you use the Formula. It can be any valid Express expression, including a constant or the name of a Variable.

type:	String
multiplicity:	exactly one

7.3.9 Model

This represents a physical Express model.

Superclasses

Component (from SoftwareDeployment)

Attributes

content

An Express representation of the content of the Model.

type:	String
multiplicity:	exactly one

Constraints

A Model must be owned by a Database. [C-2]

7.3.10 PreComputeClause

This represents a pre-compute clause for an Express aggregation map.

Superclasses

ModelElement (from Core)

Attributes

statusList

The status of the dimension to aggregate.

type:	Any
multiplicity:	exactly one

References

aggMapComponent

Identifies the AggMapComponent to which the PreComputeClause relates.

class:	AggMapComponent
defined by:	ComputeClause::aggMapComponent
multiplicity:	exactly one
inverse:	AggMapComponent::computeClause

7.3.11 Program

This represents a physical Express program. The interface to the Program may be documented as an Operation associated with the Program.

Superclasses

Component (from SoftwareDeployment)

Attributes

program

An Express representation of the Program.

type:	String
multiplicity:	exactly one

returnDimension

If present, this specifies that when the Program is called as a function it returns a value of the named Dimension.

type:	String
multiplicity:	zero or one

Constraints

A Program must be owned by a Database. [C-3]

7.3.12 Relation

This represents a reference from one or more Dimensions to another Dimension.

Superclasses

DimensionedObject (from Multidimensional)

Attributes

isTemp

If set, this indicates that values of the Relation are only temporary, and will be discarded at the end of each Express session.

type:	Boolean
multiplicity:	exactly one

pageSpace

This identifies the type of page space in which data associated with the Relation will be stored:
OWNSPACE specifies that the data will be stored in private page space associated with the Relation.
SHAREDSPACE specifies that the data will be stored in the database's global page space.

type:	String
multiplicity:	zero or one

References

referenceDimension

Identifies the Dimension referenced by the Relation.

class:	Dimension
defined by:	RelationReferenceDimension::referenceDimension
multiplicity:	exactly one

7.3.13 SimpleDimension

This represents an Express simple dimension.

Superclasses

Dimension (from Express)

Attributes

width

If specified, this defines a fixed width, in bytes, for the storage area for each value of the SimpleDimension. Fixed widths can be specified for TEXT dimensions only. Valid width values are 1 through 256.

type:	Integer
multiplicity:	zero or one

isTime

If set, indicates that the SimpleDimension is a time dimension.

type:	Boolean
multiplicity:	exactly one

multiple

This may be used for SimpleDimensions whose data type is WEEK or MONTH, to define time periods that span a multiple number of weeks or months.
With the WEEK data type, multiple can be an integer from 2 to 52.
With the MONTH data type, multiple can be 2, 3, 4, or 6.

type:	Integer
multiplicity:	zero or one

beginningPhase

This may be used for any time data type except DAY, to specify the beginning phase of the Dimension.
For single weeks, beginningPhase can be a day of the week or a date. For multiple weeks, beginningPhase must be a date. For months, quarters, or years, beginningPhase must be a month, expressed as a month name or as a date.

type:	String
multiplicity:	zero or one

endingPhase

This may be used for any time data type except DAY, to specify the ending phase of the Dimension.
For single weeks, endingPhase can be a day of the week or a date. For multiple weeks, endingPhase must be a date. For months, quarters, or years, endingPhase must be a month, expressed as a month name or as a date.

type:	String
multiplicity:	zero or one

searchAlgorithm

Indicates the type of algorithm Express should use for loading and accessing the values of the SimpleDimension. The valid values are HASH, BTREE.

type:	String
multiplicity:	exactly one

pageSpace

If specified, this defines the type of page space to be allocated to values of the SimpleDimension:
OWNSPACE specifies that the data will be stored in private page space.
SHAREDSPACE specifies that the data will be stored in the database's global page space.

type:	String
multiplicity:	zero or one

References

dataType

Identifies the SimpleDimension's data type.

class:	Classifier
defined by:	SimpleDimensionDataType::dataType
multiplicity:	exactly one

aliasDimension

Identifies any AliasDimensions associated with the SimpleDimension.

class:	AliasDimension
defined by:	AliasDimensionBaseDimension::aliasDimension
multiplicity:	zero or more
inverse:	AliasDimension::baseDimension

7.3.14 ValueSet

This represents a physical Express value set.

Superclasses

DimensionedObject (from Multidimensional)

Attributes

isTemp

If set, this indicates that values in the ValueSet are only temporary, and will be discarded at the end of each Express session.

type:	Boolean
multiplicity:	exactly one

References

referenceDimension

Identifies the Dimension whose values are to be stored in the ValueSet.

class:	Dimension
defined by:	ValueSetReferenceDimension::referenceDimension
multiplicity:	exactly one

7.3.15 Variable

This represents a physical Express variable.

Superclasses

DimensionedObject (from Multidimensional)

Attributes

storageType

The type of storage to use for the Variable. The valid values are: TEMP, INPLACE, PERMANENT.

type:	String
multiplicity:	exactly one

pageSpace

If specified, this defines the type of page space in which data associated with the Variable will be stored:
OWNSPACE specifies that the data will be stored in private page space associated with the Variable.
SHAREDSPACE specifies that the data will be stored in the database's global page space.

type: String

multiplicity: zero or one

width

If specified, this defines a fixed width, in bytes, for the storage area for each value of a Variable. Fixed widths can be specified only for dimensioned TEXT and INTEGER Variables.

type: Integer

multiplicity: zero or one

7.3.16 Worksheet

This represents a physical Express worksheet.

Superclasses

Class (from Core)

Contained Elements

Attribute

Attributes

isTemp

If set, this indicates that values in the Worksheet are only temporary, and will be discarded at the end of each Express session.

type: Boolean

multiplicity: exactly one

References

columnDimension

Identifies a Dimension used as the column dimension of the Worksheet.

class:	Dimension
defined by:	WorksheetColumnDimension::columnDimension
multiplicity:	zero or one

rowDimension

Identifies a Dimension used as the row dimension of the Worksheet.

class:	Dimension
defined by:	WorksheetRowDimension::rowDimension
multiplicity:	zero or one

Constraints

A Worksheet must be owned by a Database. [C-4]

7.4 Express Associations

The Oracle Express package contains the following associations, in alphabetical order:

- AggMapComponentDimension
- AggMapComponentRelation
- AggMapComponents
- AliasDimensionBaseDimension
- ComputeClause
- RelationReferenceDimension
- SimpleDimensionDataType
- ValueSetReferenceDimension
- WorksheetColumnDimension
- WorksheetRowDimension

7.4.1 AggMapComponentDimension

Identifies the associated Dimension for an AggMapComponent.

Ends

aggMapComponent

Identifies the AggMapComponent.

class:	AggMapComponent
multiplicity:	zero or more

dimension

Identifies the Dimension associated with the AggMapComponent.

class:	Dimension
multiplicity:	exactly one

7.4.2 AggMapComponentRelation

Identifies a Relation used by an AggMapComponent.

Ends

aggMapComponent

Identifies the AggMapComponent.

class:	AggMapComponent
multiplicity:	zero or more

relation

Identifies the Relation used by the AggMapComponent.

class:	Relation
multiplicity:	zero or one

7.4.3 AggMapComponents *protected*

Identifies the AggMapComponents that constitute an AggMap.

Ends

aggMap

Identifies the AggMap.

class:	AggMap
multiplicity:	exactly one
aggregation:	composite

aggMapComponent

Identifies an AggMapComponent that forms part of the AggMap.

class:	AggMapComponent
multiplicity:	zero or more

7.4.4 AliasDimensionBaseDimension *protected*

Associates AliasDimensions with the SimpleDimension on which they are based.

Ends

baseDimension

Identifies the SimpleDimension on which an AliasDimension is based.

class:	SimpleDimension
multiplicity:	exactly one

aliasDimension

Identifies an AliasDimension that is based on the SimpleDimension.

class:	AliasDimension
multiplicity:	zero or more

7.4.5 ComputeClause *protected*

Identifies a PreComputeClause associated with an AggMapComponent.

Ends

aggMapComponent

Identifies the AggMapComponent.

class:	AggMapComponent
multiplicity:	exactly one
aggregation:	composite

computeClause

Identifies the PreComputeClause associated with the AggMapComponent.

class:	PreComputeClause
multiplicity:	zero or one

7.4.6 RelationReferenceDimension

Associates Express Relations with the Dimension they reference.

Ends

relation

Identifies the Express Relations that reference the Dimension.

class:	Relation
multiplicity:	zero or more

referenceDimension

Identifies the Dimension referenced by the Relation.

class:	Dimension
multiplicity:	exactly one

7.4.7 SimpleDimensionDataType

Identifies the data type for a SimpleDimension.

Ends

SimpleDimension

Identifies the SimpleDimension.

class:	SimpleDimension
multiplicity:	zero or more

dataType

Identifies the data type for the SimpleDimension.

class:	Classifier (from UML Core)
multiplicity:	exactly one

7.4.8 ValueSetReferenceDimension

Associates Express ValueSets with the Dimension whose values are to be stored in the ValueSet.

Ends

valueSet

Identifies the Express ValueSets that reference a Dimension.

class:	ValueSet
multiplicity:	zero or more

referenceDimension

Identifies the Dimension whose values are to be stored in the ValueSet.

class:	Dimension
multiplicity:	exactly one

7.4.9 WorksheetColumnDimension

Identifies a Dimension used as the column dimension in a physical Express worksheet structure.

Ends

columnDimensionInWorksheet

Identifies the Worksheets using the Dimension as a column dimension.

class:	Worksheet
multiplicity:	zero or more

columnDimension

Identifies a Dimension used as the column dimension of the Worksheet.

class:	Dimension
multiplicity:	zero or one

7.4.10 WorksheetRowDimension

Identifies a Dimension used as the row dimension in a physical Express worksheet structure.

Ends

rowDimensionInWorksheet

Identifies the Worksheets using the Dimension as a row dimension.

class:	Worksheet
multiplicity:	zero or more

rowDimension

Identifies a Dimension used as the row dimension of the Worksheet.

class:	Dimension
multiplicity:	zero or one

7.5 OCL Representation of Express Constraints

[C-1]	An AggMap must be owned by a Database
context	AggMap
inv:	self.namespace->size = 1 and self.namespace.oclIsKindOf(Database)

[C-2]	A Model must be owned by a Database
context	Model
inv:	self.namespace->size = 1 and self.namespace.oclIsKindOf(Database)

[C-3]	A Program must be owned by a Database
context	Program
inv:	self.namespace->size = 1 and self.namespace.oclIsKindOf(Database)

[C-4]	A Worksheet must be owned by a Database
context	Worksheet
inv:	self.namespace->size = 1 and self.namespace.oclIsKindOf(Database)

InformationSet *8*

8.1 Overview

The InformationSet package contains the metamodel of information sets that is an extension of the OLAP package.

An important aspect of data warehousing is the collection of data from external resources using, for example, application generated reports, questionnaires or surveys. To allow for inter-operability of tools supporting data collection, the metadata identifying the data to be collected must be defined, together with metadata that can be used to ensure accuracy and validity of data.

The InformationSet package is designed to enable interchange of common metadata about InformationSet structures, rules (e.g. validation, calculation), and, possibly, visual renderings.

The characteristics of the InformationSet are:

- the definition of logical structures that define the data to be collected can be both single and multi-dimensional (e.g. value of export by country and product)
- data can be derived both from either from non-human sources or from human sources (e.g. questionnaire, report form).
- instruments for data collection need to be designed in such a way that relevant and accurate data is collected, and in some instances this means that the designer needs control over the visual rendering of the collection instrument. Validation and Navigation within the InformationSet is also required in order to facilitate the collection/retrieval of accurate data.

The InformationSet package contains metamodel elements that support the following functions:

- Semantic definition of the InformationSet and its constituent parts
- Visual rendering supporting different media

- Validation, navigation and calculation based on data values entered/retrieved
- Via OLAP, the links to Nomenclatures (LevelBasedHierarchy) that give the valid codes and multi lingual labels for the Dimensions

The InformationSet is a domain specific extension of the OLAP model with some additional classes to support specific requirements of the InformationSet (e.g. validation, rendering). All information to be collected can be identified in terms of the OLAP model, thus supporting the definition of both multidimensional and unidimensional Segments (the OLAP Cube).

In addition to the definition of multi-dimensional structures that comprise the logical InformationSet, there is a need to:

- ensure accuracy of the data by applying validation and navigation expressions
- derive values from other values in the InformationSet
- provide visual rendering capability to support data collection using visual media such as forms or screens

8.2 Organization of the InformationSet Package

The InformationSet package depends on the following packages:

- org.omg::CWM::ObjectModel::Core
- org.omg::CWM::Foundation::BusinessInformation
- org.omg::CWM::Foundation::Expressions
- org.omg::CWM::Analysis::OLAP
- org.omg::CWM::Analysis::InformationVisualization

The package is illustrated with two class diagrams:

- InformationSet Inheritance
- InformationSet Relationships

8.2.1 InformationSet Inheritance

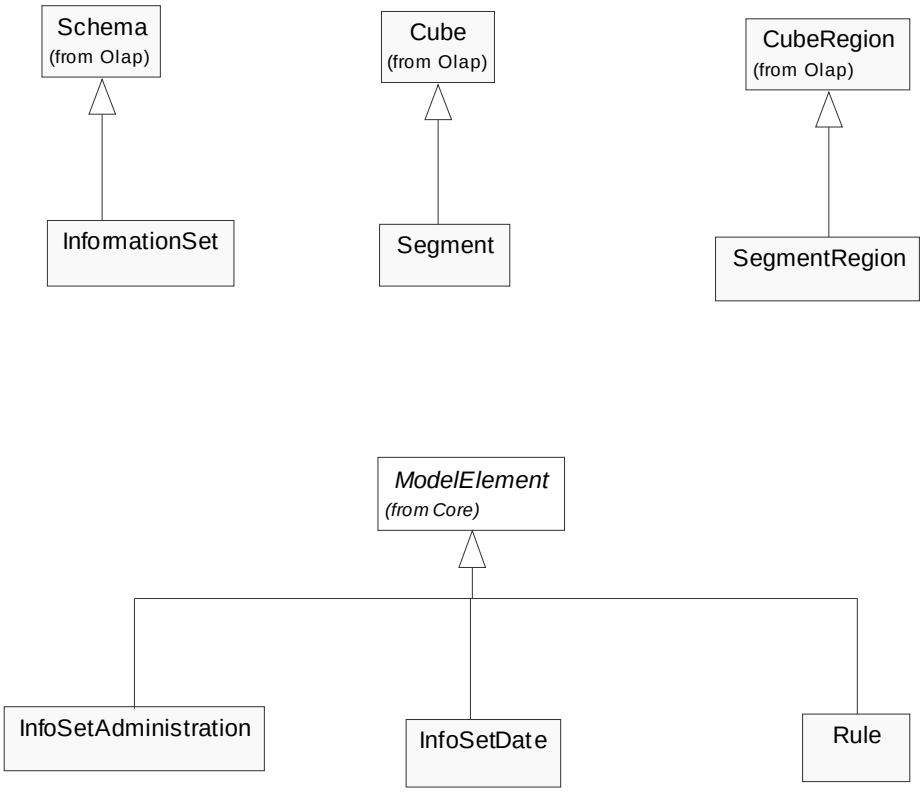

Figure 8-1 InformationSet Inheritance

The core classes in the InformationSet package *(InformationSet, Segment, SegmentRegion*) are derived from the OLAP package classes of Schema, Cube, and CubeRegion. This means that each part of an InformationSet (the Segment) is defined in terms of a multidimensional structure of Dimensions.

Rule, InfoSetAdministration, and *InfoSetDate* are derived from ModelElement.

8.2.2 InformationSet Relationships

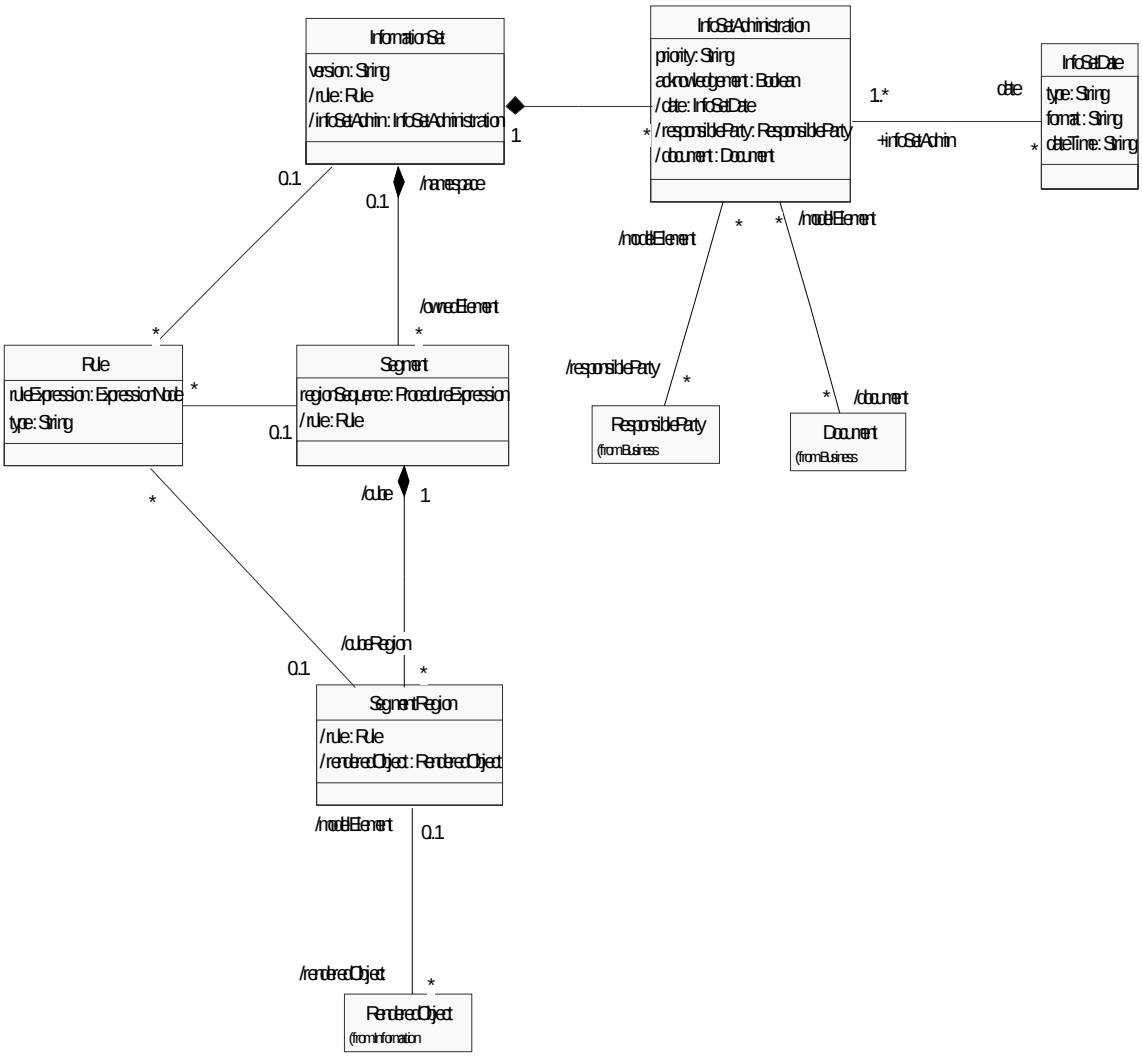

Figure 8-2 InformationSet Relationships

InformationSet is the logical container of all elements comprising the InformationSet metamodel. It is the root element of the metamodel hierarchy and marks the entry point for navigating the metamodel.

An InformationSet contains *Segments*. The Segment is a sub class of the OLAP Cube and inherits the association to Dimension through CubeDimAssoc. This association yields a set of unique Dimensions that comprise a multidimensional object and this gives the Segment the ability to be multidimensional.

The Segment comprises one or more *SegmentRegion*, and, if the order of these is significant, they can be ordered and specified as such. The SegmentRegion is a subclass of the OLAP CubeRegion. The SegmentRegion is a definition of a part of the Segment, specified in terms of the Dimensions and Dimension value combinations that comprise the SegmentRegion. This is achieved via the OLAP association between CubeRegion and MemberSelGrp and onward to MemberSelection or CodedLevel. In effect, the SegmentRegion is a "mask" or "slice" of the complete Segment. This mask may expose the entire Segment, or it may expose just one cell in the multidimensional structure, or any combination in between. The SegmentRegion can be used to:

- specify the valid combinations of Dimension values for which target data are to be collected or extracted
- identify a cell or cells for which specific *Rules* apply (e.g. navigation, validation or calculation is specified)
- identify the part of a Segment for which a specific visual rendering is specified - the high level rendering requirements can be specified using the *RenderedObject* which is part of the InformationVisualization package

A SegmentRegion can overlap with another SegmentRegion for the same Segment, and the SegmentRegion can expose the whole of the Segment. Any ambiguities resulting from overlapping SegmentRegions are resolved by making use of the ability to order the SegmentRegions. This makes it possible, for instance, to impose a specific validation that is relevant to all of the Segment, and supplement this with a validation that is specific to a smaller part of the Segment.

One or more *Rules* can be attached to InformationSet, Segment, SegmentRegion. This makes it possible to apply rules at any level:

- to the whole InformationSet, including rules that span the data in two Segments
- to a Segment, including rules that span the data in two SegmentRegions
- to a SegmentRegion

The application of rules to the extraction of data from data resources makes it possible to specify validation of the data which can then be undertaken by an appropriate tool. This makes it possible to validate the data at the earliest possible time, including immediately after extraction or, if the InformationSet is rendered on a screen, after data has been entered.

The InformationSet can have *InfoSetAdministration* details associated with it. Specific references to ResponsibleParty and Document (from the BusinessInformation package) have been included. The InfoSetAdministration can have *InfoSetDate* associated with it, giving the ability to associate one or more dates with the InformationSet (e.g. collection date), as defined by the InfoSetDate.type attribute.

8.3 InformationSet Classes

8.3.1 InformationSet

InformationSet contains all elements comprising an InformationSet database model.

Superclasses

Schema

Contained Elements

Segment

InfoSetAdministration

Attributes

version

The version of an Information Set.

type:	String
multiplicity:	exactly one

References

rule

References the Rule owned by the InformationSet.

class:	Rule
defined by:	InformationSetReferencesRule::rule
multiplicity:	zero or more

infoSetAdmin

References the InfoSetAdministration owned by InformationSet.

class:	InfoSetAdministration
defined by:	InformationSetReferencesInfoSetAdministration::infoSetAdmin
multiplicity:	zero or more

8.3.2 InfoSetAdministration

This class represents administrative details of an Information Set.

Superclasses

ModelElement

Attributes

priority

The priority of an Information Set.

type	String
multiplicity	exactly one

acknowledgement

If true, an acknowledgement is requested.

type	Boolean
multiplicity	exactly one

References

date

References the InfoSetDate owned by InfoSetAdministration.

class:	InfoSetDate
defined by:	InfoSetAdministrationReferencesInfoSetDates::date
multiplicity:	zero or more

responsibleParty

References the ResponsibleParty owned by InfoSetAdministration.

class:	ResponsibleParty
defined by:	ModelElementResponsibility::responsibleParty
multiplicity:	zero or more

document

References the Document owned by InfoSetAdministration.

class:	Document
defined by:	DocumentDescribes::document
multiplicity:	zero or more

8.3.3 InfoSetDate

This class represents dates relevant to an InformationSet such as date of dissemination or receipt. It can have different formats.

Superclasses

ModelElement

Attributes

type

Type of date, e.g. date of creation, validity period.

type:	String
multiplicity:	exactly one

format

Format of date, time or period.

type:	String
multiplicity:	exactly one

dateTime

A date, time or period.

type:	String
multiplicity:	exactly one

8.3.4 Rule

This is a rule that performs one or more of the following:

1) Defines the validation required for data extracted from the data resource part of the InformationSet, Segment or SegmentRegion. This can include the requirement or otherwise of other Segment or SegmentRegions based on the value of the data extracted from this Segment or SegmentRegion.

2) Defines a calculation that can be used to derive data values, based on data values that are part of the InformationSet, Segment or SegmentRegion.

Superclasses

ModelElement

Attributes

ruleExpression

This is the rule, which can be an expression or an expression tree.

type:	ExpressionNode
multiplicity:	exactly one

type

This allows the Rule to be categorized - for example validation, calculation.

type:	String
multiplicity:	exactly one

Constraints

One instance of Rule can be associated with only one of InformationSet, Segment, SegmentRegion. [C-1]

8.3.5 Segment

Segment represents a multidimensional structure. A Segment is defined by a collection of unique Dimensions from the InformationSet. Each unique combination of members in the Cartesian product of the Segment's Dimensions identifies precisely one data cell within the multidimensional structure.

Note that a logical segment is "defined" by a collection of unique Dimensions from the InformationSet, and is "described" by a collection of one or more Segment Regions.

Superclasses

Cube

Contained Elements

SegmentRegion

Attributes

regionSequence

Specifies the sequence of the SegmentRegion.

type:	ProcedureExpression
multiplicity:	exactly one

References

rule

References the Rule owned by Segment.

class:	Rule
defined by:	SegmentReferencesRule::rule
multiplicity:	zero or more

8.3.6 SegmentRegion

SegmentRegion represents a sub-set of a Segment. A SegmentRegion may be used for exposing a subset of the dimensionality of a Segment. The Member Selections comprising a SegmentRegion always collectively define a subset of the total dimensionality of the associated Segment.

Superclasses

CubeRegion

References

rule

References the Rule owned by SegmentRegion.

class:	Rule
defined by:	SegmentRegionReferencesRule::rule
multiplicity:	zero or more

renderedObject

References the RenderedObject owned by SegmentRegion.

class:	RenderedObject
defined by:	RenderedObjectsReferenceModelElement::renderedObject
multiplicity:	zero or more

8.4 InformationSet Associations

8.4.1 InformationSetReferencesInfoSetAdministration

This association relates an Information Set to Administrative details.

Ends

informationSet

InformationSet for adminstrative details.

class:	InformationSet
multiplicity:	one

infoSetAdmin

Administrative details for an InformationSet.

class:	InfoSetAdministration
multiplicity:	zero or more

8.4.2 InformationSetReferencesRule

An InformationSet may reference one or more Rule.

Ends

informationSet

The InformationSet for a Rule

class:	InformationSet
multiplicity:	zero or one

rule

The Rule for an InformationSet.

class:	Rule
multiplicity:	zero or more

8.4.3 InfoSetAdministrationReferencesInfoSetDates

Allows dates to be specified for Information Set details.

Ends

date

Date for administration details.

class:	InfoSetDate
multiplicity:	zero or more

infoSetAdmin

Adminstration details owning the date.

class:	InfoSetAdministration
multiplicity:	one or more

8.4.4 SegmentReferencesRule

A Segment may reference one or more Rule.

Ends

segment

The Segment for a Rule

class:	Segment
multiplicity:	zero or one

rule

The Rule for a Segment

class:	Rule
multiplicity:	zero or more

8.4.5 SegmentRegionReferencesRule

A SegmentRegion may reference one or more Rules.

Ends

segmentRegion

The SegmentRegion for a Rule.

class:	SegmentRegion
multiplicity:	zero or one

rules

The Rules for a SegmentRegion.

class:	Rule
multiplicity:	zero or more

8.5 OCL Representation of InformationSet Constraints

[C-1] One instance of Rule can be associated with only one of InformationSet, Segment, SegmentRegion.

context Rule

inv: self.informationSet->isEmpty xor self.segment->isEmpty xor self.segmentRegion->isEmpty

Information Reporting 9

9.1 Overview

The CWM Information Reporting metamodel extends the CWM Information Visualization metamodel for the purpose of defining metadata representing formatted reports.

9.2 Organization of the Information Reporting Metamodel

9.2.1 Dependencies

The Information Reporting package depends on the following packages:

org.omg::CWM::ObjectModel::Core

org.omg::CWM::Foundation::DataTypes

org.omg::CWM::Analysis::Transformation

org.omg::CWM::Analysis::InformationVisualization

9.2.2 Major Classes and Associations

The major classes and associations of the Information Reporting metamodel are shown in Figure 9-1.

Report is a subclass of InformationVisualization::RenderedObject that represents a formatted report. Reports are comprised of instances of ReportGroup, which is also a subclass of RenderedObject. Thus, a Report is composed from ReportGroups, and ReportGroups may be composed recursively from other ReportGroups. Ultimately, the recursive definition of a ReportGroup must terminate with one or more ReportFields as leaf-level components. ReportField represents single-valued attributes of the Report

that are individually mapped to data sources (e.g., report queries) and rendered (i.e., formatted) by associated instances of InformationVisualization::Rendering.

Related Reports may be grouped together into a ReportPackage, a subclass of InformationVisualization::RenderedObjectSet. A Report may also have a related ReportExecution, which is a subclass of TranformationMap that relates the Report to both its data sources and procedures required to run the Report. Each ReportGroup may also specify a separate QueryExpression which is evaluated to yield the contents of fields within the ReportGroup. This enables the specification of both derived, as well as retrieved, data values for report fields.

Note that the Information Reporting metamodel makes extensive use of associations and attributes inherited from Information Visualization. For example, the formula attribute inherited from RenderedObject would be used by an instance of ReportGroup to indicate its positioning within the overall report layout. The inherited association "CompositesReferenceComponents" is used to compose Reports from ReportGroups, as well as to compose ReportGroups recursively from other ReportGroups and ReportFields. The inherited association "NeighborsReferenceNeighbors" may be used to specify topological relationships between instances of ReportGroup at the same level of composition.

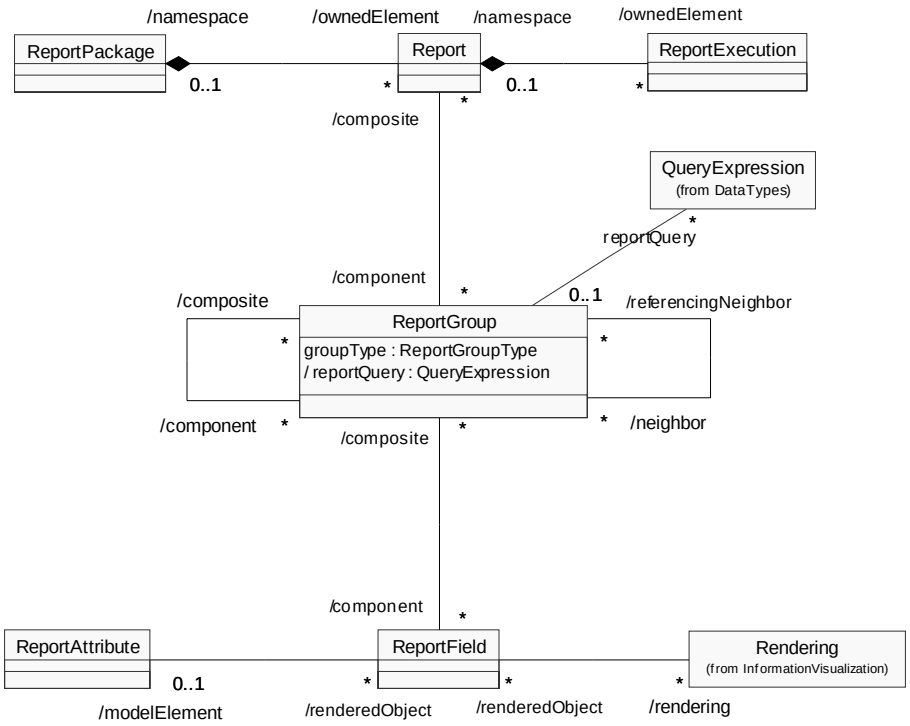

Figure 9-1 CWM Information Reporting Metamodel

9.3 Inheritance of the Information Reporting Metamodel

The inheritance of classes of the Information Reporting metamodel from classes of packages it depends on is shown in Figure 9-2 below.

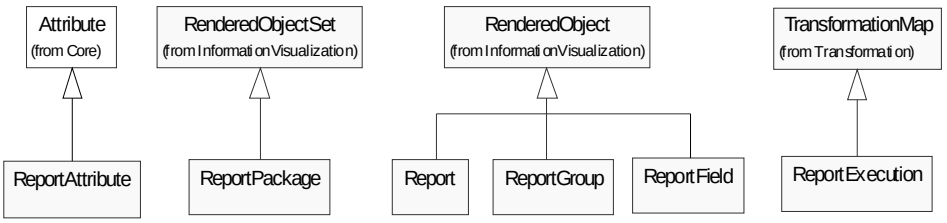

Figure 9-2 CWM Information Reporting Metamodel: Inheritance

9.4 Information Reporting Classes

9.4.1 Report

Report is a subclass of RenderedObject that defines a two-dimensional, formatted report. A Report may be rendered as a printed report, in HTML, or as a dynamic, online report.

Superclasses

RenderedObject

Contained Elements

ReportExecution

References

reportExecution

References the collection of ReportExecutions owned by a Report.

class:	ReportExecution
defined by:	ReportOwnsReportExecutions::reportExecution
multiplicity:	zero or more
inverse:	ReportExecution::report

Constraints

Reports generally do not have neighbor relationships with other reports. [C-1]

9.4.2 ReportAttribute

ReportAttribute is a subclass of UML Attribute that generally defines a ReportField. Note that ReportAttributes may be re-used in the definitions of multiple ReportFields.

Superclasses

Attribute

9.4.3 ReportExecution

ReportExecution is a subclass of TransformationMap that defines any necessary mappings and procedures for actually generating an instance of a Report.

Superclasses

TransformationMap

9.4.4 ReportField

ReportField defines a specific field within a ReportGroup.

Superclasses

RenderedObject

Constraints

A ReportField is associated with precisely one ReportGroup. [C-2]

A ReportField may not have components. [C-3]

9.4.5 ReportGroup

ReportGroup defines a grouping of fields on a report.

Superclasses

RenderedObject

Attributes

groupType

Specifies the type of a ReportGroup

type:	ReportGroupType (header | footer | detail | other)
multiplicity:	exactly one

References

reportQuery

References the collection of QueryExpressions owned by a ReportGroup.

class:	QueryExpression
defined by:	ReportGroupReferenceQueryExpressions ::reportQuery
multiplicity:	zero or more

Constraints

A ReportGroup is associated with precisely one Report. [C-4]

9.4.6 ReportPackage

Defines a grouping of related reports.

Superclasses

RenderedObjectSet

9.5 Information Reporting Associations

9.5.1 ReportGroupReferencesQueryExpressions

A ReportGroup may reference one or more instances of QueryExpression.

Ends

reportQuery

QueryExpressions referenced by the ReportGroup.

class:	QueryExpression
multiplicity:	zero or more

reportGroup

The ReportGroup referencing QueryExpressions.

class:	ReportGroup
multiplicity:	zero or one

9.6 OCL Representation of Information Reporting Constraints

[C-1] Reports generally do not have neighbor relationships with other reports.

context Report

inv: self.neighbor->isEmpty

inv: self.referencingNeighbor->isEmpty

[C-2] A ReportField is associated with precisely one ReportGroup.

context ReportField **inv:**

self.composite->size = 1

[C-3] A ReportField may not have components.

context ReportField **inv:**

self.component->isEmpty

[C-4] A ReportGroup is associated with precisely one Report.

context ReportGroup **inv:**

self.composite->size = 1

References

Non-Normative

[COBOL] ANSI X3.23-1985, *American National Standard for Information Systems - Programming Language - COBOL*, 1985

[OIM] MDC Open Information Model, Version 1.0, 1999