BCS manual

Introduction

In this guide I will try to convey everything we currently know about .bcs files.
The files can be serialized (brought into a human-readable/-editable format) by dragging and dropping them over genser.exe or Lazybones XV2 Xml Serializer. Personally I'd recommend using LBs whenever possible. Unfortunately costcreat still uses genser serialisation.
I will provide the value types with each element, with the following abbreviations:
int - Integer, any natural number (0,1,2,etc.)
hexint - Integer in hexadecimal notation (0x0,0x1,0x2,etc.)
float - Floating point number, fractions in the format 0.1
str - String, pretty much any sequence of characters
bool - True/False statement
BitArray - Pretty much a list of bits (which can be 0 or 1); both instances of this are presented as lists of bool, so this is just a technical info, which does not really affect most people

0. File info

0.1 Gender/Race (U_2C/Race&Female)

General Description:

Determines the characters gender and race. This is mostly irrelevant to be perfectly honest. It determines which character can use which transformations in quests and prior to 1.15, when the devs went in and changed loads of these the vanilla chars had these jumbled around a lot. Like Zarbon being identified as a SYM in BCS.

Genser:

```
<U_2C value="0x104" />
```

Value type: hexint

Gender: 00=Male, 01=Female.
Race: 00=Human, 01=Saiyan, 02=Namekian, 03=Frieza race, 04=Majin and 05=Other.
Genser removes all leading zeros from this, so a male Saiyan would be "0x1" instead of "0x0001".

XV2 Xml Serializer:

```
<Race value="00" />
<Female value="True" />
```

Value type: int & bool

Race: 00=Human, 01=Saiyan, 02=Namekian, 03=Frieza race, 04=Majin and 05=Other.
Gender is determined by setting Female to either True or False.

1. Partsets

```
<PartSet idx="78">
    <Part idx="0">
        <!--Face_base-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="1">
        <!--Face_forehead-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="2">
        <!--Face_eye-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="3">
        <!--Face_nose-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="4">
        <!--Face_ear-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="5">
        <!--Hair-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="6">
        <!--Bust-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="7">
        <!--Pants-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="8">
        <!--Rist-->
        <!--This entry is empty.-->
    </Part>
    <Part idx="9">
        <!--Boots-->
        <!--This entry is empty.-->
    </Part>
</PartSet>
```

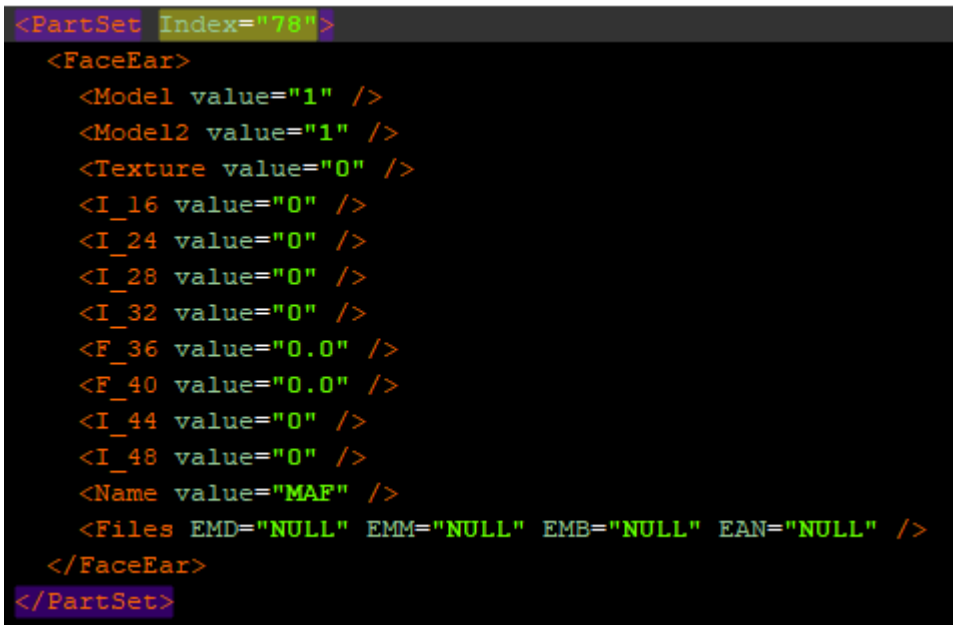
General Description:

A partset contains one “costume slot”. While you can have one partset for the chest, one for the legs and one for the shoes for example, try to contain as many of your models in as few partsets as possible. Especially if you plan on using costcreat, as that limits the amount of total partsets a BCS can have.

Genser:

In genser the Partset **must** always contain all 10 parts, even if not all of them are used. To the left you see the most minimal Partset possible, with blank entries on all Parts.

XV2 Xml Serializer:



```
<PartSet Index="78">
  <FaceEar>
    <Model value="1" />
    <Model2 value="1" />
    <Texture value="0" />
    <I_16 value="0" />
    <I_24 value="0" />
    <I_28 value="0" />
    <I_32 value="0" />
    <F_36 value="0.0" />
    <F_40 value="0.0" />
    <I_44 value="0" />
    <I_48 value="0" />
    <Name value="MAF" />
    <Files EMD="NULL" EMM="NULL" EMB="NULL" EAN="NULL" />
  </FaceEar>
</PartSet>
```

XV2 Xml Serializer ignores all non-existing parts, so you can too.

1.1 Parts

General Description:

A collection of up to 10, well, parts, that may contain one model each.

Genser:

```
<Part idx="0">
    <!--Face_base-->
```

Value type: int

The idx (index) value of any given Part specifies the partname section of a file XXX_YYY_partname.
They must be unique within a Partset (meaning you can not have two Parts with, e.g., an index of 1 within the same partset) and must have one of the following values:

```
"0" - Face_base
"1" - Face_forehead
"2" - Face_eye
"3" - Face_nose
"4" - Face_ear
"5" - Hair
"6" - Bust
"7" - Pants
"8" - Rist
"9" - Boots
```

XV2 Xml Serializer:

`<FaceEar>` Value type: N/A

XV2 Xml Serializer has each part as its own element, with those being:

```
<FaceBase />
<FaceForehead />
<FaceEye />
<FaceNose />
<FaceEar />
<Hair />
<Bust />
<Pants />
<Rist />
<Boots />
```

As stated above, you just need to include the parts you actually use and leave out any you don't. Each element must be unique, so you can't have, for example, two `FaceForehead` elements in the same partset.

1.1.1 MODEL/Model

General Description:

The MODEL element is pretty straightforward. It represents YYY part of a file XXX_YYY_partname, so, for example, `<MODEL value="1" />` in a hair part (idx=”5”) will load XXX_001_Hair.emd, XXX_001_Hair.emb, XXX_001_Hair.dyt.emb and XXX_001_Hair.emm.

Genser:

`<MODEL value="YYY" />` Value type: int

If you do not want to load a model with this (for example, if you just want to attach an accessory) you can have this as 65535.

XV2 Xml Serializer:

`<Model value="YYY" />` Value type: int

If you do not want to load a model with this (for example, if you just want to attach an accessory) you can have this as -1.

1.1.2 MODEL2/Model2

General Description:

The MODEL2 element, as far as I know, does nothing on its own, but combined with a certain U_18/I_24 value will load a different .dyt.emb from the one specified in the MODEL element.

Genser/XV2 Xml Serializer:

`<MODEL2 value="YYY" />` Value type: int

`<Model2 value="YYY" />`

Standard practice is to have this the same as MODEL, unless used with `<U_18 value=”0x2”>/<I_24 value="2" />`.

1.1.3 TEXTURE/Texture

General Description:

The TEXTURE element represents the index of what texture within .dyt.emb is to be used. As an example, all .dyt.emb files used by humans in the game contain 3 textures, DATA000.dds, DATA001.dds and DATA002.dds, so <TEXTURE value="2" /> will load DATA002.dds.
As a side note, for human/saiyan characters DATA002 holds the SSJ textures.

Genser/XV2 Xml Serializer:

```
<TEXTURE value="0" />
```
Value type: int

```
<Texture value="0" />
```

1.1.4 U_10/Shader

General Description:

The U_10 element will modify the shader the game will use on the part. Note that the use for this is mostly unknown.

Genser/XV2 Xml Serializer:

```
<U_10 value="0x0" />
```
Value type: hexint

```
<I_16 value="0" />
```
Value Type: int

Known values for CaC races are:

```
"0x0" - "0" - No change
"0x1" - "1" - bright
"0x2" - "2" - white with with black outlines
"0x3" - "3" - shiny
"0x4" - "4" - bright white again
"0x5" - "5" - dark (like always in shadow)
"0x6" - "6" - dark and mostly grey. hair seems to kinda keep its color
"0x7" - "7" - pretty much the same as 6
"0x8" - "8" - black and weird solid greens?
"0x9" - "9" - same as 8
```

Any other values will crash the game.

1.1.5 U_18/I_24

General Description:

The U_18/I_24 element can modify the way .dyt.embs will be loaded by the game and certain values will apply a tint to the 1st person scouter overlay.

Genser/XV2 Xml Serializer:

```
<U_18 value="0x0" />
```
Value type: hexint

```
<I_24 value="0" />
```
Value type: int

```
Binary          Dec Hex    Function
0000 0000 0000  0   0x0    Standard
0000 0000 0001  1   0x1    Load dyt from parts FILES EMB value
0000 0000 0010  2   0x2    Load dyt from MODEL2
0000 0000 0100  4   0x4    Load dyt ramps from normal .emb
0000 0000 1000  8   0x8    Green Scouter Overlay
0000 0001 0000  16  0x10   Red Scouter Overlay
0000 0010 0000  32  0x20   Blue Scouter Overlay
0000 0100 0000  64  0x40   Purple Scouter Overlay
0000 1000 0000  128 0x80
0001 0000 0000  256 0x100
0010 0000 0000  512 0x200  Orange Scouter Overlay
```

Note that these can be combined. For example:

```
Binary          Dec Hex    Function
0000 0000 0100  4   0x4    Load dyt ramps from normal .emb
+
0000 0000 1000  8   0x8    Green Scouter Overlay
=
0000 0000 1100  12  0xC    Load dyt ramps from normal .emb and add Green Overlay
```

Overlays can not be combined.
Accessories that are loaded as PhysicsObjects (more on that later) might not show up, unless given a U_18/I_24 other than 0x4/4 or combinations thereof.

1.1.6 U_1C/HideParts

General Description:

Setting this allows you to hide specific parts of the body, to avoid clipping or save on polycount. If, for example, you wanted to make Friezas weird floating chair as pants, there would be no need to show the boots of the character. In fact it would look rather goofy, so you could hide the boot part with this element.

Genser:

```
<U_1C value="0x0" />
```
Value type: hexint

The following is an overview on how it works and how to find out which values to use:

Binary	Dec	Hex	Function
0000 0000 0000	0	0x0	Hide nothing
0000 0000 0001	1	0x1	Hide Face_base
0000 0000 0010	2	0x2	Hide Face_Forehead
0000 0000 0100	4	0x4	Hide Face_eye
0000 0000 1000	8	0x8	Hide Face_Nose
0000 0001 0000	16	0x10	Hide Face_ear
0000 0010 0000	32	0x20	Hide Hair
0000 0100 0000	64	0x40	Hide Bust
0000 1000 0000	128	0x80	Hide Pants
0001 0000 0000	256	0x100	Hide Rist
0010 0000 0000	512	0x200	Hide Boots

These can also be combined. For example:

Binary	Dec	Hex	Function
0000 0000 0100	4	0x4	Hide Face_eye
+			
0000 0000 1000	8	0x8	Hide Face_Nose
=			
0000 0000 1100	12	0xC	Hide Face_eye and Face_Nose

XV2 Xml Serializer:

```
<HideParts FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
```
Value type: BitArray

This is a BitArray, but presented as a list of bools. Just set the value(s) of the part(s) you want to hide as True.

1.1.7 U_20/HideMats

General Description:

Setting this allows you to hide certain materials within a mesh to avoid clipping. Examples of why you wanted to do this include hiding the ear a scouter sits on or hiding a bulky bracer or wristband on long sleeved shirts.
The material that is to be hidden needs to be called “something_hide_” or “something_hide_from_part”, where something is the actual name of the material and part is the part you are setting this element up on. So if you wanted to hide a bracer when equipping a long sleeve, the bracer would need to have a material called “bracer_hide_” or “bracer_hide_from_bust” and the long sleeve would need this element set to hide rist materials.

Genser:

```
<U_20 value="0x0" />
```
Value type: hexint

The following is incomplete and subject to change:

Binary	Dec	Hex	Function
0000 0000 0000	0	0x0	Hide nothing
0000 0000 0001	1	0x1	Hide Face_base materials marked hide
0000 0000 0010	2	0x2	Hide Face_Forehead materials marked hide
0000 0000 0100	4	0x4	Hide Face_eye materials marked hide
0000 0000 1000	8	0x8	Hide Face_Nose materials marked hide
0000 0001 0000	16	0x10	Hide Face_ear materials marked hide
0000 0010 0000	32	0x20	Hide Hair materials marked hide
0000 0100 0000	64	0x40	Hide Bust materials marked hide
0000 1000 0000	128	0x80	Hide Pants materials marked hide
0001 0000 0000	256	0x100	Hide Rist materials marked hide
0010 0000 0000	512	0x200	Hide Boots materials marked hide

Of course these can be combined, but you get the gist of that by now…

XV2 Xml Serializer:

```
<HideMats FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
```
Value type: BitArray

Again, this is a BitArray, but presented as a list of bools. Just set the value(s) of the part(s) you want to hide materials on as True.

1.1.8 Unknown elements

General Description:

Unfortunately, it is currently unknown what these 4 elements do, or, really, **if** they do anything at all.

Genser:

```
<F_24 value="0.0" />
<F_28 value="0.0" />
<U_2C value="0x0" />
<U_30 value="0x0" />
```

Value type: float

Value type: hexint

XV2 Xml Serializer:

```
<F_36 value="0.0" />
<F_40 value="0.0" />
<I_44 value="0" />
<I_48 value="0" />
```

Value type: float

Value type: int

1.1.9 NAME/Name

General Description:

The NAME element represents XXX part of a file XXX_YYY_partname, so, for example, `<NAME value="HUM" />` in a hair part (idx="5") with a MODEL value of 1 will load HUM_001_Hair.emd, HUM_001_Hair.emb, HUM_001_Hair.dyt.emb and HUM_001_Hair.emm.

Genser/XV2 Xml Serializer:

```
<NAME value="XXX" />
```

Value type: str

```
<Name value="XXX" />
```

1.1.10 FILES/Files

General Description:

The FILES element can be used to load files that do not adhere to the standard XXX_YYY_partname naming scheme XV2 uses, files that are part of another partset and even files belonging to other characters altogether.

Genser:

```
<!--MODEL, EMM, EMB, EAN-->
<FILES value="PATH_TO_EMD, PATH_TO_EMM, PATH_TO_EMB, PATH_TO_EAN" />
```

Value Type: str

If you wanted to load a file named Foobar.emd, you would use the following FILES values:
`<FILES value="Foobar, NULL, NULL, NULL" />`
Because of the “NULL” values, this will still load the .emm, .emb and .ean (if applicable) determined by NAME, MODEL and Part index.
Alternatively, the function the vanilla game uses this element mostly for is loading shared .emb files, so
`<FILES value="NULL, NULL, HUM_000_Hair, NULL" />`
will load the .emd, .emm and .ean (if applicable) that are determined by NAME, MODEL and Part index, but HUM_000_Hair.emb.
The last use this has is loading other characters files. The values are paths relative to the directory the .bcs is located in, so in the examples above, the files need to be in the same directory as the .bcs, but you can load files from different directories by using relative path syntax. For example:
`<FILES value="../HUF/HUF_000_Hair, ../HUF/HUF_000_Hair, ../HUF/HUF_000_Hair, NULL" />`
The “..” represents the parent directory of the current directory (so, if the .bcs is in \data\chara\HUM, “..” is \data\chara). If you wanted to load from a subdirectory of the current directory (let’s say, \data\chara\HUM\belt) you would need the following values
`<FILES value="./belt/someemd, ./belt/someemm, ./belt/someemb, NULL" />`
with the single colon representing the current directory.

XV2 Xml Serializer:

```
<Files EMD="PATH_TO_EMD" EMM="PATH_TO_EMM" EMB="PATH_TO_EMB" EAN="PATH_TO_EAN" />
```

Value type: (array of) str

If you wanted to load a file named Foobar.emd, you would use the following FILES values:
`<Files EMD="Foobar" EMM="" EMB"" EAN="" />`
Because of the “” values, this will still load the .emm, .emb and .ean (if applicable) determined by NAME, MODEL and Part index.
Alternatively, the function the vanilla game uses this element mostly for is loading shared .emb files, so
`<Files EMD="" EMM="" EMB"HUM_000_Hair" EAN="" />`

will load the .emd, .emm and .ean (if applicable) that are determined by NAME, MODEL and Part index, but HUM_000_Hair.emb.
The last use this has is loading other characters files. The values are paths relative to the directory the .bcs is located in, so in the examples above, the files need to be in the same directory as the .bcs, but you can load files from different directories by using relative path syntax. For example:

```
<Files EMD="../HUF/HUF_000_Hair" EMM="../HUF/HUF_000_Hair" EMB"../HUF/HUF_000_Hair" EAN="" />
```

The ".." represents the parent directory of the current directory (so, if the .bcs is in \data\chara\HUM, ".." is \data\chara). If you wanted to load from a subdirectory of the current directory (let's say, \data\chara\HUM\belt) you would need the following values

```
<Files EMD="./belt/someemd" EMM="./belt/someemm" EMB"./belt/someemb" EAN="" />
```

with the single colon representing the current directory.

1.2 ColorSelector

General Description:

These let you force color colorable parts.
The PART_COLOR element determines what PartColors collection will be used, while the COLOR element determines the specific color within that collection (more on both later).

Genser/XV2 Xml Serializer:

```
<ColorSelector>
    <!--eye_-->
    <PART_COLORS value="5" />
    <!--Color preview: #1C1C1C-->
    <COLOR value="1" />
</ColorSelector>
```

Value types: int

```
<ColorSelector>
  <PartColors value="5" />
  <Color value="1" />
</ColorSelector>
```

1.3 PhysicsObjects

```
<PhysicsObject>
    <U_00 value="0xffff" />
    <U_02 value="0xffff" />
    <U_04 value="0x2" />
    <U_18 value="0x1" />
    <U_1C value="0x0" />
    <U_20 value="0x0" />
    <NAME value="XXX" />
    <STR_28 value="PATH_TO_EMD, PATH_TO_EMM, PATH_TO_EMB, PATH_TO_EAN, BONE_TO_ATTACH_TO, PATH_TO_SCD" />
</PhysicsObject>
```

```
<PhysicsObject>
  <I_00 value="-1" />
  <I_02 value="-1" />
  <Texture value="0" />
  <I_24 value="0x1" />
  <HideParts FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
  <HideMats FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
  <Name value="XXX" />
  <BoneToAttach value="b_C_Base" />
  <Files EMD="PATH_TO_EMD" EMM="PATH_TO_EMM" EMB="PATH_TO_EMB" EAN="PATH_TO_EAN" SCD="PATH_TO_SCD" />
</PhysicsObject>
```

These represent meshes that can be added to partsets. The vanilla game uses them mostly for physics enabled objects or accessories. As far as I know, you can add an unlimited number of PhysicsObjects to any one part. Most of the elements here are identical to their equivalents in a part, so I'll just point to that, unless there are differences.

1.3.1 U_00/I_00

General Description:

Exactly the same as MODEL within parts.

Genser:

`<U_00 value="0xffff" />` Value type: hexint

The element is often 0xFFFF:, which as in MODEL tells the game to load no model.

XV2 Xml Serializer:

`<I_00 value="-1" />` Value type: int

The element is often -1, which as in MODEL tells the game to load no model.

1.3.2 U_02/I_02

General Description:

Exactly the same as MODEL2 within parts.

Genser:

`<U_02 value="0xffff" />` Value type: hexint

XV2 Xml Serializer:

```
<I_02 value="-1" />
```
Value type: int

1.3.3 U_04/Texture

General Description:

Again, exactly the same as TEXTURE within parts.

Genser:

```
<U_04 value="0x2" />
```
Value type: hexint

XV2 Xml Serializer:

```
<Texture value="0" />
```
Value type: int

1.3.4 U_18/I_24

General Description:

Exactly the same as U_18/I_24 in parts.

Genser:

```
<U_18 value="0x1" />
```
Value type: hexint

XV2 Xml Serializer:

```
<I_24 value="1" />
```
Value type: int

1.3.5 U_1C/HideParts

General Description:

Exactly the same as U_1C/HideParts in parts.

Genser:

```
<U_1C value="0x0" />
```
Value type: hexint

XV2 Xml Serializer:

```
<HideParts FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
```
Value type: BitArray

1.3.6 U_20/HideMats

General Description:

Exactly the same as U_20 in parts.

Genser:

```
<U_20 value="0x0" />
```
Value type: hexint

XV2 Xml Serializer:

```
<HideMats FaceBase="False" Forehead="False" Eye="False" Nose="False" Ear="False" Hair="False" Bust="False" Pants="False" Rist="False" Boots="False" />
```
Value type: BitArray

1.3.7 NAME/Name

General Description:

You probably guessed it, but it's the same as NAME in parts.

Genser/XV2 Xml Serializer:

```
<NAME value="XXX" />
```
Value type: str

```
<Name value="XXX" />
```

1.3.8.1 BoneToAttach & Files

General Description:

BoneToAttach determines the bone from the character skeleton that acts as the base bone of the PhysicsObjects skeleton.
Files works the same way as FILES, but with an added string that has the path to the scd, which is a file that holds physics information for XV2 models.

XV2 Xml Serializer only:

```
<BoneToAttach value="b_to_Attach" />
<Files EMD="PATH_TO_EMD" EMM="PATH_TO_EMM" EMB="PATH_TO_EMB" EAN="PATH_TO_EAN" SCD="PATH_TO_SCD" />
```
Value type: str

1.3.8.2 STR_28

General Description:

This is mostly identical to FILES in parts and thus the same rules and functions apply, but it has two additional values. PATH_TO_SCD behaves exactly as the other values, but

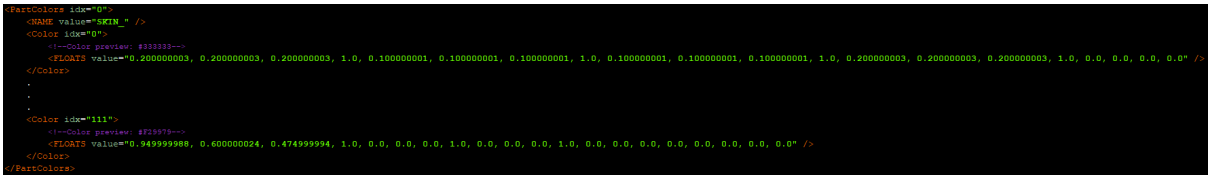
BONE_TO_ATTACH_TO is the base bone for the entire PhysicsObject and can only have values that are bones present in the skeleton of the character the .bcs belongs to.

Genser only:

```
<STR_28 value="PATH_TO_EMD, PATH_TO_EMM, PATH_TO_EMB, PATH_TO_EAN, BONE_TO_ATTACH_TO, PATH_TO_SCD" />
```

Value type: str

2. PartColors



```
<PartColor Index="0">
  <Name value="SKIN_" />
  <Colors Index="0">
    <Color1 R="0.2" G="0.2" B="0.2" A="1.0" />
    <Color2 R="0.1" G="0.1" B="0.1" A="1.0" />
    <Color3 R="0.1" G="0.1" B="0.1" A="1.0" />
    <Color4 R="0.2" G="0.2" B="0.2" A="1.0" />
    <Color5 R="0.0" G="0.0" B="0.0" A="0.0" />
  </Colors>
  .
  .
  .
  <Colors Index="111">
    <Color1 R="0.95" G="0.6" B="0.475" A="1.0" />
    <Color2 R="0.0" G="0.0" B="0.0" A="1.0" />
    <Color3 R="0.0" G="0.0" B="0.0" A="1.0" />
    <Color4 R="0.0" G="0.0" B="0.0" A="0.0" />
    <Color5 R="0.0" G="0.0" B="0.0" A="0.0" />
  </Colors>
</PartColor>
```

Each PartColors holds a set of colors to be used by both the ColorSelector part element and the ingame coloration.
The elements index value determines which Materials the colors within the element will be applicable to and are used by the ColorSelectors PART_COLOR element.
The NAME element within, holds the material prefix (so index “0”, which has a NAME value of “SKIN_” will be applicable to all materials that have a SKIN_ prefix in .emm, such as SKIN_Bust).
Each Color element represents a single color (duh) that can be selected in game and the index is used by the ColorSelectors COLOR element.

```
<FLOATS value="RED1, GREEN1, BLUE1, ALPHA1, RED2, GREEN2, BLUE2, ALPHA2, RED3, GREEN3, BLUE3, ALPHA3, RED4, GREEN4, BLUE4, ALPHA4, 0.0, 0.0, 0.0, 0.0" />
```

The FLOATS element holds multiple RGBA colors in float format and is always followed by a set of 4 “0.0” float values.
The XV2 Xml Serializer element should be self explanatory.

3. Body

`<Body idx="0">` Value type: int

For CaC races each Body element represents one combination of the height and body type sliders in character creation, hence vanilla CaC races will have 12 (0-11) body elements (except NMC, who have one additional element for “Turn Giant”).
In non-CaCs it is mostly used, if a transformation bulks the character up or slims them down.

Within each Body element there is at least one BoneScale element. Every bone not addressed by a BoneScale element will stay at default values.
Each BoneScale element has the two following sub elements:

3.1 Scale

`<SCALE value="X, Y, Z" />` Value type: float

These are X, Y and Z modifiers for the scale of the Bone.
Default is “1.0”, so any numbers greater than that would grow the bone and numbers lower than that would shrink the bone.

3.2 NAME

`<NAME value="b_C_Base" />` Value type: str

This is the name of the bone that is being manipulated. If you want to know the Bone names of a particular character, take a look at their .esk in Xenoviewer, export the .esk to fbx and look at it in Blender/3dsMax or serialize the .esk with XenoXMLConverter and look at the xml.
Note that most bones are nested. This means, if you for example shrunk b_R_Arm2, b_R_Hand will be shrunk too as it is nested under b_R_Arm2, as well as all finger bones as they in turn are nested under b_R_Hand.

In XV2 Xml Serializer these are folded into each other, but it’s basically the same.

4. SkeletonData

```
<SkeletonData idx="0">
    <U_00 value="0x0" />
    <Bone name="b_C_Chest">
        <U_00 value="0x12" />
        <U_04 value="0x0" />
        <U_08 value="0x0" />
        <!--Following values are half-floats. They will be changed into floats values in a future revision-->
        <U_0C value="0x0, 0x0, 0x999a, 0x3e19, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xcccd, 0x3ecc, 0x999a, 0x3e99, 0x7ae1, 0x3e94" />
    </Bone>
```

These are for fine tuning specific scd skeletons. I'd suggest using LBs Serializer instead of genser here as it makes it much easier to read. Attention: All of these findings are speculations at this point.

```
<Bone Name="b_C_Chest">
  <I_00 value="18" />
  <I_04 value="0" />
  <F_12 value="0.0" />
  <F_16 value="0.12" />
  <F_20 value="0.0" />
  <F_24 value="0.0" />
  <F_28 value="0.0" />
  <F_32 value="0.0" />
  <F_36 value="0.3" />
  <F_40 value="0.35" />
  <F_44 value="0.3" />
</Bone>
```

4.1 Bone

`<Bone Name="b_C_Chest">` Value type: string

Pretty self explanatory. Bone name within the esk/scd.

4.2 I_00/U_00

`<I_00 value="18" />` Value type: int (hexint in genser)

Unknown, but in most instances I've seen so far, it's 18 (0x12). Other values cause your character to be pushed in various directions.

4.2 I_04/U_04

`<I_04 value="0" />` Value type: int (hexint in genser)

Bone thickness. Accepts values from 0-3 and affects how thick other scd bones consider the bone to be for interaction. Example:

b_R_Leg1 has a I_04 value of 3, while b_L_Leg1 has a I_04 value of 1. The right “skirt” part is raised higher as the right leg is considered thicker.

4.3 F_12-20

```
<F_12 value="0.0" />
<F_16 value="-0.25" />
<F_20 value="0.0" />
```

Value type: float

These seem to influence the inertia of scd bones affected by the bone on the X-, Y- and Z_axis respectively. Positive values make the **affected** bone behave as if they were heavier, while negative values make them behave as if lighter.