General Entity Overview

Introduction

The purpose of this document is to describe the OFBiz entities in various components and their design. A brief overview of each component will be presented which will include a description of the entities in the component and their relations to other entities.

There are some files that go along with the definitions of these entities. These files are located in the 'entitydef' folder in the **datamodel** component for the base OFBiz components. For special purpose components the entity definitions are in the 'entitydef' folder of the particular component. For an explanation of what these entity types are and how they are used, see the Extensibility Pattern section of this document.

Another important set of files is in the "data" folder of each component. These are XML Entity Engine Import files that contain seed and demo data related to the various entities in the data model.

Extensibility Pattern

There is one pattern that you will see repeated often in the OFBiz data model that warrants some explanation. This pattern is used to make the data model more flexible and eliminate a great many tables that might otherwise have to exist. It aids in the conceptualization of the use of an entity that follows this pattern. It also provides a method for run-time extensibility of the entities that use this pattern.

For a given Entity there are 3 or 4 related entities that are used to implement this pattern. These are:

- EntityClass (optional),
- EntityType,
- EntityAttribute, and
- EntityTypeAttr.

Each Entity instance can have one or more types which are described by EntityType. If an Entity can have multiple types the EntityClass, or Entity Classification entity is used to implement the many-to-many relation between Entity and EntityType. Otherwise no EntityClass is necessary and Entity will have an entityTypeId field.

EntityType

EntityType is used to describe an Entity. A given EntityType can inherit features from a parent EntityType if one is specified in the parentTypeId field of a given EntityType instance. If a table is associated with a given EntityType instance that has the same name as the entityTypeId field value then the hasTable field should have the value 'Y', otherwise it should have the value 'N'. A description field is provided for a short description of an EntityType instance.

EntityAttribute

EntityAttribute is used to store instances of name-value pair attributes for a given Entity instance. An attribute can be used in place of a column on the Entity table, especially when the attribute does not apply to all types of Entities. Attributes can also be used ad-hoc for any name-value pair information that applies to a given Entity instance. If many attributes apply to a given EntityType it may be best to create a separate entity to hold those attributes, and have that entity be associated with the EntityType instance by naming it the same as the entityTypeId and setting the hasTable field to 'Y' on the EntityType instance. This will be faster than repeatedly querying a collection of attributes for a given Entity instance.

EntityTypeAttr

EntityTypeAttr is used to specify which attributes should exist for an Entity instance that corresponds to a given entityTypeId. The entityTypeId and name field combinations are listed to specify this. For example, the EntityType with entityTypeId 'BOX' should always have attributes named 'HEIGHT', 'WIDTH', 'DEPTH', and 'COLOR', so an instance of EntityTypeAttr will exist for each of these attributes.

There are variations on this pattern for certain entities, but this description is generally applicable in this data model. Note that the default type seed data for the data model is in various *Data.xml files, generally in the data directory in each component.

Deprecated Entities

Entities prefixed with "Old" are deprecated. The underlying table names remain unchanged by explicitly specifying the corresponding table name (see the table-name attribute in the entity definition).

When changing an entity significantly, and especially when changing the primary key, always deprecate the existing entity and create a new one so that an upgrade path for existing databases is possible. To facilitate this in addition to deprecating the old entity (prefixing with "Old") and defining the new one you should always create a service to move data from the old entity to the new one. The service doesn't have to figure out when to run itself (which is tough to figure out and generally not reliable), and generally should be made to be run manually.

When deprecating an entity make sure to search for all references to that entity and change them to use the new entity. At the end of the process the only reference to the deprecated entity, and with the "Old" prefix of course, should be the migration service.

When changing the name of a field, or deprecating and replacing a field that does not require deprecation of the entire entity, then follow the same pattern leaving the old field there: a "old" prefix added to the field name, change the original first letter to upper case, and specify the column-name for the field so it is the same as the original field name. For example if you are changing the "uomId" field to a new name then change the field name to "oldUomId" and specify a column-name of "UOM_ID". Just as when replacing and entity, make sure to write a service to move the data from the old field to the new one.

Whenever doing any of these sorts of changes that require additional steps to update a production server, ALWAYS add an entry on this page: Revisions Requiring Data Migration - upgrade ofbiz

Packages

Common

The Common package is meant to contain entities that are for general use and may be used by entities in many other packages.

common.geo

Geo GeoType GeoAssoc GeoAssocType

common.period

PeriodType StandardTimePeriod

common.status

Status StatusType

common.uom

Uom UomType UomConversion

Party

The Party package is meant to contain entities related to the Party entity. There are many types of parties, and a lot of data that is associated with a party or with parties, and that data is defined in this package. Parties include persons, organizations, and so forth.

party.party

Party PartyClassification PartyClassificationType PartyType PartyAttribute PartyTypeAttr PartyRole RoleType RoleTypeAttr PartyRelationship PriorityType PartyRelationshipType Customer PartyDataObject Person

Product

The Product package is meant to contain entities related to the Product entity. There are many types of products, and a lot of data that is associated with a product or with products, and that data is defined in this package. A Product can be a good or a service, and each may have various subtypes. Some related data that is fairly central to Product definitions includes features and categories. The Category package allows products to be organized, and the Feature package describes details of a product that may be used for pricing or in a parametric product search.
The Product package also contains information about serialized and non-serialized inventory items, and the storage of inventory in facilities which can be warehouses, stores, or whatever. There is a package for Product cost estimation, and another for Product price calculation. In addition there is a package for Product supplier related information to keep track of where a product might come from and to rate and put a priority on different suppliers of the same product.

product.product

The Product entity contains general information about a product, whatever type of product it is. Product implements the Extensibility Pattern, and can have multiple types, using the Product, ProductClass, ProductType, ProductAttribute, and ProductTypeAttr entities.

When a Product is of the type GOOD (as opposed to SERVICE) then it generally has identifications associated with it such as a SKU, an ISBN, a SKU or catalog ID from manufacturer or distributor, and so forth. These are stored in the GoodIdentification entity and the type of good id is defined by the GoodIdentificationType entity.

The ProductAssoc entity implements an association between products. The type of association is defined by the corresponding ProductAssocType entity. ProductAssoc partially implements the Extensibility Pattern by providing a type, but does not have attributes at the moment. ProductAssoc entities can be used to specify that a certain product is deprecated by another product, or can be substitued for another product. A Product can also be a 'dummy' product and simply be a collection of other products that are components of this product, or the product could be a special offer or marketing package used to promote and sell various products as a single product. Many other potential associations can be used between products. Marketing concepts such as up-sells and cross-sells can also be implemented using ProductAssoc entities.

The ProductDataObject entity is used to implement the many to many join between a Product and a DataObject. The DataObject entity is part of the Content package and is used to store text or images or other general content as a content management package. To simplify the data model the Product entity also has fields for a long description and URLs to images, but these are optional and larger organizations will want to use the content management features to control this sort of content.

product.category

ProductCategory contains information about categories of products, and can be used to organize products for cataloging, marketing, analyis, and other purposes. ProductCategory implements the Extensibility Pattern, and can have multiple types, using the ProductCategory, ProductCategoryClass, ProductCategoryType, ProductCategoryAttribute, and ProductCategoryTypeAttr entities.

A ProductCategory has a many to many relation with Product meaning that a Product can belong to many ProductCategories and a ProductCategory can contain many Products. This many to many relation is joined using the ProductCategoryMember entity which contains a reference to a Product and a ProductCategory.

A ProductCategory can be related to other product categories in a loose hierarchical structure. When I say loose I mean that it differs from a normal hierarchy in that it can have multiple parents, making it really a graph, but we still use terminology that comes from an hierarchical organization. This many to many relation is joined using the ProductCategoryRollup entity. An alternative to this is to implement a true hierarchy by removing or not using the ProductCategoryRollup entity and simply using a field in the ProductCategory entity to specify a reference to the ParentProductCategory. This can be done with the current data model by using the primaryParentCategoryId field as a reference to the only parent category.

product.feature

ProductFeature ProductFeatureType ProductFeatureCategory ProductFeatureAppl ProductFeatureApplType ProductFeatureIactn ProductFeatureIactnType FeatureDataObject

product.cost

CostComponent CostComponentType CostComponentAttribute CostComponentTypeAttr

product.price

PriceComponent PriceComponentType PriceComponentAttribute PriceComponentTypeAttr QuantityBreak OrderValueBreak SaleType

product.inventory

InventoryItem InventoryItemType InventoryItemAttribute InventoryItemTypeAttr PhysicalInventory InventoryItemVariance VarianceReason ItemVarianceAcctgTrans Lot

product.storage

Container ContainerType Facility FacilityType FacilityAttribute FacilityTypeAttr FacilityContactMechanism PartyFacility FacilityRoleType

product.supplier

ReorderGuideline SupplierProduct SupplierRatingType SupplierPrefOrder MarketInterest

Security

The Security package contains entities related to controlling access to various parts of applications that use the data model. The Login package has a UserLogin entity which contains the usernames and passwords of all users on the system and which is optionally related to a Party entity instance. The Login package also has an entity which keeps a history of logins into the system.
The SecurityGroup package contains entities used to map SecurityPermissions to UserLogins. It goes this by mapping a UserLogin to a SecurityGroup and then mapping the SecurityGroup to a SecurityPermission. In this manner complex security definitions can be shared efficiently.

security.login

UserLogin LoginAccountHistory

security.securitygroup

SecurityGroup SecurityPermission SecurityGroupPermission UserLoginSecurityGroup

InventoryItem

The **InventoryItem** entity records items of a **Product** in stock in a **Facility** at a **FacilityLocation**.

quantityOnHandTotal	Quantity in stock at this location
availableToPromiseTotal	Quantity that has not been reserved by any Order
accountingQuantityTotal	Used if you have setup the FIFO or LIFO inventory accounting. It represents the units that are currently posted in the "inventory" account (at their unitCost). This is required because the order in which the goods in inventory items physically enter or exit the warehouse is not always the same required by FIFO/LIFO accounting.