BioWare Aurora Engine
Store Format

1. Introduction

1.1. Overview

A **Store** is an object that exchanges items and gold with the player. Stores contain the list of items they have in their inventory, the buy/sell markup and whether it will purchase from the player items marked as stolen. Note that merchants will buy any kind of item except those marked as Plot.

Stores are stored in the game and toolset using BioWare's Generic File Format (GFF), and it is assumed that the reader of this document is familiar with GFF.

Stores can be blueprints or instances. Store blueprints are saved as GFF files having a UTM extension and "`UTM `" as the FileType string in their header. Store instances are stored as Store Structs within a module's GIT files.

2. Store Struct

The tables in this section describe the GFF Struct for a Store. Some Fields are only present on Instances and others only on Blueprints.

For List Fields, the tables indicate the StructID used by the List elements.

2.1 Common Store Fields

2.1.1. Store Fields in All Stores

The Table below lists the Fields that are present in all Store Structs, regardless of whether they are found in blueprints, instances, toolset data, or game data.

Table 2.1.1: Fields in all Store Structs

Label	Type	Description
`BlackMarket`	BYTE	1 if blackmarket store. Blackmarket stores will purchase items marked as stolen. 0 if normal store. Normal stores will not buy stolen items.
`BM_MarkDown`	INT	If store is BlackMarket, this is the buy markdown percentage for stolen items. The store will buy stolen items for this percentage of their normal cost.
`IdentifyPrice`	INT	-1 if the store will not identify items. 0 or greater: store will identify items for the specified price.
`LocName`	CExoLocString	Name of the Item as it appears on the toolset's Item palette, in the Name field of the toolset's Item Properties dialog, and in the game if it has been Identified.
`MarkDown`	INT	Sell markdown percentage. Items sold from the store are sold at their normal cost multiplied by this percentage.

		Usually 100 or greater.
MarkUp	INT	Buy markup percentage. The store purchases items for this percentage of their normal cost. Usually 100 or less.
MaxBuyPrice	INT	-1 if the store has no limit to how much it will pay for an item 0 or greater: maximum price that store will pay for an item.
OnOpenStore	CResRef	OnOpenStore event. To open a store for a player, use the OpenStore() scripting function.
OnStoreClosed	CResRef	OnStoreClosed event.
ResRef	CResRef	For blueprints (UTM files), this should be the same as the filename. For instances, this is the ResRef of the blueprint that the instance was created from.
StoreGold	INT	-1 if the store has infinite gold for buying items 0 or greater: amount of gold the store has. Buying items from players will deduct from this amount. Selling items to players will add to this amount. A store cannot buy items it cannot afford.
StoreList	List	List of Store Container Structs (see **Table 2.1.2** and **Table 2.1.3**)
WillNotBuy	List	List of StoreBaseItem Structs (StructID 0x17E4D; see **Table 2.1.5**). The Store will not buy any items that have a BaseItem type included in this list. If the *WillNotBuy* list contains any elements, then the *WillOnlyBuy* list is ignored and assumed to be empty.
WillOnlyBuy	List	List of StoreBaseItem Structs (StructID 0x17E4D; see **Table 2.1.5**) describing the only BaseItem types that the store will buy. If the *WillNotBuy* list is empty, then the *WillOnlyBuy* list is checked for elements. Otherwise, this list is ignored and should be empty.
Tag	CExoString	Tag of the Item. Up to 32 characters.

The *StoreList* of a Store describes what items the Store has for sale. It contains several StoreContainer Structs that contain lists of InventoryObject Structs (see **Section 3** in the **Items** document for a description of an InventoryObject). Each InventoryObject corresponds to an item that is available for sale.

Every *StoreList* contains the StoreContainers listed in **Table 2.1.2**.

Table 2.1.2: StoreContainer Structs

StructID	Description
0	Armor Items
1	Miscellaneous Items
2	Potions
3	Rings
4	Weapons

All Items have a *BaseItem* Field that serves as an index into **baseitems.2da** (see see **Table 5.1** in the **Items** document). In **baseitems.2da**, there is a *StorePanel* column that contains an integer value that specifies a StoreContainer by StructID. When an Item is added to a store in the toolset, or sold to a store ingame, an InventoryObject is created for that Item and added to the appropriate StoreContainer as determined by the *StorePanel* for that Item's *BaseItem.*

Each StoreContainer Struct contains a list of InventoryObjects that specify the ResRef of the Items available for sale. **Table 2.1.3** describes a StoreContainer.

Table 2.1.3: Fields in StoreContainer Structs (variable StructID)

Label	Type	Description
ItemList	List	list of InventoryObject Blueprints (see **Section 3** in the **Items** document), each having a StructID equal to its index in the list.

Store **Blueprints** contain InventoryObject **Blueprint** Structs (see **Section 3.2** in the **Items** document), while Store **Instances** contain InventoryObject **Instance** Structs (see **Section 3.3** in the **Items** document). An InventoryObject in a Store may also contain additional Fields beyond those normally found in an InventoryObject, as given in **Table 2.1.4**.

Table 2.1.4: Additional Fields in Store InventoryObject Structs (variable StructID)

Label	Type	Description
Infinite	BYTE	1 if the item is available in infinite supply. The store will always be able to sell this item no matter how many are purchased 0 if the item disappears from the store after purchase. If this Field is not present, the InventoryObject is treated as if the Field value were 0.

A Store can have a list of restricted BaseItem types, referring to items that the Store will not buy or items that the store will only buy. These restricted item lists contain StoreBaseItem Structs, detailed in Table 2.1.5 below.

Table 2.1.5: Fields in StoreBaseItem Structs (StructID 0x17E4D)

Label	Type	Description
BaseItem	INT	Index into **baseitems.2da** to refer to a BaseItem type

2.2. Store Blueprint Fields

The Top-Level Struct in a UTM file contains all the Fields in Table 2.1.1 above, plus those in Table 2.2 below.

Table 2.2: Fields in Store Blueprint Structs

Label	Type	Description
Comment	CExoString	Module designer comment.
ID	BYTE	ID of the node that the Item Blueprint appears under in the Store palette.
ResRef	CResRef	The filename of the UTM file itself. It is an error if this is different. Certain applications check the value of this Field instead of the ResRef of the actual file. If you manually rename a UTM file outside of the toolset, then you must also update the *ResRef* Field inside it.

2.3. Store Instance Fields

A Store Instance Struct in a GIT file contains all the Fields in Table 2.1.1, plus those in Table 2.3 below.

Table 2.3: Fields in Store Instance Structs

Label	Type	Description
TemplateResRef	CResRef	For instances, this is the ResRef of the blueprint that the instance was created from.
XOrientation YOrientation	FLOAT	x,y vector pointing in the direction of the Store's orientation
XPosition YPosition ZPosition	FLOAT	(x,y,z) coordinates of the Store within the Area that it is located in.

2.4. Store Game Instance Fields

After a GIT file has been saved by the game, the Store Instance Struct not only contains the Fields in Table 2.1.1 and Table 2.3, it also contains the Fields in Table 2.4.

INVALID_OBJECT_ID is a special constant equal to 0x7f000000 in hex.

Table 2.4: Fields in Item Instance Structs in SaveGames

Label	Type	Description
ObjectId	DWORD	Object ID used by game for this object.
VarTable	List	List of scripting variables stored on this object. StructID 0. See **Section 3** of the **Common GFF Structs** document.