BioWare Aurora Engine
Creature Format

1. Introduction

A **Creature** is an object that can move around in the game and interact with other objects such as doors, placeable objects, items, encounters, triggers, or other creatures. The behaviour of a creature is controlled by a set of scripts, and it may also be controlled by a Dungeon Master player who possesses it.

A **Player Character** (PC) is a specialized Creature that does not have AI scripts, but is instead controlled directly by a player.

Creatures 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.

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

Player Characters can be saved as standalone character files, or as instances in a savegame. Character files are BIC files in a player's localvault directory, dmvault directory, or on a server's servervault directory. A BIC file is a GFF, and has "`BIC `" as the FileType string in its header. Player characters in a savegame are stored as Structs within a module's IFO file.

2. Creature Struct

The tables in this section describe the GFF Struct for a Creature. Some Fields are only present on Instances and others only on Blueprints. Still others are present only in toolset data or only in the savegames.

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

2.1 Common Creature Fields

2.1.1. Fields in All Creatures

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

Table 2.1.1: Fields in all Creature Structs

Label	Type	Description
`Appearance_Type`	WORD	Index into **appearance.2da**.
`BodyBag`	BYTE	Index into **bodybag.2da**. Specifies the appearance of the bodybag that this Creature leaves behind after its corpse fades, if it had dropped any Items on death, and if the *Lootable* Field is 0. See **Table 4.5.2** in the Doors and Placeable Objects document.
`Cha`	BYTE	Charisma Ability Score, before any bonuses/penalties
`ChallengeRating`	FLOAT	Calculated Challenge Rating. See **Section 3.1. Challenge Rating**.
`ClassList`	List	List of Class Structs, having StructID 2. Must always contain at least one element, and can have up to 3 elements.

		See **Section 2.1.2. Fields in Class Struct**.
Con	BYTE	Constitution Ability Score, before any bonuses/penalties
Conversation	CResRef	ResRef of the Conversation file for this creature. This conversation runs when a script tells the creature to run ActionStartConversation().
CRAdjust	INT	Adjustment to the creature's Challenge Rating. To get the Creature's final Challenge Rating, add this value to the *ChallengeRating* Field. See **Section 3.1** for more details.
CurrentHitPoints	SHORT	The Creature's current hit points, not counting any bonuses. This value may be higher or lower than the creature's maximum hit points. See **Section 3.4. Hit Points** for more details.
DecayTime	DWORD	If the *Lootable* Field is 1, then this is the number of milliseconds that pass before the creature's corpse fades away after all Items have been removed from the the corpse. If the *Lootable* Field is 0, then this is the number of milliseconds that pass after the creature dies before its corpse fades away. After the corpse fades, and if the Creature had any Items that dropped on death, the corpse will be replaced by a bodybag placeable object that contains the Items.
Deity	CExoString	Name of the Creature's Deity. Not used directly by the game, but scripts can check the value of this Field.
Description	CExoLocString	Description of the object as seen when using the Examine action in the game.
Dex	BYTE	Desterity Ability Score, before any bonuses/penalties
Disarmable	BYTE	1 if the Creature can be disarmed, 0if not.
Equip_ItemList	List	List of EquippedItem Structs. StructID is equal to the item slot bit flag for the equipped item: HEAD 0x1 CHEST 0x2 BOOTS 0x4 ARMS 0x8 RIGHTHAND 0x10 LEFTHAND 0x20 CLOAK 0x40 LEFTRING 0x80 RIGHTRING 0x100 NECK 0x200 BELT 0x400 ARROWS 0x800 BULLETS 0x1000 BOLTS 0x2000 The Structs themselves differs between creature blueprints and instances.
FactionID	WORD	Faction ID of the Creature. This is an index into the *FactionList* Field of the module’s **repute.fac** file.
FeatList	List	List of Feat Structs. StructID 1.
FirstName	CExoLocString	First name.
fortbonus	SHORT	Fortitude save bonus. Usually 0.
Gender	BYTE	Index into **gender.2da**. 0 is assumed to be male, and 1 female, by hardcoded

		convention. The 2da serves mainly to specify the StrRef to display the name of the gender to the user.
GoodEvil	BYTE	Alignment on the Good-Evil axis. 0 is the most Evil value, and 100 is the most Good value.
HitPoints	SHORT	Base Maximum Hit Points, not considering any bonuses. See **Section 3.4** for more details.
Int	BYTE	Intelligence Ability Score, before any bonuses/penalties
Interruptable	BYTE	1 if a conversation with this creature can be interrupted, 0 otherwise
IsImmortal	BYTE	1 if the Creature can never die, and can never drop below 1 Hit Point.
IsPC	BYTE	1 if the Creature is a Player Character; 0 otherwise.
ItemList	List	List of InventoryObjects in the creature's backpack **See Section 3. Inventory Objects**, in the Items GFF document.
LastName	CExoLocString	Last name.
LawfulChaotic	BYTE	Alignment on the Law-Chaos axis. 0 is the most Chaotic value, and 100 is the most Lawful value.
Lootable	BYTE	1 if the Creature leaves behind a lootable corpse. 0 if the Creature leaves behind a bodybag placeable object instead.
MaxHitPoints	SHORT	Maximum Hit Points, after considering all bonuses and penalties.
NaturalAC	BYTE	Natural AC bonus.
NoPermDeath	BYTE	1 if the Creature cannot permanently die. 0 if the Creature can permanently die. Permanent death is otherwise known as explosive death or chunky death. Note that this setting does not prevent the creature's corpse from fading away when it dies. Corpse fade is a separate mechanism from death itself. To prevent the corpse from fading, call the SetIsDestroyable() scripting function on the Creature with an argument of FALSE, or set the *Lootable* Field to 1.
PerceptionRange	BYTE	Index into **ranges.2da**. Must be 9 to 13.
Phenotype	INT	Phenotype of the Creature, applicable only if its *Appearance_Type* Field indexes a row of **appearance.2da** where the *MODELTYPE* is "P". 0 = normal 1 = fat
Plot	BYTE	1 if creature is Plot, 0 if not
PortraitId	WORD	Index into **portraits.2da**
Race	BYTE	Index into **racialtypes.2da**
refbonus	SHORT	bonus to Reflex saving throw.
ScriptAttacked	CResRef	OnPhysicalAttacked event
ScriptDamaged	CResRef	OnDamaged event
ScriptDeath	CResRef	OnDeath event
ScriptDialogue	CResRef	OnConversation event
ScriptDisturbed	CResRef	OnInventoryDisturbed event
ScriptEndRound	CResRef	OnEndCombatRound event
ScriptHeartbeat	CResRef	OnHearbeat event
ScriptOnBlocked	CResRef	OnBlocked event
ScriptOnNotice	CResRef	OnPerception event

`ScriptRested`	CResRef	OnRested event
`ScriptSpawn`	CResRef	OnSpawnIn event
`ScriptSpellAt`	CResRef	OnSpellCastAt event
`ScriptuserDefine`	CResRef	OnUserDefined event
`SkillList`	List	List of Skill Structs (StructID 0). The index of the Skill Struct in the Creature's *SkillList* matches up on a one-to-one basis with the rows in **skills.2da**. There should be the same number of elements in *SkillList* as there are rows in **skills.2da**.
`SoundSetFile`	WORD	Index into **soundset.2da**. See **Section 7** of the *Sound Set File* document.
`SpecAbilityList`	List	List of SpecialAbility Structs (StructID 4)
`StartingPackage`	BYTE	Index into **packages.2da**. Specifies the package that this creature levels up in when using the LevelUpHenchman() scripting function.
`Str`	BYTE	Strength Ability Score, before any bonuses/penalties
`Subrace`	CExoString	Subrace string. Not used by game, but scripts can check this value.
`Tag`	CExoString	Tag of this object
`Tail`	BYTE	Index into **tailmodel.2da**.
`WalkRate`	INT	Index into **creaturespeed.2da**.
`willbonus`	SHORT	Bonus to Will saving throw
`Wings`	BYTE	Index into **wingmodel.2da**.

2.1.2. Fields in Class Struct

The Table below lists the Fields that are present in a Creature Class Struct.

Table 2.1.2.1: Fields in Class Struct (StructID 2)

Label	Type	Description
`Class`	INT	Index into **classes.2da**.
`ClassLevel`	SHORT	Level in the Class specified by the *Class* Field.

Caster classes that prepare their spells, such as Wizards and Clerics, have a list of Prepared spells, as given in the table below. Bards and Sorcerers do not have Memorized lists.

Table 2.1.2.2: Additional Fields in Class Struct for Casters that Prepare Spells

Label	Type	Description

`MemorizedList0`
`MemorizedList1`
`...`
`MemorizedList9`
	List	List of memorized spells

Caster classes that have a limited number of known spells per level keep track of those spells in the knownspell lists, given in the table below. For player characters, the knownspell lists are present for Wizards, Bards, and Sorcerers, but not for divine casters, since divine casters automatically know all spells at each spell level open to them. For nonplayer characters, the knownspell list is only present for Bards and Sorcerers. NPC wizards do not have spellbooks, and can only use the spells in their Memorized lists.

Table 2.1.2.3: Additional Fields in Class Struct for Casters that Do Not Prepare Spells

Label	Type	Description

`KnownList0`
`KnownList1`
`...`
	List	List of known spells

`KnownList9`		

The MemorizedLists and KnownLists contain Spell Structs. The game and toolset differ in what Fields the Spell Structs contain. Refer to **Table 2.1.5: Toolset Fields in all Spell Structs (known and memorized) (StructID 3)** and **Table 2.5.3: Game Fields in MemorizedSpell Struct (StructID 3)** and **Table 2.5.4: Game Fields in KnownSpell Struct (StructID 3)** for details.

2.1.3. Fields in Other Listed Structs

The Table below lists the Fields that are present in a Creature Feat Struct found in the *FeatList*.

Table 2.1.3.1: Fields in Feat Struct (StructID 1)

Label	Type	Description
`Feat`	WORD	Index into **feat.2da**.

The Table below lists the Fields that are present in a Creature Skill Struct found in the *SkillList*.

Table 2.1.3.2: Fields in Skill Struct (StructID 0)

Label	Type	Description
`Rank`	BYTE	Skill Rank. The index of the Skill Struct in the Creature's *SkillList* matches up on a one-to-one basis with the rows in **skills.2da**. There should be the same number of elements in *SkillList* as there are rows in **skills.2da**.

The Table below lists the Fields that are present in a Creature Special Ability Struct found in the *SpecAbilList*.

Table 2.1.3.3: Fields in SpecialAbility Struct (StructID 4)

Label	Type	Description
`Spell`	WORD	Index into **spells.2da**.
`SpellCasterLevel`	BYTE	Spell caster level to cast this spell as
`SpellFlags`	BYTE	Bit flags. Can have one or more of the following flags: 0x01: readied; this flag is always set by the toolset 0x02: spontaneously cast 0x04: unlimited use

2.2. Creature Blueprint Fields

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

Table 2.2: Fields in Creature Blueprint Structs

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

Table 2.2: Fields in Creature Blueprint EquippedItem Structs

Label	Type	Description
`EquipRes`	CResRef	ResRef of the Equipped Item.

2.3. Creature Instance Fields

A Creature 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 Creature 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 creature's orientation

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

2.4. Creature Toolset Fields

Some Creature Fields are only present in blueprints and instances created in the toolset, but not in the game.

Table 2.1.5: Toolset Fields in all Spell Structs (known and memorized) (StructID 3)

Label	Type	Description
`Spell`	WORD	Index into **spells.2da**.
`SpellFlags`	BYTE	General bit flags. Can have one or more of the following flags: 0x01: readied; this flag is always set by the toolset 0x02: spontaneously cast 0x04: unlimited use
`SpellMetaMagic`	BYTE	Metamagic type. These values look like they can be bit flags, but the game only supports one at a time. Do not add these values together to get multiple metamagic effects on a single spell, because the resulting behaviour is undefined. 0x00: none 0x01: empower 0x02: extend 0x04: maximize 0x08: quicken 0x10: silent 0x20: still

Table 2.1.5: Fields in SpecialAbility Struct (StructID 4)

Label	Type	Description
`Spell`	WORD	Index into **spells.2da**.
`SpellCasterLevel`	BYTE	Spell caster level to cast this spell as
`SpellFlags`	BYTE	Metamagic bit flags. Can have one or more of the following flags: 0x01: readied; this flag is always set by the toolset 0x02: spontaneously cast 0x04: unlimited use

2.5. Creature Game Instance Fields

After a GIT file has been saved by the game, the Creature Instance Struct contains not just the Fields in Table 2.1.1 and Table 2.3, but also those Fields in Table 2.5.

INVALID_OBJECT_ID is a special constant equal to 0x7f000000 in hex, or 2130706432 in decimal.

Table 2.5.1: Fields in Creature Instance Structs in SaveGames

Label	Type	Description
`ActionList`	List	List of Actions queued up on this creature. See **Common GFF Structs** document, **Section 6**.
`Age`	INT	0 for non-player creatures. For player characters, this is the Age entered during character creation.
`AmbientAnimState`	BYTE	
`AnimationDay`	DWORD	
`AnimationTime`	DWORD	
`Appearance_Head`	BYTE	
`AreaId`	DWORD	ObjectId of area containing creature
`ArmorPart_RFoot`	BYTE	
`BaseAttackBonus`	BYTE	
`BodyBagId`	DWORD	
`CombatInfo`	Struct	StructID 51882
`CombatRoundData`	Struct	StructID 51930
`CreatureSize`	INT	Index into creaturesize.2da, and matches up to hardcoded constants in the game.
`DeadSelectable`	BYTE	1 if the creature is dead and selectable. That is, mousing over it causes it to highlight. 0 otherwise.
`DetectMode`	BYTE	1 if creature is in detect mode, 0 otherwise
`EffectList`	List	list of Effects on this creature. See **Common GFF Structs** document, **Section 4**.
`Experience`	DWORD	0 for non-player characters
`ExpressionList`	List	StructID 5 ExpressionId INT ExpressionString CExoString
`FamiliarName`	CExoString	
`FamiliarType`	INT	
`FortSaveThrow`	CHAR	
`Gold`	DWORD	Amount of gold being carried by the creature
`IsCommandable`	BYTE	
`IsDestroyable`	BYTE	
`IsDM`	BYTE	1 if the creature is a DM; 0 otherwise
`IsRaiseable`	BYTE	1 if the creature can be raised; otherwise
`Listening`	BYTE	
`MasterID`	DWORD	
`MClassLevUpIn`	BYTE	
`ObjectId`	DWORD	Object ID used by game for this object.
`OverrideBAB`	BYTE	0 to use normal BAB calculated based on levels in each class. Otherwise, specifies a BAB that overrides the normal one.
`PerceptionList`	List	StructID 0 ObjectId DWORD PerceptionData BYTE 3

PersonalRepList	List	List of PersonalReputation Structs (StructID 0xABED) describing how other creatures feel about this one.
PM_IsPolymorphed	BYTE	
PregameCurrent	SHORT	62
RefSaveThrow	CHAR	
SitObject	DWORD	ObjectID of the Placeable Object that the creature is sitting on
SkillPoints	WORD	0
StealthMode	BYTE	1 if the creature is in stealth mode, 0 otherwise
VarTable	List	List of scripting variables stored on this object. StructID 0. See **Section 3** of the **Common GFF Structs** document.
WillSaveThrow	CHAR	

Table 2.5.2: Additional Fields in Class Struct (StructID 2)

Label	Type	Description
Domain1 Domain2	BYTE	Index into **domains.2da**.
School	BYTE	Present only for Wizards. Index into **spellschools.2da**.
SpellsPerDayList	List	List of SpellsPerDay Structs (StructID 17767), specifying how many more spells the creature can cast at each spell level. There are always 10 elements in this list. The index of a list element is equal to the spell level that the element corresponds to (eg., element 0 is for cantrips, and element 9 is for Level 9 spells). This list is only present for classes that cast spells per day, such as Bards. This list is not present for spell-slot classes such as Wizards.

Table 2.5.2: Fields in SpellsPerDay Struct (StructID 17767)

Label	Type	Description
NumSpellsLeft	BYTE	Let the index of this element in the *SpellsPerDayList* be the spell level, then this Field's value is equal to the number of spells left at this spell level.

Table 2.5.3: Game Fields in MemorizedSpell Struct (StructID 3)

Label	Type	Description
Ready	INT	1 if the spell is readied for casting
Spell	WORD	Index into **spells.2da**.
SpellMetaMagic	SHORT	Same meaning as in toolset. See Table

Table 2.5.4: Game Fields in KnownSpell Struct (StructID 3)

Label	Type	Description
Spell	WORD	Index into **spells.2da**.

Table 2.5.5: Fields in PersonalReputation Struct (StructID 0xABED)

Label	Type	Description
Amount	INT	Reputation adjustment amount. Describes how the reputation of this creature has been adjusted in the eyes of another creature. For example, hitting another creature would typically set this Amount to -100.
Day	DWORD	Specifies game time at which this PersonalReputation object was created.
Decays	BYTE	1 if the reputation adjustment decays after a set time

		0 if it does not decay
`Duration`	INT	Duration in seconds of reputation adjustment
`ObjectId`	DWORD	Object ID of the other creature for which the reputation adjustment Amount applies.
`Time`	DWORD	Specifies game time at which this PersonalReputation object was created.

2.6. Player Fields

Player Structs in a savegame or in a BIC file contain all the Fields in Table 2.1.1 and 2.5.1, plus those in Table 2.6.1 below.

Table 2.6.1: Additional Fields in Player Structs (StructID 0xBEAD)

Label	Type	Description
`Age`	INT	
`Experience`	DWORD	
`QBList`	List	List of 36 QuickBar Structs having StructID 0. Describes the player's QuickBar assignments. Elements 0 to 11 are for the normal QuickBar. Elements 12 to 23 are for the Shift-QuickBar. Elements 24 to 35 are for the Control-QuickBar.

Table 2.6.2: Creature Fields not found in Player Structs

Label
`ChallengeRating`
`Conversation`
`Comment`
`TemplateResRef`

Player Instance Structs exist in the *Mod_PlayerList* Lists in **module.ifo**. A Player Instance Struct contains all the Fields in Table 2.1.1, 2.5.1, 2.6.1, plus those in Table 2.6.3 below.

Table 2.6.3: Additional Fields in Player Instance Structs (StructID 0xBEAD)

Label	Type	Description
`Mod_CommntyName`	CExoString	Player Name
`Mod_FirstName`	CExoLocString	Character's First Name. Same as *FirstName* Field.
`Mod_IsPrimaryPlr`	BYTE	
`Mod_LastName`	CExoLocString	Character's Last Name. Same as *LastName* Field.
`Mod_MapAreasData`	Binary	
`Mod_MapDataList`	List	List of Structs. StructID 0. Each Struct has a single Binary Field called Mod_MapData.
`Mod_MapNumAreas`	INT	
`ReputationList`	List	List of Structs. StructID 47837. Each Struct has a single INT Field called Amount that gives the player's rating from 0 to 100 with each faction in the module. There is one Struct per Faction, in the same order as given in the module Faction file **repute.fac**.

2.6.4. QuickBar

Table 2.6.4.1: Fields in QuickBar Empty Structs (StructID 0)

Label	Type	Description
`QBObjectType`	BYTE	0 if qbar slot is empty. If slot is empty, none of the other Fields are present. 1 = item 2 = spell 3 = skill 4 = feat 5 = script 6 = dialog 7 = attack 8 = emote 9 = castspell itemproperty 10 = mode toggle 38 = possess familiar 39 = associate command 40 = examine 41 = barter 42 = quickchat 43 = cancel polymorph 44 = spell-like ability

Table 2.6.4.2: Fields in QuickBar Item Structs (StructID 0)

Label	Type	Description
`QBCastPropIndex`	BYTE	0xFF if no cast property
`QBCastSubPropIdx`	BYTE	0xFF if no subproperty
`QBContReposX` `QBContReposY`	BYTE	0xFF if not inside a container
`QBItemInvSlot`	DWORD	object ID
`QBItemReposX` `QBItemReposY`	BYTE	location of item in inventory
`QBObjectType`	BYTE	1 for items

Table 2.6.4.2: Fields in QuickBar Spell Structs (StructID 0)

Label	Type	Description
`QBDomainLevel`	BYTE	0 for most spells. Domain level for cleric domain spells.
`QBINTParam1`	INT	Index into **spells.2da**.
`QBMetaType`	BYTE	MetaMagic flags on a spell, if applicable.
`QBMultiClass`	BYTE	Index into creature's *ClassList*.
`QBObjectType`	BYTE	2 for spells

Table 2.6.4.3: Fields in QuickBar Skill Structs (StructID 0)

Label	Type	Description
`QBINTParam1`	INT	Index into **skills.2da**.
`QBObjectType`	BYTE	3 for skills

Table 2.6.4.4: Fields in QuickBar Feat Structs (StructID 0)

Label	Type	Description
`QBINTParam1`	INT	Index into **feat.2da**.
`QBObjectType`	BYTE	4 for feats

Table 2.6.3.5: Fields in QuickBar Mode Structs (StructID 0)

Label	Type	Description
`QBINTParam1`	INT	0 for detect mode 1 for stealth mode
`QBObjectType`	BYTE	10 for modes

3. Calculations and Procedures

3.1. Challenge Rating

3.1.1. Additive CR

The Challenge Rating of a Creature is calculated using many sources. Below is the step-by-step procedure for calculating Challenge Rating.

Add up all the following:

HD * 0.15

(Natural AC bonus) * 0.1

[(Inventory Value) / (HD * 20000 + 100000)] * 0.2 * HD

[(Total HP) / (Average HP)] * 0.2 * HD * (Walk Rate) / (Standard Walk Rate)

[(Total of all Ability Scores) / (HD + 50)] * 0.1 * HD

[(Total Special Ability Levels) / { (HD * (HD + 1)) + (HD * 5) }] * 0.15 * HD

[(Total Spell Levels) / { (HD * (HD + 1)) }] * 0.15 * HD

[((Bonus Saves) + (Base Saves)) / (Base Saves)] * 0.15 * HD

[(Total Feat CR Values) / (HD * 0.5 + 7)] * 0.1 * HD

then multiply the total sum by the racial challenge rating modifer, the *CRModifier* value from **racialtypes.2da**, using the creature's race as an index into the 2da. The resulting value is the **Additive CR**.

The following are some explanations of the terms in the above formulae:

HD = Hit Dice = total number of levels the creature has in all of its classes.

Natural AC Bonus

Inventory Value = total value of all items in the Creature's inventory, not counting items equipped in the creature weapon slots or the creature hide slot. See **Section 4.4** of the **Items** document for how to calculate item cost.

Total HP = total hit points, not including bonuses from constitution or feats.

Average HP = average hit points based on average hit point dice rolls for each class. For example, a creature having classes Outsider 7/Fighter 3 (d8 and d10 hit dice, respectively) would have Average HP = [7 * (8 + 1)/2] + [3 * (10+1)/2] = 7*4.5 + 3*5.5.

Walk Rate = creature's walk rate from the *WALKRATE* column of **creaturespeed.2da**, using the creature's *WalkRate* GFF Field as the 2da row index.

Standard Walk Rate = The *WALKRATE* column value for row 0 in **creaturespeed.2da**, which corresponds to player characters' movement rate.

Total of all Ability Scores = total of all ability scores, before any modifications due to race, feats, or equipped items.

Total Special Ability Levels = total levels in all special abilities in the creature's *SpecAbilityList* GFF List. For each spell in the special ability list, get its *Innate* value from **spells.2da**, and add that to the total special ability levels.

Total Spell Levels = total levels of all spells in all the spell lists in all the creature's classes. The level of a spell is taken as its *Innate* value from **spells.2da**. If the *Innate* level is 0, then count it as 0.5.

Bonus Saves = total of the creature's *fortbonus*, *refbonus*, and *willbonus* GFF Fields.

Total Feat CR Values = total CR Values of all feats that the creature has. The CR Value of a feat is its value under the *CRValue* column in **feat.2da**.

3.1.2. Calculated CR

The **Calculated CR** is the final CR value before any manual challenge rating adjustments or roundoffs. If the Additive CR is less than 1.5, then the Calculated CR is adjusted slightly downward from there.

```
If 1.5 > AdditiveCR > 0.75, then

    AdditiveCR = AdditiveCR - 0.25

Else if AdditiveCR <= 0.75, then

    AdditiveCR = AddtiveCR - 0.35

End if

CalculatedCR = AdditiveCR
```

3.1.3. Final CR

The **Final CR** derived from the Calculated CR is always a whole number, unless it is less than 1, in which case, it must be one of the following fractional ratings: 1/2, 1/3, 1/4, 1/6, or 1/8. The **fractionalcr.2da** file gives the cutoff values for each fractional CR. The Final CR also includes any manual challenge rating adjustments specified in the Advanced page of the Creature Properties dialog in the toolset (that is, the value of the creature's *CRAdjust* GFF Field).

```
RoundedCR = AddtiveCR, rounded off to nearest integer

AdditiveCR = AdditiveCR + CRAdjust
```

```
RoundedCR = RoundedCR + CRAdjust

If AddtiveCR > 0.75, then

    FinalCR = RoundedCR

Else

    Find the first row in fractionacr.2da where the value in the Min column is less than AdditiveCR.

    Take the value in the Denominator column and divide it by 1 to get the FinalCR.

End if
```

3.2. Ability Scores

The ability scores saved with a Creature GFF Struct are not necessarily the same as those observed for the creature in the game.

A creature's final ability scores in the game are dynamically calculated from their base values. Only the base values themselves are saved in a Creature Struct.

Ability scores are dynamically adjusted based on:

- Racial modifiers, specified by the *StrAdjust*, *DexAdjust*, etc., columns in **racialtypes.2da**.
- Feats, such as Great Strength
- Modifiers from Effects, such as from spells or items

3.3. Saving Throws

A Creature's saving throws are not stored in its GFF Creature Struct. Instead, saving throws are determined dynamically.

First, the base saves for a creature are determined by adding up the saving throws contributed by each class that the creature has. The saving throw contribution of each class is obtained by looking up the class's saving throw table in the *SavingThrowTable* column of **classes.2da**. In the 2da specified by that column, the saving throws are found in the row that has the same *Level* label as the creature's level in the class.

A creature's final saving throws are dynamically calculated from their base values.

Things that may affect saving throws are:

- Constitution, Dexterity, and Wisdom ability bonuses. Ability scores themselves are affected by several things, as detailed in **Section 3.2**.
- Feats, such as Iron Will.
- Modifiers from Effects, such as from spells or items.

- Bonus saving throw values set in the Creature Properties dialog, and saved as the *fortbonus*, *refbonus*, and *willbonus* GFF Fields.

3.4. Hit Points

There are several different hit point values that are saved with a creature.

3.4.1. HitPoints

The *HitPoints* GFF Field contains the creature's Base Maximum Hit Points, not considering any bonuses. This represents the total number of hit points gained by rolling hit dice at levelup.

For a rules-compliant creature, this should not be less than the creature's character level (ie. total levels in all classes), and should not be higher than the total if every die roll was maximized. The toolset does allow non-rules-compliant creatures though, so smaller and larger numbers are actually possible.

> Example: Suppose we have a level 5 Barbarian that rolled 12, 6, 6, 7, 7 for hit points. Its *HitPoints* would be 12+6+6+7+7 = 38.

3.4.2. MaxHitPoints

Maximum Hit Points, after considering all bonuses and penalties.

> Example: Suppose that the level 5 Barbarian in the example above has a constitution of 16, but has no active feats or effects that raise hit points. It then has a +3 modifier due to constitution. Multiplying that by 5 levels gives +15 HP, for a *MaxHitPoints* of 38 + 15 = 53.
>
> Example2: Suppose we have a level 3 elven Commoner that rolled 1, 1, 1 for hit points. The Commoner has Constitution 8, adjusted down to 6 for being an elf, for a resulting -2 ability modifier. This creature's *HitPoints* are 3, but its *MaxHitPoints* are also 3, because all creatures are required to have a minimum of 1 hit point per level, even if ability modifiers would normally result in less.

3.4.3. CurrentHitPoints

The Creature's current hit points, not counting any bonuses. This value may be higher or lower than the creature's maximum hit points.

The toolset always creates creatures that will by default spawn into the game at exactly full health, with neither damage, nor bonus hit points. Thus, *CurrentHitPoints* is always equal to *HitPoints* for a creature created in the toolset.

In the game, this value reflects any damage the creature may have taken or any bonus hit points that the creature may have gained. *CurrentHitPoints* = *HitPoints* - (total damage taken).

Note that a creature that has been reduced to 0 hit points in the game does *not* necessarily have 0 *CurrentHitPoints*, because *CurrentHitPoints* does not consider hit point bonuses.

> Example: Suppose that the level 5 Barbarian in the examples above has been reduced to 0 hit points ingame. The barbarian has lost 53 hit points from maximum, so the *CurrentHitPoints* are 38 - 53 = -15.

3.5. INI List File Format

The Creature Wizard, Creature Levelup Wizard, and Creature Template system all use a set of INI files having a common format. These INI files are called List Files.

3.5.1. INI File format

INI List files conform to the standard Windows INI file format. They are plain text documents containing a number of *Sections*, *Keys*, and *Values*. The start of a section is designated by a single line containing the section name in square brackets, like this:

```
[Name of Section]
```

Each section can contain zero or more Key/Value pairs, with one pair per line, and written as follows:

```
Key1=Value1
Key2=Value2
```

It is legal for a key to have no value specified, as in:

```
Key=
```

3.5.2. List File format

In an INI List file, if a key is specified with no value, or if a key is not present at all, then an attempt to read its value will return a default of 0 or an empty string, depending on whether the key data is an integer or a string.

The following example illustrates the types of Keys that may be present in an INI List file, with explanatory comments set off by green-colored text preceded by semi-colons. Note that a real INI List file does not include any comments. Any string values that correspond to ResRefs are not case-sensitive.

```
[HALFLING_WIZARD_VAMPIRE_EXAMPLE]
; Bonus ability scores. Add these to the creature's current
; ability scores.
STR=0
INT=0
WIS=0
DEX=0
CON=0
CHA=0
PORTRAIT=         ; Default portrait base resref. Can be blank.
GENDER=M          ; Default gender; see GENDER column in gender.2da.
ALIGNGOODEVIL=50  ; default evil-good alignment. 0-100
ALIGNLAWCHAOS=50  ; default chaos-law alignment. 0-100
; Additional feats.
FEATCOUNT=5       ; Number of additional feats in this section
; There should be as many FEATLABEL keys as specified by
; the FEATCOUNT. Each FEATLABEL value should be the exact text
; of the label of a feat in feat.2da.
FEATLABEL1=skillaffinitymovesi
FEATLABEL2=skillaffinitylisten
FEATLABEL3=lucky
FEATLABEL4=fearless
FEATLABEL5=good
BASEARMORCLASS=0    ; bonus to natural AC
; Saving throw bonuses. These are in addition to the creature's
; base saves, and are added to the totals saved to the
; fortbonus, refbonus, and willbonus GFF Fields
SAVEFORT=0
SAVEREF=0
SAVEWILL=0
; Additional Spells. Obsolete.
; No longer used as of game version 1.60, toolset 1.3.0.0, vts 025
; Use the packages system instead.
SPELLCOUNT=4
SPELLLABEL1=Daze
SPELLLEVEL1=0
SPELLLABEL2=Ray_of_Frost
SPELLLEVEL2=0
SPELLLABEL3=Ray_of_Frost
SPELLLEVEL3=1
SPELLLABEL4=Magic_Missile
SPELLLEVEL4=1
; Additional Skills. Obsolete.
; No longer used as of game version 1.60, toolset 1.3.0.0, vts 025
; Use the packages system instead.
SKILLCOUNT=3
SKILLLABEL1=Concentration
SKILLRANK1=4
SKILLLABEL2=Lore
SKILLRANK2=4
SKILLLABEL3=Spellcraft
SKILLRANK3=4
; Default Equipped Item ResRefs
HEAD=
CHEST=NW_CLOTH005
BOOTS=
ARMS=
RHAND=NW_WSWDG001
LHAND=
CLOAK=
LRING=
RRING=
NECK=
BELT=
ARROWS=
BULLETS=
BOLTS=
; Unequipped items to add to creature inventory
UNEQUIPPEDCOUNT=2
UNEQUIPPED1=NW_IT_TORCH001
UNEQUIPPED2=NW_IT_MPOTION001
; Default Scripts ResRefs
ONHEARTBEAT=NW_C2_Default1
ONNOTICE=NW_C2_Default2
ONENDCOMBATROUND=NW_C2_Default3
ONSPELLCASTAT=NW_C2_DefaultB
ONMELEEATTACKED=NW_C2_Default5
ONDAMAGED=NW_C2_Default6
ONINVENTORYDISTURBED=NW_C2_Default8
ONDIALOGUE=NW_C2_Default4
ONSPAWN=NW_C2_Default9
ONRESTED=NW_C2_DefaultA
ONDEATH=NW_C2_Default7
```

```
ONUSERDEFINED=NW_C2_DefaultD
ONBLOCKED=NW_C2_DefaultE
; Additional special abilities.
; SPECIALABILITYCOUNT is the number of additional
; special abilities in this section of the INI file
SPECIALABILITYCOUNT=2
; There should be as many SPECIALABILITYLABEL,
; SPECIALABILITYCASTERLEVEL, and SPECIALABILITYUNLIMITEDUSE
; keys as specified by the SPECIALABILITYCOUNT.
; Each SPECIALABILITYLABEL value should be the exact text of the
; label of a spell in spells.2da.
SPECIALABILITYLABEL1=Gaze_Dominate
; SPECIALABILITYCASTERLEVEL is the caster level for the ability
SPECIALABILITYCASTERLEVEL1=1
; SPECIALABILITYUNLIMITEDUSE specifies if this ability can be
; used unlimited times per day.
; If its value is zero, then the entry counts as a single use
; of the ability per day. Additionay uses per day require
; duplicate entries.
SPECIALABILITYUNLIMITEDUSE1=0
SPECIALABILITYLABEL2=Aura_Fear
SPECIALABILITYCASTERLEVEL2=10
SPECIALABILITYUNLIMITEDUSE2=0
; Creature Template properties
SUBRACESTRREF=5644   ; StrRef of string to add to SubRace Field
SUBRACE=             ; literal text of string to add to SubRace
NEWRACE=UNDEAD       ; Label of new Race in racialtypes.2da.
NEWHD=12             ; New HD to apply to all class levels
; Creature item blueprint resrefs
CQUALITIES=NW_CREITEMVAM   ; creature hide item resref
; Creature weapon item resrefs for weapons slots 1, 2, 3
CLAW1=
CLAW2=
CLAW3=
; Special creature weapon modifier item resrefs.
; For more details, see Section 3.8. Applying Creature Templates
CWSLASH=                   ; slash weapon modifier
CWPIERCE=                  ; pierce weapon modifier
CWSLASHPIERCE=NW_CREWPVBT  ; slash+pierce weapon modifier
CWBLUDGEON=                ; slam weapon modifier
CWALL=                     ; all weapons modifier
```

3.6. Creature Wizard Race Initialization

The first step to creating a creature in the Creature Wizard is to pick its race from among those defined in **racialtypes.2da**. Several starting characteristics are defined in racialtypes.2da (Appearance, default portrait from appearance, racial feats, default class), but some others are obtained from a Race INI file.

The ResRef of the race INI file is **race_<label>**, where <label> is the value from the *Label* column in racialtypes.2da, truncated to 11 characters if it is longer than that, to ensure that the final ResRef is 16 characters or less. The race INI file contains one section, having the same name as the ResRef itself, but in all-caps and without the "RACE_" prefix. (eg., [HUMANOID_MO])

A Race INI file is a list file (See **Section 3.5. INI List File Format**). It contains a single section, named after the <label> portion of its ResRef.

The following example displays some of the keys that can be present in a race section. Not all of them need to be present.

```
[HALFLING]
STR=0
INT=0
WIS=0
DEX=0
CON=0
CHA=0
PORTRAIT=
GENDER=M
PHENOTYPE=
ALIGNGOODEVIL=50
ALIGNLAWCHAOS=50
```

In addition to the above, a race section can also include feats, equipped inventory, and unequipped inventory. See **Section 3.5.2. List File format** for the exact keynames of these properties.

Although ability modifiers are included in the race list file, please note that these are NOT the standard racial ability score modifiers. The standard racial modifiers are in **racialtypes.2da**, so any modifiers in the race list file are in addition to those in racialtypes.2da.

3.7. Auto-levelup

Creatures can be automatically leveled up by the toolset's Creature Wizard or Creature Levelup Wizard, or by the game's scripting function LevelUpHenchman().

When a creature is auto-leveled, the toolset or game requires a *Package* to determine what ability, skill, feat, and spell choices to make. Creature levelup packages include the same Packages that are available to players at character creation. Autoleveling up a non-player creature has results that are similar to what would happen if a player clicked the Recommended button at every opportunity during character creation and levelup.

There are some minor differences between how the game handles autolevelup and how the toolset does it. Most of these differences stem from the toolset ignoring certain restrictions.

3.7.1. Check Class Requirements

Check if the creature meets the class prerequisites, by check the following columns in **classes.2da**: *MaxLevel*, *AlignRestrict*, *AlignRstrctType*, *InvertRestrict*, *PreReqTable*. See **Table 5.3.1: classes.2da**. Note that the toolset does not actually prevent taking a class with an incompatible alignment. Instead, it shifts the creature's alignment by the minimum amount required to make it conform to the requirements of the class level being added.

The *PreReqTable* points to a separate 2da that contains additional prerequisites for taking prestige classes. The requirements are detailed in **Table 5.3.6: Prestige Class** Prerequisites Table: cls_pres_*.2da.

3.7.2. Determine levelup package

In the game, using LevelUpHenchman() specifies the Class and Package to use for levelup.

In the Creature Wizard and Creature Levelup Wizard, the procedure is different:

Use the creature's *StartingPackage* GFF Field value as an index into **packages.2da** and check the *ClassID* for the package. If the package *ClassID* matches the class being leveled up in, then use the *StartingPackage* as the levelup package.

If the *StartingPackage ClassID* does not match the class being leveled up in, then find the first package in **packages.2da** that has a *ClassID* that matches the levelup class.

If the creature is being leveled up for the first time because it is being created in the Creature Wizard, then its *StartingPackage* defaults to the first package in **packages.2da** that has a *ClassID* that matches the creature's first class.

3.7.3. Apply Class List File

If adding level 1 in a class, and the class is the creature's first class, then read the default scripts from the Primary section of the C*lass INI File.*

The ResRef of the class INI file is **class_<label>**, where <label> is the value from the *Label* column in classes.2da, truncated to 10 characters if it is longer than that, to ensure that the final ResRef is 16 characters or less.

A Class INI file is a list file (See **Section 3.5. INI List File Format**). It contains 1 section for every supported class level, except for level 1, which has 2 sections. The sections for levels 2 and up are named after the class's truncated <label> from its ResRef, with a space followed by the level number. Level 1 has two sections, a primary section and a secondary section, with " Primary" and " Secondary" appended to the section names.

Examples of class INI file sections are:

```
[WIZARD 1 Primary]
[WIZARD 1 Secondary]
[WIZARD 2]
[WIZARD 3]
```

Each section describes things to be applied when gaining the appropriate level in the class. Level 1 is special in that gaining level 1 has different effects depending on if it is the creature's very first level in any class, or if the creature multiclassed. The Primary section applies for a creature's first class only. The Secondary section applies when multiclassing.

As of game version 1.60 and toolset version 1.3.0.0, vts025, the only keys used from a class list file are the default scripts from the primary section:

```
ONHEARTBEAT=
ONNOTICE=
ONENDCOMBATROUND=
ONSPELLCASTAT=
ONMELEEATTACKED=
ONDAMAGED=
ONINVENTORYDISTURBED=
ONDIALOGUE=
ONSPAWN=
ONRESTED=
ONDEATH=
ONUSERDEFINED=
ONBLOCKED=
```

For reference, however, older versions of the toolset also loaded the following keys

```
STR=
INT=
WIS=
DEX=
CON=
CHA=
BASEARMORCLASS=
SAVEFORT=
SAVEREF=
SAVEWILL=
```

plus the keys for special abilities, spells, feats, skills, equipped inventory, and unequipped inventory.

Loading of these keys, however, has been superceded by usage of the Package system. The list file method to gain skills and spells did not take into account differing numbers of skill points or spell slots due to ability scores. The packages system does.

3.7.4. Add or Initialize Ability Scores

If adding level 1 in the creature's very first class, its ability scores are set to the default values specified in **classes.2da** in the *Str*, *Dex*, *Con*, *Wis*, *Int*, and *Cha* columns.

If adding any other level in a class, if the creature's new total level in all classes is divisible by 4, then it gains a bonus ability point. If the class being raised is a class for which the creature's *StartingPackage* applies, then ability score that raises is the one specified by the *Attribute* column in **packages.2da**. Otherwise, the ability that raises is the one specified by the *PrimaryAbil* column in **classes.2da** for the class being leveled up in.

3.7.5. Add Skills

Calculate Skill Points

Determine the number of skill points available by reading the *SkillPointBase* from **classes.2da**. Add the creature's intelligence ability bonus to the number of skill points. The intelligence modifier is:

[(creature's *Intelligence*) + (its racial intelligence modifier from **racialtypes.2da**)] /2 **-** 5

rounded down the nearest integer. The total skill points cannot be less than 1, however.

If the creature has the Quick to Master feat (hardcoded feat number 258), then it gets an additional skill point.

Multiply the final number of skill points by 4 if adding level 1 of the creature's first class.

Class Skills Table

Determine the *Class Skills Table* (**cls_skill_*.2da**) for the class being leveled up in by getting the *SkillsTable* from **classes.2da**. There are two columns in the Class Skills Table: *SkillIndex*, an index into **skills.2da**; and *ClassSkill*, which contains a 1 if the skill is a class skill, or 0 if not. If a class cannot take a skill at all (eg., Clerics can't take Perform), then that skill does not appear at all in the Class Skills table.

Package Skill Preferences

Determine the *Package Skills Preference Table* (**packsk*.2da**) for the levelup package by getting the *SkillPref2DA* from **packages.2da**.

Iterate through the Package Skill Preferences 2da from top to bottom. If the *SkillIndex* at the current row in the Package SkillPref table corresponds to a class skill as defined by the Class Skills Table, then add 1 rank to the creature's ranks in that skill. A class skill may not have a rank that is higher than the creature's total level in all classes + 3.

Continue looping through the SkillPref table, adding 1 to each class skill until all skill points have been spent, or all class skills are maximized at CharacterLevel + 3. If the end of the SkillPref table is reached, repeat from the top.

If there are skill points left over after every class skill has been maximized, then continue looping through the SkillPref table, but adding cross-class skills instead, at a cost of 2 skill points each. A cross class skill cannot exceed total character level divided by 2, rounded down. Continue adding cross class skills until the creature has 0 or 1 skill points remaining or until every cross-class skill has been maximized, then stop.

3.7.6. Add Feats

Class Feats Table

In **classes.2da**, the *FeatsTable* column specifies the *Class Feats Table* (**cls_feat_*.2da**) for each class. The Class Feats Table describes what feats are available to a class, whether the class automatically gains a feat at a certain level, and whether the feat is a bonus feat that can only be taken as a bonus feat, or can also be taken as a normal feat. See **Table 5.3.3: Class Feats** Table: cls_feat_*.2da for more details.

Package Feat Preferences

To determine which feats the creature takes on levelup, use the *Package Feat Preference Table* (**packft*.2da**) specified in the *FeatPref2DA* column of **packages.2da**. The Feat Preference table lists the preferred feats for the creature's levelup package in order of most preferred to least preferred. The exact usage of this table is outlined in more detail further below.

Calculate number of normal feats

At character level 1, and at every character level divisible by 3, a creature gains a feat.

If the creature has the Quick to Master feat, then at character level 1, it gets 2 normal feats instead of just 1.

Calculate number of bonus feats

Certain classes gain bonus feats at specific class levels. These levels are specified in the *Class Bonus Feats Table* (**cls_bfeat_*.2da**), which is specified in the *BonusFeatsTable* column of **classes.2da**. The *Bonus* column in **cls_bfeat_*.2da** is 0 if there are no bonus feats at a particular level, or 1 if there is 1 bonus feat.

Feat prerequisites

To take a feat, the creature must meet the prerequisites for it as defined by the following columns from **feat.2da**: PreReqEpic, MINATTACKBONUS, MINSTR, MINDEX, MININT, MINWIS, MINSPELLLVL, PREREQFEAT1, PREREQFEAT2, OrReqFeat, REQSKILL, MinLevel, MinLevelClass, MaxLevel. See **Table 5.4.1: feat.2da** for details as to what each of these columns mean.

A creature cannot take a feat more than once unless the *GAINMULTIPLE* column value in **feat.2da** is 1.

Successor feats

Some feats are successors to other feats. If a feat has a value specified in the *SUCCESSOR* column in **feat.2da**, then if the creature gains the successor feat, the original feat is removed. For example, Sneak Attack 3 is the successor for Sneak Attack 2. If a creature gains Sneak Attack 3, it loses Sneak Attack 2.

Assign class feats

Iterate through the Class Feats Table and find all feats that have 3 as their *List* column value, and *GrantedOnLevel* equal to the level of the class being leveled up in. Add these feats to the creature's feat list. For example, when adding level 5 of a class, add all feats that have *List*=3 and *GrantedOnLevel*=5.

Cleric Domain feats

Clerics gain bonus feats at class level 1 for their chosen domains. If the creature is gaining Cleric level 1, then for each of the cleric's domains specified in the *Domain1* and *Domain2* columns in **packages.2da**, get the *GrantedFeat* column value in **domains.2da**, and add the specified feat.

Assign bonus feats

If a feat in the Class Feats Table has a *List* column value of 1 (bonus or normal) or 2 (bonus only), then it can be taken as a bonus feat. Find all the feats in the Class Feats Table that can be gained as bonus feats.

To determine which bonus feats the creature takes on levelup, use the *Package Feat Preference Table* (**packft*.2da**) specified in the *FeatPref2DA* column of **packages.2da**. Scan through the Feat Preference Table from top to bottom, using the *FeatIndex* column values as indices into **feat.2da**. If the creature meets the prerequisites for a feat, and the feat can be taken as a bonus feat, then add the feat to the creature. Continue doing this until the creature has taken as many bonus feats as it is allowed to for the current level, starting over from the top of the list if the bottom has been reached. Stop if a full pass through the list has been done without adding any feats.

Assign normal feats

If a feat in the Class Feats Table has a *List* column value of 0 (normal choosable) or 1 (bonus or normal), then it can be taken as a normal feat. Use the same procedure as for bonus feats to add normal feats to a creature.

3.7.7. Add Spells

Class SpellGain Table

To determine if a class is a spellcasting class, get the Class Spell Gain Table (**cls_spgn_*.2da**) from **classes.2da** by reading from the *SpellGainTable* column. If the class has **** in that column, then skip the rest of this section. The SpellGain table specifies the base number of spell slots or spells per day that the class has per day at each spell level. See **Table 5.3.7: Class Spell** Gain Table: cls_spgn_*.2da for details.

A creature cannot cast spells of a given spell level unless its spellcasting ability score bonus is at least equal to the spell level itself. The spellcasting ability score is listed under the *PrimaryAbil* column in **classes.2da**.

The number of bonus spell slots or spells per day that a creature has for a given spell level is:

(Bonus Spells) = (Ability Bonus) **-** (Spell Level)

Treat a negative result as 0.

Class Spells Known

Some spellcasting classes have a limited number of spells that they can know at each spell level. For these classes, the *Class Spells Known Table* (**cls_spkn_*.2da**) is specified under the *SpellKnownTable* column of **classes.2da**. If the *SpellKnownTable* value is ****, then the class does not have a limit on number of spells known.

Package Spell Preferences

To determine which spells the creature learns or prepares on levelup, use the *Package Spell Preference Table* (**packsp*.2da**) specified in the *SpellPref2DA* column of **packages.2da**.

Adding Spells

If the levelup class is one that prepares spells in advance, then spells are added to its MemorizedLists. If the levelup class is one that does not prepare spells, then spells are added to its KnownLists.

To add new spells, do the following steps for each spell level that has a column in the SpellGain/SpellKnown table:

1. Determine how many spells the creature has at this spell level. If the creature has a SpellsKnown table, then use the value from the appropriate column of the cls_spkn 2da. Otherwise, use the appropriate column in the cls_spgn 2da. If the creature uses the SpellGain table, then the creature may also have bonus spell slots at this level. Add those to the total number of spellslots for the current spell level. If the final total number of spell slots or spells known is 0, skip to the next spell level.

2. Determine if the creature gains any new spells at this spell level. To do this, subtract the number obtained in Step 1 from the number of spells the creature currently has in its *KnownList* (if the creature has a SpellsKnown table) or *MemorizedList* (if the creature has no SpellsKnown table) for this spell level. The resulting difference is the number of new spells that the creature gains. If the result is 0, skip to the next spell level.

3. If the levelup class is Cleric, then add domain spells, if any. Check the appropriate *Level_* column in **domains.2da** for extra domain spells at the current spell level, and add those spells to the creature's MemorizedList. For each spell added, subtract from the number of new spells calculated in Step 2.

4. Iterate through the Package Spell Preference Table from top to bottom, starting over from the top after reaching the bottom, perform the following:

   a) Read the *SpellIndex* column value for the current row in the SpellPreference 2da. Use it as an index into **spells.2da**.

   b) Look up the spell in **spells.2da**, using the *Bard*, *Cleric*, *Druid*, *Paladin*, *Ranger*, or *Wiz_Sorc* column as appropriate to determine the spell level. If none of these columns matches the levelup class, use the *Innate* column instead. If the spell level does not match the spell level for which we are currently adding spells, then skip to the next SpellPreference row.

   c) If the spell is not already in the creature's MemorizedList for the current spell level, then add it, flagging it as Readied, and with no Metamagic. Subtract 1 from the number of new spells calculated in Step 2. If the value becomes 0, stop adding spells.

   d) If a complete pass of the SpellPreference 2da has been done with no spells added, but there are still new spells to add, continue to Step 5.

5. Repeat step 4, but add a spell even if it is already in the MemorizedList. Continue looping through the spell preference list until a full pass is made where no spells are added, then stop adding spells, even if there are still new spells to add.

3.7.8. Add Package Equipment

If adding level 1 of a class, open the Package Equipment Table (**packeq*.2da**) specified in the *Equip2DA* column of **packages.2da**. Otherwise, skip this step.

The *Label* columns contain ResRefs of Item Blueprints (UTI files). Add all the Items in the Package Equipment table to the creature's inventory.

If an Item is equippable, then equip it to the appropriate inventory slot, unless there is already an Item there.

3.7.9. Add Hit Points

Add hit points from the levelup. Get the *HitDie* column value from **classes.2da** to determine the size of the hit point die roll.

If the class is the creature's first class, the *PlayerClass* column value in **classes.2da** is a 1, and the level is 1, then increase the creature's *CurrentHitPoints*, *HitPoints*, and *MaxHitPoints* Fields by the full hit die roll for that class. Otherwise, add the average die roll for the class (DieSize/2 + 0.5), keeping fractional hit point values until the Creature Wizard or Creature Levelup Wizard has finished adding all class levels. After all levels have been added, the total hit point value is rounded to the nearest integer.

3.8. Applying Creature Templates

A Template is a set of properties that modify an existing creature. Examples of Templates are Vampire, Half-Dragon, and Lich. Templates are applied using the List File system (see **Section 3.5. INI List File Format**). The list of available Templates is provided in **crtemplates.2da**.

3.8.1. Apply List File

The *NAME* column of **crtemplates.2da** lists the Template labels. The filename of a Template List file is **tmlt_<label>.ini**, where <label> is the *NAME* value.

Template List files contain a single section having the same name as the label.

When applying a Template to a creature, the following List File keys are used: ALIGNGOODEVIL, ALIGNLAWCHAOS, SUBRACESTRREF, SUBRACE, NEWRACE, NEWHD, HitDie, BONUSAC, and the keys for ability scores, saving throws, special abilities, skills, and feats.

The Creature Item Listfile keys are also used. They are describedin more detail in **Section 3.8.4. Creature Weapon Changes** and **Section 3.8.5. Creature Hide Changes**.

3.8.2. Hit Die Changes

If a NEWHD key is specified, then the creature's Hit Die for all class levels changes to the NEWHD value, *if* the NEWHD value is larger. The creature gains additional Hit Points assuming average die rolls for the new Hit Die size.

> Example: Suppose that a Wizard 4/Cleric 5/Barbarian 6 acquires a Template that has NEWHD=8. The creature's old class hit die sizes are d4, d8, and d12. The creature would gain 4 hit points for Wizard level 1 because the first class gets maximum hit die rolls, and 8 - 4 = 4. The creature would gain (8-4)/2 = 1 additional Hit Point for Wizard levels 2 to 4. The creature would gain no additional Hit Points for its Cleric and Barbarian levels because those classes already have Hit Dice that are greater than or equal to the Template Hit Die.

3.8.3. Race/Subrace Changes

If a NEWRACE is specified, then the creature's *Race* Field changes to the row number in **racialtypes.2da** that has the same *Label* as that specified by the NEWRACE value.

If a SUBRACESTRREF is specified, then fetch the string for that StrRef from dialog.tlk and include it in the creature's S*ubRace* Field.

If a SUBRACE is specified, then add the SUBRACE value directly to the creature's *SubRace*. Ignore SUBRACE if SUBRACESTRREF is already specified.

When adding a Subrace string, set the creature's *SubRace* Field to the new subrace string if the *SubRace* Field was originally empty. If there was already text in the *SubRace* Field, then check if the new subrace string is already part of the *SubRace* Field. If not, then prepend the new subrace to the beginning of the existing *SubRace* string with a "; " separator between the new subrace and the old text.

3.8.4. Creature Weapon Changes

The creature item keys are resolved using a very specific order and method.

Specific Damage-type Weapon Modifiers

The CWSLASH, CWPIERCE, CWSLASHPIERCE, and CWBLUDGEON keys are Creature Weapon modifiers, and are applied first, and in the order listed in this sentence.

Each of these keys refers to an Item Blueprint (UTI file).

For each key, do the following for each of the creature's 3 Creature Weapon slots:

1. Check if there is an item in the current slot. If not, skip to the next slot

2. Check if the current slot's item has the same damage type as that for the current weapon modifier. If not, skip to the next slot

3. Load the weapon modifier's item blueprint and add its properties to those of the current slot's item.

Additional Normal Creature Weapons

After that, the normal creature weapons CLAW1, CLAW2, and CLAW3 are added to the creature's CreatureWeapon item slots if those slots are not already occupied. Note that the 1, 2, and 3 do not specify exact slot numbers. For example, if a creature already has an item in its Claw1 and Claw2 slot, then the CLAW1 item will go into the Claw3 slot, and CLAW2 and CLAW3 will not be added at all.

All-weapon modifier

After handling the creature weapon modifiers and the normal creature weapons, the CWALL key is applied. It is a Creature Weapon modifier that is applied to all creature weapons regardless of their damage type.

If the creature is a Creature Blueprint, then if any weapons were modified by the CWSLASH, CWPIERCE, CWSLASHPIERCE, CWBLUDGEON, or CWALL modifiers, the new creature weapons are saved as new Item Blueprints with user-specified ResRefs.

3.8.5. Creature Hide Changes

The CQUALITIES key specifies an Item Blueprint for the creature's Hide item. If the creature does not already have a hide item, the item having the ResRef specified by the CQUALITIES key value is loaded and added to the creature's Hide slot.

If the creature already has a hide item, then the CQUALITIES item's properties are added to those of the creature's existing hide item. If the creature is a Creature Blueprint, then the resulting item is saved as a new Item Blueprint with a user-specified ResRef.

5. Creature-related 2DA Files

5.1. Appearance

The appearance 2da defines all the Creature appearances that exist. Many characteristics of a Creature are determined by its Appearance. These characteristics are defined in appearance.2da.

Table 5.1.1: appearance.2da

Column	Type	Description
LABEL	String	programmer label

STRING_REF	Integer	StrRef of the name of the appearance type, as it appears in the Appearance dropdown in the toolset
NAME	String	programmer label
RACE	String	If *MODELTYPE* is not "P", then this is the ResRef of the MDL file to use for the creature model. If *MODELTYPE* is "P", then this is the player model letter used in constructing the complete creature model. For example, if *RACE* is "D", then chest part 3 for a normal-phenotype male creature is pmd0_chest003.
ENVMAP	String	"default": use the default environment map for the current area's tileset, as specified in the .SET file's *[General] EnvMap* property. ****: use no environment map on the creature model Interpret any other value as the ResRef of the TGA file to use as the environment map for the creature.

BLOODCOLOR	String	R = red
		G = green
		W = white
		Y = yellow
		N = none

MODELTYPE	String	P = player: creature model is composed of multiple body parts each with their own MDL and textured with PLTs. Model changes when armor is worn, and colors are selectable. S = simple: creature model is a single MDL and textured with a single TGA or DDS texture file. Colors are not selectable. Model does not change when wearing armor. Weapons do not appear when equipped. F = same as simple, but weapon items do appear when equipped in right or left hand inventory slots. L = large: same as F, but only right-hand weapon appears.
WEAPONSCALE	Float	Size scaling factor to apply to weapon models equipped by creatures having this appearance. Only meaningful if *MODELTYPE* is not S.
WING_TAIL_SCALE	Float	Size scaling factor to apply to wings or tails attached to the creature model.
HELMET_SCALE_M	Float	Size scaling to apply to helms equipped by male creatures. Only meaningful if *MODELTYPE*=P
HELMET_SCALE_F	Float	Size scaling to apply to helms equipped by female creatures. Only meaningful if *MODELTYPE*=P
MOVERATE	String	Default walking/running speed for creatures having this appearance. Specifies a row in **creaturespeed.2da** that has this value in its *2DAName* column.
WALKDIST	Float	Distance in metres travelled by creature from the beginning of its walk animation to the end of its walk animation
RUNDIST	Float	Distance in metres travelled by creature from the beginning of its run animation to the end of its run animation
PERSPACE	Float	Personal space used to determine if the creature will fit

		through an opening
CREPERSPACE	Float	Personal space used for combat. Usually larger than *PERSPACE*.
HEIGHT	Float	Height of the creature. Used for pathfinding under obstacles and zoomin camera height.
HITDIST	Float	When this creature is attacking another creature, subtract the *HITDIST* from the actual distance between attacker and target before comparing the distance to the *PREFATCKDIST*.
PREFATCKDIST	Float	Preferred distance from which to attack a target. Creature will use short-range, normal-range, or long-range versions of its melee animations depending on distance of the target.
TARGETHEIGHT	String	Target height when hitting creatures having this appearance H = normal height, used by most appearances L = low, used by short appearances, such as badgers
ABORTONPARRY	Integer	1 if attack animation aborts when the attacked creature plays the parry animation
RACIALTYPE	Integer	Index into **racialtypes.2da**. Default racialtype of creatures having this appearance.
HASLEGS	Integer	1 if the appearance has legs, 0 otherwise. The Feat "Called Shot: Leg" only works if the creature has legs.
HASARMS	Integer	1 if the appearance has arms, 0 otherwise. The Feat "Called Shot: Arm" only works if the creature has arms.
PORTRAIT	String	Base ResRef of the default portrait for creatures having this appearance. Example: if PORTRAIT is po_badger, then use the portraits po_badger_h.tga, po_badger_l.tga, po_badger_m.tga, etc. This value should not exceed 14 characters in length.
SIZECATEGORY	Integer	Index into creaturesize.2da, references hard-coded list of creature size definitions in game engine.
PERCEPTIONDIST	Integer	Default perception range in metres for creatures having this appearance
FOOTSTEPTYPE	Integer	-1 if makes no sound when walking or running Otherwise, index into **footstepsounds.2da**.
SOUNDAPPTYPE	Integer	Index into **appearancesndset.2da**. See **Table 5.7.3.2** in the **Items** GFF document.
HEADTRACK	Integer	1 if the creature's head tracks nearby creatures, speakers in a conversation, or objects being moused over by the player. 0 otherwise.
HEAD_ARC_H	Float	Maximum angle in degrees that the creature's head will turn to the side when tracking something.
HEAD_ARC_V	Float	Maximum angle in degrees that the creature's head will tilt up or down when tracking something.
HEAD_NAME	String	Name of the head node to rotate in the creature's model in order to make its head track an object.
BODY_BAG	Integer	Index into bodybag.2da, specifying the default bodybag to leave behind when the creature dies and its corpse fades.
TARGETABLE	Integer	1 if the creature can be targetted, such as by mousing over it.

		0 if the creature cannot be targetted

Table 5.1.2: tailmodel.2da

Column	Type	Description
LABEL	String	programmer label
MODEL	String	ResRef of the MDL file to use for the tail

Table 5.1.3: wingmodel.2da

Column	Type	Description
LABEL	String	programmer label
MODEL	String	ResRef of the MDL file to use for the wings

Table 5.1.4: creaturespeed.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef to display when selecting speed in the toolset. If StrRef is ****, as it is for row 0 (playerspeed), it is unselectable in the toolset.
2DAName	String	String value used in **Appearance.2da** under the *MOVERATE* column to specify the default creature speed for a given appearance.
WALKRATE	Float	Walking speed of the creature in m/s
RUNRATE	Float	Running speed of the creature in m/s

Table 5.1.5: phenotype.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef

Table 5.1.6: creaturesize.2da

Column	Type	Description
LABEL	String	Programmer label describing the size category of the current row.
ACATTACKMOD	Integer	Attack modifier when a creature of the specified size is attacking a medium-sized creature. Not used.
STRREF	Integer	StrRef of the name of the size category. Not used.

Table 5.1.7: footstepsounds.2da

Column	Type	Description
Label	String	programmer label
Dirt0 Dirt1 Dirt2	String	ResRef of WAV to play when stepping on a suface of the specified type. There are 3 sound variations for each surface material, played at random at each footstep.
Grass0 Grass1 Grass2		
Stone0 Stone1 Stone2		
Wood0 Wood1 Wood2		
Water0 Water1 Water2		

Carpet0 Carpet1 Carpet2		
Metal0 Metal1 Metal2		
Puddles0 Puddles1 Puddles2		
Leaves0 Leaves1 Leaves2		
Sand0 Sand1 Sand2		
Snow0 Snow1 Snow2		

5.2. Races

Table 5.2: racialtypes.2da

Column	Type	Description
Label	String	programmer label
Abbrev	String	Obsolete. Unused.
Name	Integer	StrRef of the race name, capitalized. eg, Dwarf
ConverName	Integer	StrRef of the race name as an adjective. eg., Dwarven
ConverNameLower	Integer	Lower-case version of ConverName. eg. dwarven
NamePlural	Integer	StrRef of the race name in plural and capitalized. eg, Dwarves
Description	Integer	StrRef of a description of the race
Appearance	Integer	Index into **appearance.2da**. Default appearance for a creature of this race.
StrAdjust DexAdjust IntAdjust ChaAdjust WisAdjust ConAdjust	Integer	Racial ability modifier. Applied dynamically by the game. For example, if StrAdjust is 2, and the creature has a Strength of 12 ingame when unbuffed and naked, then its Strength is stored as 10.
Endurance	Integer	Obsolete. Unused.
Favored	Integer	Index into **classes.2da**. Favored class for this race. If ****, then favored class is creature's highest class.
FeatsTable	String	ResRef of racial feats table
Biography	Integer	StrRef of default biography for player characters of this race
PlayerRace	Integer	1 if players can choose this race at character creation, 0 if not.
Constant	String	Identifier to use when referring to this race in a script. Must match constant defined for it in nwscript.nss.
Age	Integer	Default Age of a player character of this race.
ToolsetDefaultClass	Integer	Index into **classes.2da**. Default class for this race when creating a creature in the Creature Wizard.
CRModifier	Float	Modifier used in CR calculation for creatures of this race.

5.3. Classes

Table 5.3.1: classes.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef of the class name. eg. Barbarian
Plural	Integer	StrRef of the plural class name. eg. Barbarians
Lower	Integer	lowercase cass name. eg. barbarian
Description	Integer	StrRef of description of the class
Icon	String	ResRef of TGA icon used in game GUIs to represent class
HitDie	Integer	Size of die to roll for hit points on leveling up in this class
AttackBonusTable	String	ResRef of class base attack bonus table (cls_atk_*)
FeatsTable	String	ResRef of class feats table, listing feats available (cls_feat_*)
SavingThrowTable	String	ResRef of class saves table, listing saving throws by level (cls_savethr_*)
SkillsTable	String	ResRef of class skills table (cls_skills_*)
BonusFeatsTable	String	ResRef of class bonus feats table, listing levels at which class gains bonus feats
SkillPointBase	Integer	Base number of skill points available on levelup in class
SpellGainTable	String	ResRef of class spellgain table (cls_spgn_*) **** if class does not cast spells
SpellKnownTable	String	ResRef of class spellknown table (cls_spkn_*) **** if class is not restricted to a certain number of known spells at each spell level, or if class does not cast spells
PlayerClass	Integer	1 if players can choose this class, 0 if not
SpellCaster	Integer	1 if the class is a spellcaster, 0 if not
Str Dex Con Wis Int Cha	Integer	Default starting ability scores for creatures created in Creature Wizard if this class is the creature's first class.
PrimaryAbil	String	Primary ability score. Autolevelup will always pick this ability to raise. For spellcaster classes, this is also the casting ability.

AlignRestrict	Integer	Bit field:
		neutral = 0x01
		lawful = 0x02
		chaotic = 0x04
		good = 0x08
		evil = 0x10

AlignRestrictType	Integer	Bit field: 0x01 = law-chaos axis 0x02 = good-evil axis
InvertRestrict	Integer	0 if alignment restrictions applied normally 1 if alignment restrictions applied inversely
Constant	String	Identifier used to refer to this class in a script. Must match constant defined for it in nwscript.nss.
EffCRLvl01 ... EffCRLvl20	Integer	Effective level of character for purposes of encounter challenge rating calculations used by original NWN Official Campaign. Ignored if not playing original Official Campaign.
PreReqTable	String	ResRef of prestige class prerequisites table. (cls_pres_*)

		**** if class can be taken at character level 1.
MaxLevel	Integer	Maximum level allowed in this class. Usually applies to prestige classes. 0 if no maximum exists.
XPPenalty	Integer	1 if this class counts toward multiclassing XP penalties
ArcSpellLvlMod	Integer	0 if this class does not affect arcane spellcasting level. If greater than zero, then let this value be X. For every X levels in this class, the character is treated as being one level higher in the character's first arcane spellcasting class, for purposes of calculating spell slots and spells per day.
DivSpellLvlMod	Integer	Same as ArcSpellLvlMod, but for divine spellcasting classes.
EpicLevel	Integer	-1 if class becomes epic after the default level, ie. 20. Otherwise, this value is greater than 0 and specifies the level beyond which the character is considered epic in this class.
Package	Integer	Default package for this class. Index into **packages.2da**.

Table 5.3.2: Class Base Attack Bonus Table: cls_atk_#.2da

Column	Type	Description
BAB	Integer	Base attack bonus gained from having level = (2da row) + 1

Table 5.3.2: Class Saving Throw Table: cls_savethr_*.2da

Column	Type	Description
Level	Integer	class level
FortSave	Integer	Fortitude save
RefSave	Integer	Reflex save
WillSave	Integer	Will save

Table 5.3.3: Class Feats Table: cls_feat_*.2da

Column	Type	Description
FeatLabel	String	programmer label
FeatIndex	Integer	index into **feat.2da**
List	Integer	-1: feat granted at creation, at level 1 0: feat is choosable 1: feat can be chosen as a bonus feat or normal feat 2: feat can only be chosen as a class bonus feat 3: feat is granted at the level specified by *GrantedOnLevel*.
GrantedOnLevel	Integer	class level at which feat is awarded -1 if feat is not automatically granted at any level, and must be chosen.
OnMenu	Integer	1 if appears on radial menu 0 if does not appear on radial menu

Table 5.3.4: Class Bonus Feats Table: cls_bfeat_*.2da

Column	Type	Description
Bonus	Integer	Determines whether a character gets a bonus feat at level = (2da row) + 1 in this class. 1 - bonus feat awarded 0 - no bonus feat

Table 5.3.5: Class Skills Table: cls_skill_*.2da

Column	Type	Description
SkillLabel	String	programmer label
SkillIndex	Integer	Index into **skills.2da**.
ClassSkill	Integer	1 if class skill, 0 if cross-class skill

Table 5.3.6: Prestige Class Prerequisites Table: cls_pres_*.2da

Column	Type	Description
LABEL	String	programmer label
ReqType	String	Several possible values, each one dictating how the *ReqParam* columns are interpreted: FEAT: required feat. *ReqParam1* indexes into **feats.2da**. FEATOR: must have at least one of the FEATOR requirements in order to take this prestige class. *ReqParam1* indexes into **feats.2da**. SKILL: *ReqParam1* indexes into **skills.2da**. *ReqParam2* specifies the required number of ranks in the specified skill. RACE: must be of one of the specified races. *ReqParam1* indexes into **racialtypes.2da**. BAB: base attack bonus must be greater than or equal to *ReqParam1*. VAR: the scripting variable named in *ReqParam1* column must exist on the creature and be set to the value in *ReqParam2*. Ignored by toolset. ARCSPELL: *ReqParam1* must be 1. Specifies that the character must be able to cast arcane spells. Ignored by toolset.
ReqParam1	varies	See *ReqType* for how to interpret this column.
ReqParam2	varies	See *ReqType* for how to interpret this column. **** if not required.

The spellgain 2das define the number of spells per day a caster can cast, or the number of spell slots in which a caster can prepare spells each day. All caster classes each have a spellgain 2da specified for them under the *SpellGainTable* column in classes.2da.

Table 5.3.7: Class Spell Gain Table: cls_spgn_*.2da

Column	Type	Description
Level	Integer	Class level label. Should be equal to row index + 1
NumSpellLevels	Integer	Number of spell levels available at this class level
SpellLevel0 SpellLevel1 ... SpellLevel9	Integer	For classes that must prepare their spells in advance, such as wizards and clerics, this is the number of base number of spell slots available at the spell level named in the column name. For classes that do not prepare their spells in advance, such

		as sorcerers and bards, this is the base number of spells per day at the spell level named in the column name. Note that not all class spellgain 2das will have spell levels up to 9. Value in the column is **** if there are no spells per day at this level. The actual number of spell slots or spells per day available to a creature is modified by the creature's ability score in the relevant casting stat. For sorcerer or bard-type casters, a 0 value in one of these columns means that there are no spells per day at this level unless the character qualifies for it by virtue of having a sufficiently high spellcasting ability score bonus.

The spellknown 2das define the number of spells that a caster knows at each spell level. Caster classes that do not prepare their spells in advance, such as sorcerers, each have a spellknown 2da specified for them under the *SpellKnownTable* column in classes.2da.

Table 5.3.8: Class Spells Known Table: cls_spkn_*.2da

Column	Type	Description
Level	Integer	Class level label. Should be equal to row index + 1
SpellLevel0 SpellLevel1 ... SpellLevel9	Integer	Number of spells known at the spell level named in the column name. Note that not all class spellknown 2das will have spell levels up to 9. **** if there are no spells known at this level.

Table 5.3.9: hen_companion.2da and hen_familiar.2da

Column	Type	Description
NAME	String	programmer label
BASERESREF	String	ResRef of UTC creature blueprint
STRREF	Integer	StrRef of name of associate
DESCRIPTION	Integer	StrRef of description of associate

5.4. Feats

Table 5.4.1: feat.2da

Column	Type	Description
LABEL	String	programmer label
FEAT	Integer	StrRef of the feat name
DESCRIPTION	Integer	StrRef of the feat description
ICON	String	ResRef of the TGA icon
MINATTACKBONUS	Integer	Minimum base attack bonus required to take this feat **** if no min attack bonus requirement
MINSTR MINDEX MININT MINWIS MINCON MINCHA	Integer	Minimum ability score to take this feat. **** if no minimum required ability score
MINSPELLLVL	Integer	Minimum spell level that the creature must be able to cast in order to take this feat. For example, Empower Spell

		requires that the creature can cast level 2 spells. **** if there is no spell level requirement
PREREQFEAT1 PREREQFEAT2	Integer	Index into **feat.2da** specifying feats that the creature must have in order to take this one.
GAINMULTIPLE		0 if feat cannot be gained multiple times 1 if feat can be gained more than once. Not supported at this time. Always 0.
EFFECTSSTACK	Integer	1 if effects from the feat stack with other effects of the same type 0 if effects do not stack
ALLCLASSESCANUSE	Integer	1 if all classes can use this feat, 0 if not
CATEGORY	Integer	Index into **categories.2da**. Not used.
MAXCR	Integer	not used
SPELLID	Integer	Index into **spells.2da** specifying a spell that implements this feat
SUCCESSOR	Integer	Index into **feat.2da** specifying a feat that succeeds this one. Example: if this feat is Elemental Shape, then the successor is Elemental Shape 2.
CRValue	Float	Challenge Rating weighting for this feat when calculating creature challenge rating. See **Section 3.1. Challenge Rating**.
USESPERDAY	Integer	Number times feat can be used per day. -1 if uses per day depends on certain hardcoded conditions such as number of levels in a class (Example, stunning fist). **** if feat can be used unlimited times per day or if feat is passive.
MASTERFEAT	Integer	Index into **masterfeats.2da**, specifying the "master feat" that this feat belongs to. Example: the "Improved Critical: Club" feat belongs the "Improved Critical" master feat.
TARGETSELF	Integer	1 if the feat targets oneself, **** if the feat does not.
OrReqFeat0 OrReqFeat1 OrReqFeat2 OrReqFeat3 OrReqFeat4	Integer	If any of the OrReqFeats are non-****, then the creature must have at least one of the OrReqFeats in order to take the current feat.
REQSKILL	Integer	Index into skills.2da specifying a required skill. **** if no skill required
ReqSkillMinRanks	Integer	Number of ranks required in the required skill
REQSKILL2	Integer	Index into skills.2da specifying a second required skill. **** if no skill required
ReqSkillMinRanks2	Integer	Number of ranks required in the second required skill
Constant	String	Identifier of scripting constant used to refer to this feat. Feat index and Constant name must match constant definition in nwscript.nss
TOOLSCATEGORIES	Integer	Specifies one of a set of harcoded feat categories used by toolset in Creature Properties dialog to allow filtering feat lists by feat category 1 = combat feat 2 = active combat feat 3 = defensive feat 4 = magical feat

		5 = other feat
HostileFeat	Integer	1 if using feat on another creature is considered a hostile act, **** if not.
MinLevel	Integer	Minimum level in *MinLevelClass* required to take this feat **** if no min level
MinLevelClass	Integer	Index into **classes.2da** specifying class in which creature must have *MinLevel* levels.
MaxLevel	Integer	Maximum character level to be able to take this feat. Example: the Luck of Heroes feat can only be taken at level 1.
MinFortSave	Integer	Minimum fortitude save to be able to take this feat. **** if there is no fortitude save requirement.
PreReqEpic	Integer	1 if feat can only be taken by epic characters. 0 if feat can be taken by non-epic characters.

Table 5.4.2: masterfeats.2da

Column	Type	Description
LABEL	String	programmer label
STRREF	Integer	StrRef of the master feat name
DESCRIPTION	Integer	StrRef of the master feat
ICON	String	ResRef of the TGA icon for the master feat

Table 5.4.3: categories.2da

Column	Type	Description
Category	String	programmer label

Table 5.4.4: race_feat_*.2da

Column	Type	Description
FeatLabel	String	programmer label
FeatIndex	Integer	Index into **feat.2da**.

5.5. Skills

Table 5.5: skills.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef of skill name
Description	Integer	StrRef of skill description
Icon	String	ResRef of skill TGA icon
Untrained	Integer	1 if skill can be used without training. 0 if must have at least 1 rank in skill to use it.
KeyAbility	String	Ability used to modify skill check. Possible values: STR, CON, DEX, INT, WIS, CHA
ArmorCheckPenalty	Integer	1 if skill is affected by armor check penalty, 0 if not.
AllClassesCanUse	Integer	1 if all classes can use this skill, 0 if not all classes can use this skill
Category	none	Unused. Always ****.
MaxCR	Integer	Maximum Talent CR for this skill in the game's Talent system. **** if no max
Constant	String	Identifier used to represent this skill in scripting. Must match constant defined for it in nwscript.nss.
HostileSkill	Integer	1 if using this skill on another creature is considered a

		hostile act, 0 if not.

5.6. Spells

Table 5.6.1: spells.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef of spell name
IconResRef	String	ResRef of TGA icon for this spell

School	String	Spell School. Possible values are as listed in the *Letter* column in **spellschools.2da**:
		A = abjuration
		C = conjuration
		D = divination
		E = enchantment
		I = illusion
		N = necromancy
		T = transmutation
		V = evocation

Range	String	Range of spell
		P = personal
		T = touch
		S = short
		M = medium
		L = long

VS	String	Verbal/Somatic v = spell is verbal only s = spell is somatic only vs = spell is verbal and somatic
Metamagic	Integer	Bit field specifying what metamagic feats are useable with this spell. 0x00: none 0x01: empower 0x02: extend 0x04: maximize 0x08: quicken 0x10: silent 0x20: still
TargetType	Integer	Bit field specifying what things this spell can target. 0x01: self 0x02: creature 0x04: ground 0x08: item 0x10: door 0x20: placeable 0x40: trigger
ImpactScript	String	ResRef of script to run when spell hits its target.
Bard Cleric Druid Paladin Ranger Wiz_Sorc	Integer	Spell level of this spell for the specified class
Innate	Integer	Spell level of this spell when used as a spell-like ability

ConjTime	Integer	Number of milliseconds to do the conjure animation
ConjAnim	String	Conjuration animation to use head: conjure is done with raised arms, head looking up hand: conjure is done with hands in front, head looking at hands
ConjHeadVisual ConjHandVisual ConjGrndVisual	String	ResRef of visual effect MDL to apply to the caster's head, hand or ground node during the conjure animation **** if no visual
ConjSoundMale	String	ResRef of WAV to play during the conjure if caster is male
ConjSoundFemale	String	ResRef of WAV to play during the conjure if caster is female
CastAnim	String	Animation to use for the cast. area: arms spread out to sides touch: one hand held out to touch target self: hands drawn in toward chest out: both hands aimed forward, arms outstretched up: both hands pointed up
CastTime	Integer	Number of milliseconds to hold the cast animation
CastHeadVisual CastHandVisual CastGrndVisual	String	ResRef of visual effect MDL to apply to the caster's model's headconjure, handconjure, or root node during the cast animation **** if no visual.
CastSound	String	ResRef of WAV to play on cast. **** if no sound.
Proj	Integer	1 if spell has a projectile, 0 if not
ProjModel	String	If *Proj*=1, then this is the ResRef of the MDL to use for the spell projectile. Otherwise, ****
ProjType	String	If *Proj*=1, then this is the projectile type. homing accelerating linked ballistic spiral bounce **** if no projectile
ProjSpawnPoint	String	Node to spawn the projectile at. hand: spawn at hand node monster0 monster1 monster2 monster3 monster4: spawn at specified special monster node ****: no projectile to spawn
ProjSound	String	ResRef of WAV to play from moving projectile
ProjOrientation	String	Orientation of projectile model path: along the path of travel **** no path
ImmunityType	String	Name of immunity type that works against this spell. Acid Cold Death Disease Divine

		Electricity Fear Fire Mind_Affecting Negative Poison Positive Sonic Not actually used by the game.
ItemImmunity	Integer	Not used by game.
SubRadSpell1 ... SubRadSpell5	Integer	Index into **spells.2da** specifying spells cast off of a subradial when casting this spell. SubRadSpell1 is the spell used when this spell is dragged directly from spellbook to quickbar.
Category	Integer	Index into **categories.2da** specifying the category of the spell as used by the talent system. An example usage of the *Category* is when a creature AI asks itself, "do I have any healing spells"?
Master	Integer	Index into **spells.2da** specifying the spell for which this spell is a subradial option. Reverse of the *SubRadSpell* columns.
UserType	Integer	1 = spell 2 = special ability 3 = feat 4 = item
SpellDesc	Integer	StrRef of spell description
UseConcentration	Integer	1 if should use Concentration checks when casting this spell
SpontaneouslyCast	Integer	1 if spell can be cast spontaneously, sacrificing another spell of equal level, 0 if not.
AltMessage	Integer	StrRef of an alternate message to display instead of "<creature> casts <spellname>". Example 1: " <CUSTOM0> uses breath weapon." Example 2: " <CUSTOM0> is surrounded by an aura." **** if there is no alternate message for this spell
HostileSetting	Integer	1 if this spell is hostile 0 if this spell is harmless
FeatID	Integer	Index into **feat.2da** pointing to the feat that this spell implements. The feat in turn has a *SPELLID* that points at this spell.
Counter1 Counter2	Integer	Index into **spells.2da** specifying spells that can be used as counterspells to this spell
HasProjectile	Integer	Should be same as *Proj*.

Table 5.6.2: spellschools.2da

Column	Type	Description
Label	String	programmer label
Letter	String	single letter used to identify the spell school, used in column *School* in **spells.2da**.
StringRef	Integer	StrRef of the spell school name
Opposition	Integer	Opposition school, index into **spellschools.2da**. **** if no opposition school.
Description	Integer	StrRef of spell school description

Table 5.6.3: domains.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef of name of domain
Description	Integer	StrRef of description
Icon	String	ResRef of TGA icon
Level_1 ... Level_9	Integer	Index into **spells.2da** specifying extra known spell granted at this level. **** if nothing granted at level
GrantedFeat	Integer	Index into **feat.2da** specifying an additional feat granted by this domain.
CastableFeat	Integer	1 if the *GrantedFeat* is castable 0 if not

Table 5.6.4: categories.2da

Column	Type	Description
Label	String	programmer label describing the category. This 2da is for designer reference. The game has a list of hardcoded category constants that this 2da must conform to.

5.7. Packages

The packages.2da defines the packages that can be chosen by players to automatically recommend feats, skills, spells, and ability increases. It also defines the packages used by the game and toolset to automatically level up non-player creatures.

Table 5.7.1: packages.2da

Column	Type	Description
Label	String	programmer label
Name	Integer	StrRef of the package name
Description	Integer	Description of the package
ClassID	Integer	Index into **classes.2da** specifying what class this package is for.
Attribute	String	Primary ability for this package, the one that raises during levelup. Allowed values: STR, DEX, CON, INT, WIS, CHA
Gold	Integer	Starting gold for a player character created with this package.
School	Integer	Index into **spellschools.2da**, or **** if no package has no specialist spell school.
Domain1 Domain2	Integer	Index into **domains.2da**, or **** if package has no domains.
Associate	Integer	Specifies default starting associate. If ClassID references wizard or sorcerer in classes.2da, then this is an Index into **hen_familiar.2da**. If ClassID references druid in classes.2da, then this is an index into **hen_companion.2da**. The game uses a hard-coded index into classes.2da to determine if the class is wizard, sorcerer, or druid.
SpellPref2DA	String	ResRef of spell preference 2da
FeatPref2DA	String	ResRef of feat preference 2da
SkillPref2DA	String	ResRef of skill preference 2da
Equip2DA	String	ResRef of starting equipment 2da
SoundSet	Integer	Index into soundset.2da for default soundset. Unused,

		always 0.

Table 5.7.2: Package Equipment: packeq*.2da

Column	Type	Description
Label	String	ResRef of UTI item blueprint specifying item to include in inventory of creature

Table 5.7.3: Package Feat Preference Table: packft*.2da

Column	Type	Description
FeatIndex	Integer	Index into **feat.2da**.
Label	String	programmer label

Table 5.7.4: Package Skill Preference Table: packsk*.2da

Column	Type	Description
SkillIndex	Integer	Index into **skills.2da**.
Label	String	programmer label

Table 5.7.5: Package Spell Preference Table: packsp*.2da

Column	Type	Description
SpellIndex	Integer	Index into **spells.2da**.
Label	String	programmer label

5.8. Challenge Rating

Table 5.8: fractionalcr.2da

Column	Type	Description
Label	String	programmer label
DisplayStrRef	Integer	StrRef of the fractional CR
Denominator	Integer	Denominator of the fractional CR, assuming a numerator of 1.
Min	Float	Minimum calculated CR required to have the final CR rounded off to the fractional value implied by the value in the *Denominator* column.

5.9. Other

Table 5.9.1: portraits.2da

Column	Type	Description
BaseResRef	String	"Base" ResRef of TGA texture file to use as the portrait. The actual ResRef used depends on the portrait size to display. To get the actual ResRef, prepend "po_" to the BaseResRef, and append one of the following letters: h = huge (256x512 pixels), size used in character creation portrait selection l = large (128x256), appears in Character Record sheet in game. m = medium (64x128), appears in centre of radial menu, in conversation window, examine window, and as player portrait in upper right corner.

		s = small (32x64), appears as party member portraits along right-hand side of game GUI. t = tiny (16x32) appears in chat window, and in text bubbles if text bubble mode is set to "Full" in Game Options|FeedBack Options.
Sex	Integer	Index into **gender.2da**
Race	Integer	Index into **racialtypes.2da**, or **** for door and plaeable object portraits.
InanimateType	Integer	Index into **placeabletypes.2da**, or **** for creature portraits
Plot	Integer	0 for normal portraits. 1 if portrait is for a plot character. Shows up when the "Plot Characters" radio button is selected in the toolset's Select Portrait dialog. Plot portraits do not show up for selection in the game during character creation.
LowGore	String	Alternate version of BaseResRef to use if the game violence settings are low.

Table 5.9.2: gender.2da

Column	Type	Description
NAME	Integer	StrRef of the gender.
GENDER	String	single capital letter abbreviation
GRAPHIC	String	Not used
CONSTANT	String	Identifier to use in scripting to refer to the gender. Used in toolset Script Wizard to autogenerate source code for a script.

Table 5.9.3: ranges.2da

Column	Type	Description
Label	String	label
PrimaryRange	Float	Max spot range for creatures, or Range value for spells and weapons.
SecondaryRange	Float	Max listen range for creatures, or **** for spell and weapon ranges.
Name	Integer	StrRef of the range name if this is a creature perception range. **** for spell and weapon ranges.