BioWare Aurora Engine
Item Format

BioWare Corp.
http://www.bioware.com

1. Introduction

1.1. Overview

An **Item** is an object that can be placed in an area, or carried in a container, whether the container be a creature, placeable object, or store.

Items 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.

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

1.2. Terminology

icon: the 2D image of an item as it appears in a player's inventory. The term "icon" may refer to the entire image, or to the component parts of the image, if the inventory icon is constructed from several other images.

model: the 3D model of an item as it appears in the game environment. This may refer to the model as seen when equipped by the player or when lying on the ground. An item's equipped model and ground model may or may not be the same, and not all items have equipped models.

2. Item Struct

The tables in this section describe the GFF Struct for an Item. 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 Item Fields

2.1.1. Item Fields in All Items

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

Table 2.1.1: Fields in all Item Structs

Label	Type	Description
`AddCost`	DWORD	Additional cost
`BaseItem`	INT	Index into **baseitems.2da**.
`Charges`	BYTE	Number of charges left on this item. Note that there is no property for the original number of charges.
`Cost`	DWORD	Cost of the Item
`Cursed`	BYTE	1 if the item cannot be removed from its container (ie., undroppable, unsellable, unpickpocketable). 0 if the item can be removed from its container.
`DescIdentified`	CExoLocString	Identified description
`Description`	CExoLocString	Unidentified description
`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.
`Plot`	BYTE	1 if this is a Plot Item. Plot Items cannot be sold. 0 if this is a normal Item.
`PropertiesList`	List	List of ItemProperty Structs. StructID 0. **See Section 2.1.3**.
`StackSize`	WORD	1 if item has an unstackable base item type. Otherwise, this is the stack size of the item, always greater than or equal to 1. (eg., 50 arrows)
`Stolen`	BYTE	1 if stolen 0 if not stolen
`Tag`	CExoString	Tag of the Item. Up to 32 characters.
`TemplateResRef`	CResRef	For blueprints (UTI files), this should be the same as the filename. For instances, this is the ResRef of the blueprint that the instance was created from.

2.1.2. Additional Item Fields by ModelType

The *BaseItem* of an item specifies a row in **baseitems.2da** that provides information on the item type. One of the columns in baseitems.2da is the **ModelType**, which controls what the item looks like. Refer to the ModelType entry in **Table 4.1.1**.

Depending on the ModelType, an Item Struct will contain the Fields listed in one or more of the tables below, in addition to those in Table 2.1.1.

Table 2.1.2.1: Fields in Layered (ModelType 1) and Armor (ModelType 3) Item Structs

Label	Type	Description
`Cloth1Color`	BYTE	Index into a row of pixels in pal_cloth01.tga. Specifies the colors to use for cloth1 PLT layer.
`Cloth2Color`	BYTE	Index into a row of pixels in pal_cloth01.tga. Specifies the colors to use for cloth2 PLT layer.
`Leather1Color`	BYTE	Index into a row of pixels in pal_leath01.tga. Specifies the colors to use for leather1 PLT layer.
`Leather2Color`	BYTE	Index into a row of pixels in pal_leath01.tga. Specifies the colors to use for leather2 PLT layer.
`Metal1Color`	BYTE	index into a row of pixels in pal_armor01.tga. Specifies the colors to use for metal1 PLT layer.
`Metal2Color`	BYTE	index into a row of pixels in pal_armor01.tga. Specifies the colors to use for metal2 PLT layer.

Table 2.1.2.2: Fields in Simple (ModelType 0) and Layered (ModelType 1) Item Structs

Label	Type	Description
`ModelPart1`	BYTE	part number

Table 2.1.2.3: Fields in Composite Item (ModelType 2) Structs

Label	Type	Description
`ModelPart1`	BYTE	part number
`ModelPart2`	BYTE	part number
`ModelPart3`	BYTE	part number

Table 2.1.2.4: Fields in Armor (ModelType 3) Item Structs

Label	Type	Description
`ArmorPart_Belt`	BYTE	Index into parts_belt.2da

`ArmorPart_LBicep`	BYTE	Index into parts_bicep.2da
`ArmorPart_LFArm`	BYTE	Index into parts_forearm.2da
`ArmorPart_LFoot`	BYTE	Index into parts_foot.2da
`ArmorPart_LHand`	BYTE	Index into parts_hand.2da
`ArmorPart_LShin`	BYTE	Index into parts_shin.2da
`ArmorPart_LShoul`	BYTE	Index into parts_shoulder.2da
`ArmorPart_LThigh`	BYTE	Index into parts_legs.2da
`ArmorPart_Neck`	BYTE	Index into parts_neck.2da
`ArmorPart_Pelvis`	BYTE	Index into parts_pelvis.2da
`ArmorPart_RBicep`	BYTE	Index into parts_bicep.2da
`ArmorPart_RFArm`	BYTE	Index into parts_forearm.2da
`ArmorPart_RFoot`	BYTE	Index into parts_foot.2da
`ArmorPart_RHand`	BYTE	Index into parts_hand.2da
`ArmorPart_Robe`	BYTE	Index into parts_robe.2da
`ArmorPart_RShin`	BYTE	Index into parts_shin.2da
`ArmorPart_RShoul`	BYTE	Index into parts_shoulder.2da
`ArmorPart_RThigh`	BYTE	Index into parts_legs.2da
`ArmorPart_Torso`	BYTE	Index into parts_torso.2da

2.1.3. ItemProperty Fields

Each ItemProperty element in the *PropertiesList* of an Item contains the Fields given below.

Table 2.1.3: Fields in ItemProperty Structs (StructID 0)

Label	Type	Description
`ChanceAppear`	BYTE	Obsolete. Always 100.
`CostTable`	BYTE	Index into **iprp_costtable.2da**. Equal to the value in the *CostTableResRef* column of **itempropdef.2da** at the row specified by the *PropertyName* Field. Must always be specified.
`CostValue`	WORD	Index into a cost table. The ResRef of the cost table is the value under the *Name* column of **iprp_costtable.2da**, at the row specified by the *CostTable* Field. Must always be specified.
`Param1`	BYTE	Index into **iprp_paramtable.2da**. Specifies a params table. This value is -1 if there are no parameters. There are 2 ways to determine if there are no parameters depending on whether the ItemProperty struct has a non-zero Subtype Field or not. **If there is a Subtype:** there are no parameters if the *Param1ResRef* column does not exist in the subtype table (see the Subtype Field description in this struct) or if there is a *Param1ResRef* column and the value is **** in the row indexed by the Subtype Field.

		If there is no Subtype: there are no parameters if the itempropdef.2da has **** as the value under its *Param1ResRef* column at the row indexed by the PropertyName Field.
`Param1Value`	BYTE	Index into a params 2da. The ResRef of the params 2da is specified by the *TableResRef* column of **iprp_paramtable.2da** at the row specified by the *Param1* Field. If an ItemProperty does not have a params table, this Field defaults to 0.
`Param2`	BYTE	Obsolete. Would be the same as Param1 but using the Param1ResRef column instead.
`Param2Value`	BYTE	Obsolete. Would be the same as Param1.
`PropertyName`	WORD	Index into **itempropdefs.2da**. Must always be specified.
`Subtype`	WORD	Index into a item property subtype 2da. The ResRef of the subtype 2da is specified by the *SubTypeResRef* column value in **itempropdefs.2da**, at the row specified by the *PropertyName* Field. Set to 0 if there is no SubTypeResRef.

2.2. Item Blueprint Fields

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

Table 2.2: Fields in Item Blueprint Structs

Label	Type	Description
`Comment`	CExoString	Module designer comment.
`PaletteID`	BYTE	ID of the node that the Item Blueprint appears under in the Item palette.
`TemplateResRef`	CResRef	The filename of the UTI 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 UTI file outside of the toolset, then you must also update the *TemplateResRef* Field inside it.

2.3. Item Instance Fields

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

An Item Instance Struct in an InventoryObject instance contains does not contain any additional Fields.

Table 2.3: Fields in Item 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 item's orientation

`XPosition`
`YPosition`
`ZPosition`
	FLOAT	(x,y,z) coordinates of the Item within the Area that it is located in.

2.4. *Item Game Instance Fields*

After a GIT file has been saved by the game, the Item 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.

3. InventoryObject Struct

An InventoryObject is a Struct that may be present in the inventory list of a Container object. Creatures, Items, and Placeable Objects can all be Containers.

A Container Item contains a Field called *ItemList*, of GFF type List, containing InventoryObject Structs. Items in the toolset never have an ItemList, but they can in a savegame (inside the GIT and/or IFO files) or character file (BIC). The game saves the StructIDs as being equal to the index of the Struct element within the ItemList.

3.1 *Common InventoryObject Fields*

The Table below lists the Fields that are present in all InventoryObject Structs, regardless of where they are found.

Table 3.1: Fields in all InventoryObject Structs

Label	Type	Description
`Repos_PosX`	WORD	x-position of item in inventory grid
`Repos_PosY`	WORD	y-position of item in inventory grid

3.2. *InventoryObject Blueprint Fields*

An InventoryObject Blueprint Struct contains all the Fields in Table 3.1, plus those in Table 3.2 below.

Table 3.2: Fields in InventoryObject Blueprint Structs

Label	Type	Description
`InventoryRes`	CResRef	ResRef of UTI Item Blueprint file

3.3. *InventoryObject Instance Fields*

An Item Instance Struct in a GIT file contains all the Fields in Table 3.1, plus those in Item Instances, as given in Tables 2.1.1 and 2.3.

3.4. InventoryObject Game Instance Fields

An Item Instance Struct in a GIT file contains all the Fields in Table 3.1, plus those in Item Game Instances, as given in Tables 2.1.1, 2.3, and 2.4.

The XOrientation is always 1.0, and the YOrientation and 3 Position Fields are always 0.0.

Character files use the same format as savegames for InventoryObjects, except that they do not include the Fields in Table 2.4.

4. Calculations and Procedures

4.1. Icon and Model Part Availability

The icons and model resources for items have ResRefs that end in a 3-digit, zero-padded numerical string.

Examples: helm_001.mdl, pmh0_bicepl006.mdl, waxbt_b_011.mdl, iwaxbt_t_011.tga.

This section describes how to determine the ResRefs (aka filenames) of the model and icon resources for an item, depending on its *ModelType* as specified in **baseitems.2da**. In the descriptions below, the following tokens are used:

- <ItemClass> = the *ItemClass* from baseitems.2da.
- <number> = part number

If an item model cannot be found, then the MDL file specified by the *DefaultModel* column in baseitems.2da will be used instead.

If an item icon cannot be found, then the icon specified by the *DefaultIcon* column in baseitems.2da will be used instead.

The icon for an item must have dimensions equal to 32 pixels multiplied by the InvSlotWidth and InvSlotHeight values from **baseitems.2da**.

4.1.1. Simple

Model = <ItemClass>_<number>.mdl

(eg., it_torch_001.mdl)

Icon = i<ItemClass>_<number>.tga

(eg., iit_torch_001.tga)

Simple item types typically use one model for all variants, with only the icon being different.

The available icons in the toolset consist of all existing TGA icons from the *MinRange* to the *MaxRange* as specified in baseitems.2da, using the Icon ResRef naming convention given above.

4.1.2. Layered

Model = <ItemClass>_<number>.mdl

(eg., helm_001.mdl)

Icon = i<ItemClass>_<number>.plt

(eg., ihelm_001.plt)

The available parts are determined by scanning for all existing PLT icons from the *MinRange* to the *MaxRange* as specified in baseitems.2da, using the Icon ResRef naming convention given above. If an icon exists, assume that the MDL also exists and make that part number selectable in the toolset. (Exception: for helmets: check for the MDL instead of the PLT)

4.1.3. Composite

Model = <ItemClass>_<position>_<number>.mdl

(eg., waxbt_b_011.mdl, waxbt_m_011.mdl, waxbt_t_011.mdl)

Icon = i<Model ResRef>.tga

(eg., iwaxbt_b_011.tga, iwaxbt_m_011.tga, iwaxbt_t_011.tga)

The <position> is one of the following letters: b (bottom), m (middle), or t (top), usually corresponding to the pommel, hilt, or blade of a weapon.

For weapons, the 3-digit <number> field is broken up into two parts. The first 2 digits map to a physical shape variant, and the last digit is a color variant. For example, parts 011, 021, and 031 all have the same color, but different shapes. Parts 011, 012, and 013 all have the same shape, but different colors.

To determine what colors are available for each of the b, m, and t portions of a composite item, do the following steps. Scan for icons, incrementing the <number> from the baseitems.2da *MinRange* to *MaxRange*, and find first icon file that exists. Check the color code in the icon's resref (the last digit). This is the minimum available color. Increment the <number> from there until another icon is found that has the same color. Recall what color the icon was before that. That color is the maximum available color.

To determine what shapes are available, scan for TGA icons from <number> = (*MinRange* + minimum color) to <number> = *MaxRange*, incrementing by 10 each time. If an icon exists, then assume that all the MDL file exists as well, and also assume that the shape variant exists in all the other colors as well. For example, suppose that the minimum color found earlier was 2, the maximum color was 4, *MinRange* = 10, and *MaxRange* = 100. The icon scan would check for 012, 022, 032, ... 092. If 082 exists, for example, then assume that 083 and 084 also exist

When drawing the Icon for a composite item, the 3 portions are painted one after the other in the order: bottom, middle, top. The order is important because the icons overlap.

4.1.4. Armor

Model = p<gender><race><phenotype>_<bodypart><number>.mdl

(eg., pmh0_chest001.mdl)

Icon = ip<gender>_<bodypart><number>.plt

(eg., ipm_chest001.plt)

where

<gender> = m or f

<bodypart> = belt, bicepl, bicepr, chest, footl, footr, forel, forer, handl, handr, legl, legr, neck, pelvis, shinl, shinr, shol, shor**--**as listed in **capart.2da**

The complete icon for an Armor item consists of several individual bodypart icons drawn onto a surface in the following order: pelvis, chest, belt, right shoulder, left shoulder, robe. The order is important because the icons may overlap. There are no icons for the other body parts.

For each <bodypart>, the available <number> values are determined by checking the appropriate **parts_<armorpart>.2da** file, as given in **Table 4.1.4** below.

Table 4.1.4: Armor body part and parts 2da matchups

Body Part	2DA
`belt`	parts_belt
`bicep1` `bicepr`	parts_bicep
`chest`	parts_chest
`footl` `footr`	parts_foot
`forel` `forer`	parts_forearm
`handl` `handr`	parts_hand
`legl` `legr`	parts_legs
`neck`	parts_neck
`pelvis`	parts_pelvis
`robe`	parts_robe
`shinl` `shinr`	parts_shin
`shol` `shor`	parts_shoulder

If a row in the **parts_<armorpart>.2da** file has an *ACBONUS* column value that is *not* ****, then the <number> equal to that row index is available as a selectable bodypart. Note that in some cases, part 000 is available even though there are no 000 MDL or PLT files for armor. If part 000 is selected, then that armor part is empty. (For example, no shoulder or belt, or no robe.)

In the toolset, the available parts are sorted in order of increasing *ACBONUS* as listed in the parts 2da. If several parts have identical *ACBONUS*, then they are sorted in order of increasing row number. Note that for all armor parts except the chest, the *ACBONUS* 2da value is used only for sorting part numbers, and do not actually affect the AC of the armor.

Robe parts are different from other armor parts in that they may hide other parts. A robe-hidden part does not appear in the armor icon, and its 3D model is not rendered. The **parts_robe.2da** file contains additional columns beyond those normally present in a parts 2da. There exists one column for every other part, each one titled *HIDE<PART>*, where <PART> is one of the values in the Body Part column of **Table 4.1.4**. For example, HIDEFOREL, HIDELEGR and HIDECHEST. Each HIDE<PART> column contains a 1 if the specified part is hidden by the robe, or 0 if not.

4.2. Property Availability

To determine which Item Properties are available for a given base item type, get the number in the PropColumn column from **baseitems.2da** (**Table 5.1**), then find the column in **itemprops.2da** (**Table 5.3**) that starts with that number in its name. If a row in itemprops.2da has 1 as the value under that column, then the Item Property for the corresponding row in **itempropdefs.2da** (**Table 5.2**) is available for the baseitem. If the value in itemprops.2da is ****, then the corresponding Item Property in itempropdefs.2da is not available.

For example, baseitem 12 (amulet) has 16 as its *PropColumn* value. Thus, the available Item Properties are determined by checking under the column "16_Misc" in itemprops.2da. This means that AC Bonus vs. Alignment is available for amulets but Enhancement Bonus is not.

4.3. Property Description Construction

The format used by the toolset for an item property description is:

PropertyName : Subtype [CostValue] [Param1: Param1Value]

eg., Damage Bonus vs. Racial Type: Dragon [1 Damage] [Type: Acid]

eg., On Hit: Daze [DC = 14] [Duration: 50% / 2 rounds]

eg., Light [Dim (5m)] [Color: Blue]

The game uses similar formatting, but without the square brackets.

This section describes how to get the text for the various components of an Item Property description.

4.3.1. PropertyName

Look up the StrRef stored in column 0 (should be Name) of **itempropdef.2da** (**Table 5.2**), at the row indexed by the ItemProperty Struct's *PropertyName* Field (see **Table 2.1.3**). This StrRef points to the name of the Item Property (eg., "Damage Bonus vs. Racial Type", "On Hit", "Light")

4.3.2. Subtype

In **itempropdef.2da**, look up the *SubTypeResRef* column value at the row indexed by the ItemProperty Struct's *PropertyName* Field. If it is ****, then there are no subtypes for this Item Property, so there is no subtype portion of the Item Property description. Otherwise, the string under this column is the ResRef of the ***subtype table*** 2da.

If there is a subtype table, load it and use the ItemProperty Struct's *Subtype* Field as an index into the 2da. Get the StrRef from the *name* column. This StrRef points to the name of the Subtype (eg., "Dragon", "Daze").

4.3.3. CostTable Value

In **itempropdef.2da**, look up the *CostTableResRef* number at the row indexed by the ItemProperty Struct's *PropertyName* Field. This should be the same as the ItemProperty Struct's *CostTable* Field.

Use the CostTableResRef value as an index into **iprp_costtable.2da** and get the string under the *Name* column. This is the ResRef of the cost table 2da to use.

Load the cost table 2da. Using the ItemProperty Struct's *CostValue* Field as an index into the cost table, get the *Name* column StrRef. This StrRef points to the name of the Cost value (eg., "1 Damage", "DC = 14", "Dim (5m)")

4.3.4. Param

Get the ResRef of the Param Table.

If the ItemProperty has a subtype table (see **Section 4.3.2**), then look for a *Param1ResRef* column in the subtype table. This column contains the *param table index*, and should be identical the *Param1* Field of the ItemProperty Struct.

If the ItemProperty does not have a subtype table, or the subtype table does not have a *Param1ResRef* column, then look under the Param1ResRef column in **itempropdef.2da**, and use that as the param table index. In this case as well, the index should equal the *Param1* Field of the ItemProperty Struct.

Use the param table index as an index into **iprp_paramtable.2da**. Look under the *Name* column for a StrRef, and look under the *TableResRef* column for a string. The *Name* StrRef points to the text for the name of the parameter (eg., "Type", "Duration", "Color"). The *TableResRef* string is the ResRef of the ***param table*** 2da.

Use the ItemProperty Struct's *Param1Value* as an index into the param table found above, and get the StrRef under the "Name" column. This StrRef points to the name of the param value (eg., "Acid", "50% / 2 Rounds", "Red")

4.4. Cost Calculation

The cost of an item is determined by the following formula:

ItemCost = [BaseCost + 1000*(Multiplier^2 **-** NegMultiplier^2) + SpellCosts]*MaxStack*BaseMult + AdditionalCost

where:

BaseMult = *ItemMultiplier* column value from **baseitems.2da**.

AdditionalCost = *AddCost* Field from the Item Struct.

and the other terms are as defined in the following subsections:

4.4.1. Base Cost

BaseCost = *BaseCost* column value from **baseitems.2da** for all items except armor.

For armor, BaseCost is the cost in **armor.2da** for the armor's base AC. The armor's base AC is the *ACBONUS* from **parts_chest.2da**, using the part number of the chest as the index into parts_chest.2da.

4.4.2. Multipliers

Multiplier is the sum of the costs of all the Item Properties whose costs are positive. That is:

NegMultiplier is the sum of the costs of all Item Properties whose costs are negative.

If an Item Property has a *PropertyName* of 15 (Cast Spell), then omit it from the Multiplier/NegMultiplier totals. It will be handled when calculating the SpellCosts instead.

To calculate the cost of a single Item Property, use the following formula:

ItemPropertyCost = PropertyCost + SubtypeCost + CostValue

Add the ItemProperty's cost to the Multiplier total if it is positive. Add it to the NegMultiplier total if it is negative.

Note that Item Property Params do not affect Item Property cost.

The PropertyCost, SubtypeCost, and CostValue terms are obtained as described below.

PropertyCost

In **itempropdef.2da**, get the floating point value in the *Cost* column, at the row indexed by the *PropertyName* Field of the ItemProperty Struct. If the *Cost* column value is ****, treat it as 0. This floating point value is the PropertyCost.

SubtypeCost

If the PropertyCost obtained above from **itempropdef.2da** was 0, then get the ResRef in the *SubTypeResRef* column of itempropdef.2da, at the row indexed by the 2*PropertyName* Field of the ItemProperty Struct. This is the resref of the subtype table 2da.

In the subtype 2da, get the floating point value in the *Cost* column at the row indexed by the *Subtype* Field of the ItemProperty Struct. This floating point value is the SubtypeCost.

Only get the SubtypeCost if the PropertyCost was 0. If the PropertyCost was greater than 0, then the SubtypeCost is automatically 0 instead.

CostValue

In **iprp_costtable.2da**, get the string in the *Name* column at the row indexed by the *CostTable* Field in the ItemProperty Struct. This is the ResRef of the cost table 2da.

In the cost table, get the floating point value in the *Cost* column in the row indexed by the *CostValue* Field in the ItemProperty Struct. This floating point value is the CostValue.

4.4.3. Cast Spell Costs

To calculate the cost of a single Cast Spell Item Property, use the following formula:

CastSpellCost = (PropertyCost + CostValue)* SubtypeCost

The PropertyCost, SubtypeCost, and CostValue terms are obtained in the same way as for non-CastSpell Item Properties.

After calculating all the CastSpellCost values for all the Cast Spell Item Properties, modify them as follows:

- Most expensive: multiply by 100%
- Second most expensive: multiply by 75%

- All others: multiply by 50%

After adjusting the CastSpellCosts, add them up to obtain the total SpellCosts value. Use the total SpellCosts to calculate the total ItemCost using the formula given at the very beginning of Section 4.4.

4.5. Required Lore and Level

The character level required to use an item is equal to 1 plus the row index in **itemvalue.2da** that contains the smallest number in the MAXSINGLEITEMVALUE column that is still greater than or equal to the cost of the item.

The Lore skill check required to identify an item is equal to the row index in **skillvsitemcost.2da** that contains the smallest number in the DeviceCostMax column that is still greater than or equal to the cost of them item.

5. Item-related 2DA Files

5.1. Base Items

The baseitems 2da defines all the item types that exist. Many characteristics of an item are determined by its base item type and cannot be set by the addition of ItemProperties to its PropertiesList. These characteristics are defined in baseitems.2da.

Table 5.1: baseitems.2da columns

<table>
<tr><th>Column</th><th>Type</th><th>Description</th></tr>
<tr><td>Name</td><td>Integer</td><td>StrRef for the name of the base item type</td></tr>
<tr><td>Label</td><td>String</td><td>Programmer label</td></tr>
<tr><td>InvSlotWidth</td><td>Integer</td><td>Height of item's inventory icon, measured in number of inventory grid squares.</td></tr>
<tr><td>InvSlotHeight</td><td>Integer</td><td>Width of item's inventory icon, measured in number of inventory grid squares.</td></tr>
<tr><td>EquipableSlots</td><td>Integer</td><td>Set of bit flags specifying where the item can be equipped.
<table>
<tr><td>HEAD</td><td>0x1</td></tr>
<tr><td>CHEST</td><td>0x2</td></tr>
<tr><td>BOOTS</td><td>0x4</td></tr>
<tr><td>ARMS</td><td>0x8</td></tr>
<tr><td>RIGHTHAND</td><td>0x10</td></tr>
<tr><td>LEFTHAND</td><td>0x20</td></tr>
<tr><td>CLOAK</td><td>0x40</td></tr>
<tr><td>LEFTRING</td><td>0x80</td></tr>
<tr><td>RIGHTRING</td><td>0x100</td></tr>
<tr><td>NECK</td><td>0x200</td></tr>
<tr><td>BELT</td><td>0x400</td></tr>
<tr><td>ARROWS</td><td>0x800</td></tr>
<tr><td>BULLETS</td><td>0x1000</td></tr>
<tr><td>BOLTS</td><td>0x2000</td></tr>
</table></td></tr>
<tr><td>CanRotateIcon</td><td>Integer</td><td>1 if inventory icon for this item may be rotated 90 degrees clockwise, such as when placed on a player's quickbar.
0 if the icon may not be rotated.</td></tr>
<tr><td>ModelType</td><td>Integer</td><td>Defines how the item's model and icon are constructed. See <b>Section 4.1</b>.
<table>
<tr><td>Value</td><td>Description</td><td>Has Color Layers</td><td># of parts</td></tr>
<tr><td>0</td><td>simple</td><td>no</td><td>1</td></tr>
<tr><td>1</td><td>layered</td><td>yes</td><td>1</td></tr>
</table></td></tr>
</table>

		2	composite	no	3
		3	armor	yes	18
ItemClass	String	Base ResRef for item icon and model parts. See **Section 4.1**.			
GenderSpecific	Integer	0 if icons and models are identical for all genders of players 1 if the icons and models differ			
Part1EnvMap Part2EnvMap Part3EnvMap	Integer	Determines if part 1, 2, or 3 of the item's model should have environment mapping applied to it. 1 to use environment mapping. 0 to not use environment mapping.			
DefaultModel	String	ResRef of the default model to use for the item.			
Container	Integer	0 if this item is not a container 1 if this item is a container and can contain other items			
WeaponWield	Integer	Weapon Wield style **** - item cannot be wielded, or it is a melee weapon that is held in one or two hands depending on the size of the creature wielding it. 1 - nonweapon 4 - pole 5 - bow 6 - crossbow 7 - shield 8 - two-bladed 9 - creature weapon 10 - sling 11 - thrown			
WeaponType	Integer	Weapon damage type. 0 - none 1 - piercing 2 - bludgeoning 3 - slashing 4 - piercing and slashing			
WeaponSize	Integer	Weapon size. 1 - tiny 2 - small 3 - medium 4 - large 5 - huge			
RangedWeapon	Integer	**** if not a ranged weapon, otherwise, an integer.			
PrefAttackDist	Float	**** if not a weapon, otherwise, the preferred attacking distance when using this weapon. The distance is selected so that attack animations look their best in the most commonly anticipated situations.			
MinRange	Integer	Minimum part number to scan for in toolset. Lowest possible value is 0.			
MaxRange	Integer	Maximum part number to scan for in toolset. Highest possible value is 999.			
NumDice	Integer	Number of dice to roll to determine weapon damage. **** for non-weapons.			
DieToRoll	Integer	Size of dice to roll to determine weapon damage. **** for non-weapons.			
CritThread	Integer	Critical threat range. **** for non-weapons.			

CritHitMult	Integer	Critical hit multiplier. **** for non-weapons.
Category	Integer	0 - none 1 - melee 2 - ranged 3 - shield 4 - armor 5 - helmet 6 - ammo 7 - thrown 8 - staves 9 - potion 10 - scroll 11 - thieves' tools 12 - misc 13 - wands 14 - rods 15 - traps 16 - misc unequippable 17 - container 19 - healers
BaseCost	Integer	base cost to use in item cost calculation
Stacking	Integer	Maximum stack size of item
ItemMultiplier	Integer	Used in Cost calculation. See **Section 4.4**.
Description	Integer	StrRef of basic description when examining an item of this type, if the item is unidentified, or if there is no description for the item in its Description CExoLocString Field.
InvSoundType	Integer	Specifies sound to make when moving the item in inventory during the game. 0 - Armor 1 - Shield 2 - Wood Melee Weapon 3 - Metal Melee Weapon 4 - Ranged Weapon 5 - Ammo 6 - Potion 7 - Paper 8 - Treasure 9 - Generic The resref is contructed in the following way: Non-Armor XX_YYYYY XX = PU or DR for pickup/equipping and drop/unequpping. YYYYY is hardcoded to correspond to one of the above inventory sound types.
MaxProps	Integer	Maximum number of Cast Spell item properties allowed.
MinProps	Integer	Minimum number of Cast Spell item properties that must exist on item.
PropColumn		Column of **itemprops.2da**.that defines what item properties are available for this baseitem. There is a one-to-one correspondence between rows in

		itemprops.2da and **itempropdefs.2da**. If a baseitem can have a certain property, its row in itemprops.2da is 1. If not, then the value is ****.
StorePanel		Store Panel that items of this type appear in. 0 - armor 1 - weapons 2 - potions 3 - scrolls 4 - miscellaneous
ReqFeat0 ReqFeat1 ReqFeat2 ReqFeat3 ReqFeat4	Integer	List of feats required to use the item. Can specify up to 5 required feats. **** indicates no requirement
AC_Enchant	Integer	The type of AC bonus the item applies: 0 - dodge 1 - natural 2 - armor 3 - shield 4 - deflection
BaseAC	Integer	Base AC added when item is equipped. Note that sheields use this column, but armor does not.
ArmorCheckPen	Integer	Armor check penalty
BaseItemStatRef	Integer	StrRef describing the statistics of this item
ChargesStarting	Integer	Initial number of charges on an item of this type
RotateOnGround	Integer	0 - no rotation 1 - rotate 90 degrees around positive y-axis 2 - rotate 90 degrees around positive x-axis
TenthLBS	Integer	Weight in tenths of pounds
WeaponMatType	Integer	Weapon material type. Index into **weaponsounds.2da**. Determines the sound emitted when weapon strikes something.
AmmunitionType	Integer	**** if no ammunition 0 - none 1 - arrow 2 - bolt 3 - bullet 4 - dart 5 - shuriken 6 - throwing ax
QBBehaviour	Integer	Determines the behaviour when this property appears on the player's quick bar. 0 - default 1 - select spell, targets normally 2 - select spell, always targets self
ArcaneSpellFailure	Integer	Arcane spell failure chance when equipped. **** if does not affect arcane spell casting.
%AnimSlashL %AnimSlashR %AnimSlashS	Integer	% chance to use the left-slash, right-slash, or stab animation when using this weapon. **** if the item is not a melee weapon. Left-Slash and Stab should add up to 100, and Right-Slash and Stab should add up to 100. The Left and Right slash percentages are the chances of

		doing that move if the wielder is in the already proper stance. For example, a creature in the right-ready combat animation can only left-slash or stab, and after a left-slash, it enters the left-ready stance.
StorePanelSort	Integer	0 to 99. Lower-numbered items appear first in the store panel display in the game. Higher-numbered items appear last.
ILRStackSize	Integer	Sometimes used instead of StackSize when calculating item cost

5.2. Item Property Definitions

The table itempropdef.2da defines the available item properties that can be added to an Item's *PropertiesList*.

Table 5.2: itempropdef.2da columns

Column	Type	Description
Name	Integer	StrRef of name of the item property. eg., "Enhancement Bonus"
Label	String	Programmer label
SubTypeResRef	String	ResRef of SubType 2da
Cost	Float	Used in Cost calculation. See **Section 4.4**.
CostTableResRef	Integer	Index into **iprp_costtable.2da**
Param1ResRef	Integer	**** for properties that have no parameters, or whose parameters are defined in the subtype table. Otherwise, index into **iprp_paramtable.2da**.
GameStrRef	Integer	StrRef of name of the item property, formatted to form a partial string. eg., "Enhancement Bonus:"
Description	Integer	StrRef of description of the item property

5.3. Item Property Availability

The table itemprops.2da defines the available properties that are available for different baseitems, according the PropColumn for the baseitem.

Table 5.3: itemprops.2da columns

Column	Type	Description
numbered columns, column name is <number>_<string>	Integer	The <number> in the column name is a number that can appear under the *PropColumn* column in **baseitems.2da**. The <string> in the column name is unimportant and present only for the convenience of the programmer editing the 2da. Each row in itemprops.2da has a one-to-one correspondence with a row in itempropdef.2da. The value under these columns is 1 if the property on the row is available for the specified property column The value is **** if the property is not available See **Section 4.2**.

StringRef	Integer	StrRef of the property name. Should be the same as the matching StrRef under the "Name" column of itempropdef.2da.
Label	String	Programmer label. Depending on the row within itemprops.2da, there may be multiple additional columns after the Label column, with no heading. This is a violation of the 2da formatting convention where all rows have the same number of columns and all columns have a header. However, this violation of the rules does not matter, because in this case, the Label column is known to always be the last column, with no meaningful columns after it. This is the only 2da where, if you wish to add columns, you should insert them, not append them to the end. The insertion should be immediately before the Label column.

5.4. Item Property Subtype Tables

All subtype tables contain the columns specified in Table 5.4.1.

Table 5.4: subtype table 2da columns

Column	Type	Description
Name	String	StrRef of the name of the SubType.
Label	String	Programmer label
Cost		(Required if itempropdef.2da has **** for the Cost)
Param1ResRef	Integer	Index into iprp_paramtable.2da. Specifies the param table for this item property's subtype. The ResRef of the param table is listed in the TableResRef column of iprp_paramtable.2da. If the value in this column is ****, check the Param1ResRef column in itempropdefs.2da, using the same row index into itempropdefs.2da that brought you to this subtype table in the first place.

Some subtype tables contain additional columns beyond those specified in Table 5.4.1. Those tables are detailed below.

Table 5.4.1: iprp_ammotype.2da additional columns

Additional Column	Type	Description
AmmoType	Integer	Index into baseitems.2da

Table 5.4.2: iprp_feats.2da additional columns

Additional Column	Type	Description
FeatIndex	Integer	Index into feat.2da

Table 5.4.3: iprp_spells.2da additional columns

Additional Column	Type	Description
CasterLvl	Integer	Cast the spell as if the caster had the specified level.
InnateLvl	Float	Spell Level. Cantrips are 0.5. All other spells have InnateLvl equal to their Spell Level.

SpellIndex	Integer	Index into spells.2da
PotionUse	Integer	Can be applied to potions
WandUse	Integer	Can be applied to wands
GeneralUse	Integer	Can be applied to items that are not potions or wands
Icon	String	ResRef of TGA icon to use ingame

Table 5.4.4: spellshl.2da columns

Additional Column	Type	Description
Letter	String	Unused.

5.5. Item Cost Tables

The table iprp_paramtable.2da defines the available cost tables.

Table 5.5: iprp_costtable.2da columns

Column	Type	Description
Name	String	ResRef of CostTable 2da
Label	String	Programmer label
ClientLoad	Integer	0 if loaded by client 1 if loaded by game client

Cost Tables

All cost tables contain the columns specified in Table 5.5.1.

Table 5.5.1: cost table 2da columns

Column	Type	Description
Cost	Float	Used in Item Cost calculation. See **Section 4.4**.
Name	Integer	StrRef of the name of the cost table entry. If the Name is ****, then the cost table value for this row is not available for assignment to an ItemProperty. That is, an ItemProperty may not have its CostValue Field set to the index of a row that contains **** for the Name. eg., "+1"
Label	String	Programmer label

Some cost tables contain additional columns beyond those specified in Table 5.5.1. Those tables are detailed below.

Table 5.5.2: iprp_ammocost.2da additional columns

Additional Column	Type	Description
Arrow	String	ResRef of Item Blueprint (UTI file) to use to create instances of the ammunition. **** if there is no arrow blueprint for this ammo type.
Bolt	String	ResRef of Item Blueprint (UTI file) to use to create instances of the ammunition. **** if there is no bolt blueprint for this ammo type.
Bullet	String	ResRef of Item Blueprint (UTI file) to use to create instances of the ammunition. **** if there is no bullet blueprint for this ammo type.

Table 5.5.3: iprp_bonuscost.2da additional columns

Additional Column	Type	Description
Value	Integer	Amount of bonus to ability score, etc. as appropriate.

Table 5.5.4: iprp_chargecost.2da additional columns

Additional Column	Type	Description
PotionCost	Integer	Use this column instead of the Cost column if the baseitem is a potion.
WandCost	Integer	Use this column instead of the Cost column if the baseitem is a wand.

Table 5.5.5: damagecost.2da additional columns

Additional Column	Type	Description
NumDice	Integer	Number of dice to throw
Die	Integer	Sise of each die to throw
Rank	Integer	Strength of this damage bonus relative to the others in the 2da. Starts from 1 for the weakest, and counts up.
GameString	Integer	StrRef of string to display in the game for the damage amount. eg. "1d4"

Table 5.5.6: damvulcost.2da additional columns

Additional Column	Type	Description
Value	Integer	Percent amount of damage vulnerability

Table 5.5.7: immuncost.2da additional columns

Additional Column	Type	Description
Value	Integer	Percent amount of immunity

Table 5.5.8: meleecost.2da additional columns

Additional Column	Type	Description
Value	Integer	Amount of bonus to damage, AC, attack bonus, enhancement bonus, saving throw, etc. as appropriate

Table 5.5.9: monstcost.2da additional columns

Additional Column	Type	Description
NumDice	Integer	Number of dice to throw
Die	Integer	Size of each die to throw

Table 5.5.10: neg5cost and neg10cost.2da additional columns

Additional Column	Type	Description
Value	Integer	Amount of penalty to ability score, AC, skill, etc. as appropriate.

Table 5.5.11: onhitcost.2da additional columns

Additional Column	Type	Description
Value	Integer	DC of the onhit effect

Table 5.5.12: redcost.2da additional columns

Additional Column	Type	Description
Value	Float	Amount by which to reduce weight of the item's contents. Mulitply weight by this value and subtract the result from the normal weight.

Table 5.5.13: resistcost.2da additional columns

Additional Column	Type	Description
Amount	Integer	Number of hit points of damage resistance

Table 5.5.14: soakcost.2da additional additional columns

Additional Column	Type	Description
Amount	Integer	In damage reduction, the number of hit points by which to

		reduce the damage taken from an attack that does not exceed the required attack bonus.

Table 5.5.15: spellcost.2da additional columns

Additional Column	Type	Description
SpellIndex	Integer	Index into spells.2da

Table 5.5.16: srcost.2da additional columns

Additional Column	Type	Description
Value	Integer	Amount of spell resistance

Table 5.5.17: weightcost.2da additional columns

Additional Column	Type	Description
Value	Float	Amount by which to muliply the item's weight to obtain its new, modified weight.

5.6. Item Param Tables

The table iprp_paramtable.2da defines the available param tables.

Table 5.6: iprp_paramtable.2da columns

Column	Type	Description
Name	Integer	StrRef of Param name
Lable [sic]	String	Programmer label
TableResRef	String	ResRef of Param 2da

ItemProperty Param tables

All param tables contain the columns specified in Table 5.6.1.

Table 5.6.1: param table 2da columns

Column	Type	Description
Name	String	StrRef of the name of the param.
Label	String	Programmer label

Some param tables contain additional columns beyond those specified in Table 5.6.1. Those tables are detailed below.

Table 5.6.2: onhitdur.2da additional columns

Additional Column	Type	Description
EffectChance	Integer	Percent chance to cause the onhit effect when landing a hit.
DurationRounds	Integer	Number of rounds of duration of the effect if saving throw failed.

Table 5.6.3: weightinc.2da additional columns

Additional Column	Type	Description
Value	Float	Additional weight in pounds.

5.7. Miscellaneous Item Tables

5.7.1. Item Valuation

Table 5.7.1.1: itemvalue.2da columns

Column	Type	Description
LABEL	String	Programmer label. A String referring to the character level that the row corresponds to. Row 0 is level 1, row 1 is level 2, and so on.
DESIREDTREASURE	Integer	Always 0
MAXSINGLEITEMVALUE	Integer	Cost of the most expensive item that a character of level (row+1) can use.
TOTALVALUEFILTER	Integer	Unused.

Table 5.7.1.2: skillvsitemcost.2da columns

Column	Type	Description
DeviceCostMax	Integer	Cost of the most expensive item that can be identified with a Lore skill check equal to the row index.
SkillReq_Class	Integer	Required number of ranks in Use Magic Device skill to be able to use an item that has a cost equal to or less than the DeviceCostMax, if the item type is not normally useable any of the character's classes, but is not restricted by race or alignment
SkillReq_Race	Integer	Required number of ranks in Use Magic Device skill to be able to use an item that has a cost equal to or less than the DeviceCostMax, if the item type is not normally useable by the character's race.
SkillReq_Align	Integer	Required number of ranks in Use Magic Device skill to be able to use an item that has a cost equal to or less than the DeviceCostMax, if the item type is not normally useable by the character's alignment.

5.7.2. Armor Statistics

Table 5.7.2.1: capart.2da columns

Column	Type	Description
NAME	String	Always 0.
MDLNAME	String	<bodypart> portion of an armor part ResRef. See **Section 4.1.4**.
NODENAME	String	Node on the base model to append this part model to.

Table 5.7.2.2: armor.2da columns

Column	Type	Description
ACBONUS	Integer	Base AC bonus of the item
DEXBONUS	Integer	Max Dexterity bonus when wearing armor of this AC
ACCHECK	Integer	Armor Skill Check penalty
ARCANEFAILURE%	Integer	Percent chance of Arcane Spell Failure
WEIGHT	Integer	Weight of armor having this AC
COST	Integer	BaseCost for an armor item that has this AC
DESCRIPTIONS	Integer	StrRef of a qualitative description of this armor type, that does not include statistics.
BASEITEMSTATREF	Integer	StrRef of description that includes statistics on the armor type.

5.7.3. Weapon Combat Sounds

The **weaponsounds.2da** file specifies what sounds play when weapons of specific types hit targets of various material types. Each row refers to a weapon type. In baseitems.2da, the *WeaponMatType* column value is an index into weaponsounds.2da. The non-label columns in weaponsounds.2da specify the ResRefs of wave files to play when a weapon hits the material type named in the column.

Material type is determined by the appearance of the object being hit:

For creatures, the *SOUNDAPPTYPE* in **appearance.2da** indexes into **appearancesndset.2da**. If the row in appearancesndset.2da has **** values, then the material type is determined by the armor worn by the creature. The base AC of the creature's armor is an index into **defaultacsounds.2da**, where the *ArmorType* string value serves as a column name in **weaponsounds.2da** by randomly appending a "0" or a "1" to the end of it.

For placeables, the *SoundAppType* in **placeables.2da** indexes into **placeableobjsnds.2da**, in which the *ArmorType* string value serves as a column name in weaponsounds.2da by randomly appending a "0" or a "1" to the end of it.

Table 5.7.3.1: weaponsounds.2da columns

Column	Type	Description
Label	String	Programmer label
Leather0 Leather1	String	-
Chain0 Chain1	String	-
Plate0 Plate1	String	-
Stone0 Stone1	String	In addition to those cases where the target really does use stone as its material type for onhit sounds: If the target has Stoneskin, Shadowskin, or Petrification, then use this sound instead of what would normally be used.
Wood0 Wood1	String	In addition to those cases where the target really does use wood as its material type for onhit sounds: Played when target has Barkskin.
Chitin0 Chitin1	String	-
Scale0 Scale1	String	-
Ethereal0 Ethereal1	String	-
Miss0 Miss1	String	-
Parry0	String	Played on a parry or on a miss that caused the parry animation to play.
Critical0	String	Played on a critical hit.

Table 5.7.3.2: appearancesndset.2da columns

Column	Type	Description
Label	String	Programmer label
ArmorType	String	Specifies set of columns in **weaponsounds.2da** to use when creature is hit. **** means to use the ArmorType from **armourtypes.2da**

		for the armor that the creature is wearing.
WeapTypeL WeapTypeR WeapTypeS WeapTypeClsLW WeapTypeClsH WeapTypeRch MissIndex	Integer	Row in **weaponsounds.2da** to use when creature attacks and hits. In order, these columns correspond to the following attacks: left swing, right swing, stab, close low, close high, far reach, and miss **** means to use the WeaponMatType of the creature's equipped weapon from **baseitems.2da**.
Looping	String	ResRef of looping WAV to emanate from the creature
FallFwd	String	ResRef of WAV to play when creature falls forward.
FallBck	String	ResRef of WAV to play when creature falls backward.

Table 5.7.3.3: defaultacsounds.2da columns

Column	Type	Description
ArmorIndex	Integer	Index into **parts_chest.2da** that matches up to the armor the creature is wearing.
ArmorType	String	Armor type. Use as column name in **weaponsounds.2da**.