Poser : Unofficial CR2 File Specification

Forward

The Unofficial CR2 File Specification has been a little neglected over the past couple years. Minor modifications have been made even as recently as September 2004. But much has changed since its inception, including more information about Poser 5 additions and some of those already existing mysterious keywords.

This document contains the same information on the website in a PDF format so that you have a handy, printable reference.

Thank you,

Robert Templeton

Credits:

- Kevin Rose for his wonderful, albeit defunct, CR2 Guide.

- Kattman for the Poser CR2 Autopsy.

- Lemurtek and Bloodsong for their hard work at creating tables of the hidden associations of the numbers within GetStringRes().

- The Poser 4 and Poser 5 Reference manuals.

- "*Secrets of Figure Creation with Poser 5*" by B L Render

- Anthony Appleyard for suggesting the creation of a downloadable document.

Contents

Introduction

Poser uses a set of files with unique extensions to augment basic geometries (Wavefront .obj files) for enhanced use within it. There are four basic classes:

Class	File Extension	Compressed Extension	Poser Type
Scene	pz3	pzz	Scene
Geometry	cr2	crz	Character
	pp2	ppz	Prop
	hr2	hrz	Hair
Pose	hd2	hdz	Hand
	lt2	ltz	Light
	cm2	cmz	Camera
	pz2	p2z	Pose
	fc2	fcz	Face/Expression
Material	mt5	mz5	Poser 5 Material
	nod	---	Poser 5 Shader Node

Compressed files use zlib compression. There is an option in Poser Preferences to enable/disable file saves using compression. Also, there is a Python script which will decompress compressed files.

Of special interest is the character file (.**cr2**) because the geometry mesh, bone rigging (skeleton), morphs, and materials are all declared and defined within it for a posable, morphable, texturable character. Also, the Pose class files are subsets of it and the Scene type.

Unlike other CR2 file descriptions that can be found online, this one will not be examining a particular file due to the fact that there are a myriad more variations then are contained in any one file. Here, the basic structure, sections, section formats, and details will be explained as best as can be done by examination and with the help of available resources.

Corrections, omissions, and more detailed descriptions are welcomed and can be noted in the Unofficial CR2 File Specification forum.

Notation:

- <> = field
- [] = optional field
- Poser 5

Basic File Structure

Line Endings: Since Poser is a cross-platform application that runs on both MacOS and Windows, one may notice that some Poser files appear with "junk" when viewed in a Mac text editor or that they appear with "junk" and as one long line in a Windows text editor. This is due to the fact that each OS uses a different end of line (EOL, henceforth) character set. MacOS uses a single character, a Carriage Return (CR = hex: 0D), to denote EOLs. Windows (and MSDOS for that matter) uses two characters, a Carriage Return followed by a Linefeed (CRLF = hex: 0D0A). Note that both systems use a CR, while Windows adds a now useless, but still necessary, character (not used much anymore, anyway). There are two situations where this difference is relevant.

Editing a Poser file where the text editor does not support both EOL types, making sense of the file contents and editing may be difficult. My suggestion in this case is to either use a text editor that will load and display the file correctly as well as possibly save it in your system's format or use a little program that converts the file contents from one EOL type to the other.

Parsing a Poser file from a program or plugin, perhaps to extract information or to modify the contents, does require some care. In most cases, the library or API file reading calls will automatically handle both types of EOL. When they don't, your best avenue is to seek information concerning this either from the API reference, online communities, or directly with the maker of the API. If you are using your own file line reader (as I am doing), the algorithm that handles both EOL types is as follows (pseudo C++):

```
Bool FileReader::IsSpace(char c)
{
    // Tab, Space, CR, LF
    return ((c == 0x09) || (c == 0x20) || (c == 0x0D) || (c == 0x0A));
}


// Determine EOL condition
Bool FileReader::IsEOL(char c)
{
    return ((c == 0x0D) || (c == 0x0A));
}


// Determine EOF condition
Bool FileReader::IsEOF()
{
    return (GetPosition() == (GetFileLength()-1));
}


// Read a line from file (up to EOL or EOF),
// skipping leading whitespace, even blank lines
char* FileReader::ReadLine()
{
    char buffer[1024]; // this is normally allocated elsewhere
    char *inbuf = &buffer[0];
    // while there is whitespace
    do {
        if (IsEOF()) return NULL;
        if (!ReadChar(inbuf)) throw ErrorException(ERROR_NOREADFILE);
    } while (IsSpace(*inbuf));


    // Append characters until EOL (or EOF)
    for (inbuf++; !IsEOF(); inbuf++)
    {
        if (!ReadChar(inbuf)) throw ErrorException(ERROR_NOREADFILE);
        if (IsEOL(*inbuf))
        {
            // Null-terminate buffer
            *inbuf = 0;
            return buffer;
        }
    }
    return NULL;
}
```

Blank lines are allowed as well as any type of whitespace (spaces and tabs) surrounding fields. These are all removed during parsing (see above).

Comments can be added in one of two ways. You can add a text line at the end of a section (before the closing brace) or by using C/C++ style commenting (// text):

{
number 4.01
This is a comment
}

{
// This is also a comment
// or two

}

If using the first method, which I have yet to encounter, one would imagine that the first word should not match any of the available parameter/command/section names expected for the section in which the comment is included.

Case Sensitivity: Poser files are mostly case sensitive. Field name case cannot vary from one reference to another and file specifiers should match the case of the system path and file names.

Fields are the atomic units of information within the file, which can be as simple as a single parameter data and as complex as an entire section. Within this filespec, variable or general fields are denoted and surrounded by <>. If the field or a part thereof is optional, it is enclosed by [].

When only several *mutually-exclusive* specific field options are available, they will be delimited by '|' to show an OR relationship between options:

<1|0>

For clarification of data types and interpretations, where possible, a variable type and identification will be used, for example:

<real:x>

which denotes a floating point value that represents an x coordinate. This will mainly be limited to numeric values, others being text.

As a convention, specific to Poser files, to simplify structure formats, the general field *<part>* will denote the *<part name>:<character number>* pair (i.e.: *BODY:2*) used throughout when referencing declared actors and props.

Parameters are single-line sets of "parameter data" fields within a section, each field separated by whitespace (spaces or tabs), data fields being optional in some cases:

parameter <[data1 data2 ... dataN]>

There are two exceptions to this format in a CR2 file, as far as can be ascertained, and these are the *sphereMatsRaw* parameter which defines its data - two 4x4 matrices - on separate lines from the parameter and the *xxxMap* material parameters which add a "0 0" line when a file reference is defined.

One of the general fields for a parameter is a True|False, On|Off value specified by a 1 or 0, respectively. In the course of checking the varied parameters within the CR2 file dissection, you may see something like this:

parameter <1|0>

This denotes that this parameter has one of these values, the first is usually the default or most used one.

Commands are a simple variant of parameters and specify specific fields on the lines that follow:

command [<data>]
<field1.1> [<field1.2> ... <field1.N>]
[...
<fieldN.1> ...]

Note that the number of fields, lines, and fields per line are all determined by the command. Fields on subsequent lines can be data, parameters, or sections.

Sections are collections of other sections, parameters, and commands typified by a "type name" header (except for the file-level section which has none), followed by opening and closing braces, each on their own line, enclosing the collection:

type <[name]>
{
<[other sections, parameters, commands]>
}

Sections are nested hierarchically, which is shown visibly within the file by tabs, though tabs are only used for this purpose and have no effect on syntactical correctness otherwise. Each file is a section in itself and a set of braces is used at the beginning and end of

the file to denote this. Between these file-level braces are all of the content sections, parameters, and commands of the file:

```
{
   type1 <[name]>
      {
      <[other sections, parameters, commands]>
      }
   parameter1
   command1
   type2 <[name]>
      {
      <[other sections, parameters, commands]>
      }
...
   commandN
   typeN <[name]>
      {
      <[other sections, parameters, commands]>
      }
   parameterN
}
```

Although there is uniformity in the ordering of subsections, parameters, and commands within similar sections, there are no general rules on the ordering except in the case of linearly-parsed ordered sets of data or where specified. Otherwise, the ordering is only implemented for visual inspection, convenience, and format homogeneity across sections and files.

CR2 File Structure

There are six types of major sections: *version*, *actor* (declaratory), *prop*|*hairprop* (declaratory), *actor* (definitive), *prop*|*hairprop* (definitive), and *figure.* There are two types of major parameters: *figureResFile* and *setGeomHandlerOffset*. Each of these fields within the CR2 file will be defined and detailed in order over the next pages. Each subsection will be handled separately on subpages so that this reference maintains a correspondence with the format for easy access to the relevant information.

The CR2 file follows the *Basic File Structure* and has these main sections and parameters:

{
Version
  {
  }

[readScript <data>]
... other readscript parameters

// * Declaratory Area *****
[figureResFile <data1>]
actor | prop | hairProp (Declaratory) *<data1>*
  {
  }
... other actor sections
[prop|hairProp<data1>
  {
  }
... other prop sections]
...
[figureResFile <dataN>]
actor <dataN>
  {
  }
... other actor sections
[prop|hairProp <dataN>
  {
  }
... other prop sections]
...
[figureResFile <data1>]

// * Definitive Area *****
actor <data1>
  {
  }
... other actor sections
[prop|hairProp <data1>
  {
  }
... other prop sections]
figure (1)
  {
  }
...
[figureResFile <dataN>]

actor <dataN>
  {
  }
... other actor sections
[prop|hairProp <dataN>
  {
  }
... other prop sections]
figure (N)
  {
  }
[figureResFile <data1>]
setGeomHandlerOffset <data>

}

The first thing to note is that more than one figure can be defined within the CR2 file. Poser is keyed to this by the use of *figure numbers* - the number appearing after the *<part name>:* in the *actor* or *prop* name field (i.e.: *actor Hip:1* is the Hip part for figure 1).

The **Declaratory** area declares the actors, props, and related parts, usually stating geometry or geometry groups.

The **Definitive** area defines them with respect to Poser's augmented usage.

Each **Definitive** and **Declaratory** area is started with the appropriate *figureResFile* reference *if and only if* the geometry mesh is stored externally in file and the figure's actors reference geometry groups and not geometry mesh files themselves. If geometry, whether in file or embedded, is defined for each and every *actor* individually, *figureResFile* can be omitted. The **Declaratory** area is terminated (or separated from the **Definitive** area) by a *figureResFile* reference to the initial *figureResFile.* In the case of no *figureResFile* references, determination of *actor* and *prop* sections as declaratory or definitive depends on whether or not the particular *actor* or *prop* has already been encountered (<part name>:<figure number> are the same).

It is vitally important that the **Declaratory** area precede the **Definitive** area for each figure or overall. Otherwise references in the **Definitive** sections to the figure's actors and props will not be set. The number of *actor* sections and *prop* sections in the **Definitive** area must match the number in the **Declaratory** area.

Next, note that the order of these major sections should be maintained, though the *actor* and *prop* sections need not be order-correlated between the **Declaratory** and **Definitive** areas, except for easier visual reference. One can even mix various figure's actors and props when no *figureResFile* references exist as long as the advice in the previous paragraph is heeded.

The function of *setGeomHandlerOffset* may be a means of termination of the file parse as it always appears just once as the last field of the file, no matter how many figures are defined.

In its stripped-down form, the CR2 file has the following minimal format:

```
{
version
   {
   }
// *** Declaratory Area ***
actor <data1>
   {
   <.obj file reference | embedded .obj data>
   }
... other actor sections with similar internal fields

// *** Definitive Area ***
actor <data1>
   {
   }
... other actor sections

figure (1)
   {
   }
setGeomHandlerOffset <data>
}
```

Version

The *version* section defines the version of Poser under which the file was created:

```
version
   {
   number <real:major.minor>
   }
```

The *version* section header takes no name field.

The only content for this section is the *number* parameter. The data for this parameter is a numerical value in the format <major release number>[.<minor release (patch) number>], where <major release number> currently has values from 1 to 5 and <minor

release number> is an optional value such as '0' or '03'. This value can be treated as a floating point (real) number.

readScript

The *readScript* parameter reads and includes the file located at <path> as if it were part of the same Poser file.

readScript <path>

This is what allows Morph INJection and REMoval to work within Poser and has been used extensively on Daz3D's Unimesh figures (Michael3, Victoria3, David, YoungTeens, Aiko, and so on).

figureResFile

The *figureResFile* parameter defines the Wavefront .obj file to be used as the geometry of a figure. There are two possible data fields for this parameter:

figureResFile <file specifier|GetStringRes(1052,<int>)>

This first data field type takes a relative file path specifier in Mac format (i.e.: using ':' instead of '\' for path separation). The general format of this field is:

:Runtime:Geometries:[<paths>]:<file>.obj

where <*paths*> is any allowable number of colon-separated directories that lead to <*file*>.obj and where <*file*> is the name of an existing geometry file.

The second is used by Poser internally to reference the geometry file so that different languages can be handled correctly. The first argument within *GetStringRes()* defines the type of string resource pool from which to get the information while the second is an index into that pool, for whatever language is defined, for a specific string. The number *1052* denotes a string resource pool of .obj file location specifier strings. The argument <*int*> takes any of the following known values resolving to these known file specifiers (Note: *:Runtime:Geometries* has been omitted for clarity):

<int>	**String Resource**
1-8	UNUSED
9	*:props:ball.obj*
10	*:props:box.obj*
11	*:props:cane.obj*
12	*:props:stairs.obj*
13	UNUSED
14	*:props:cone.obj*
15	*:props:cylTex.obj*
16	*:props:square.obj*
17	*:props:torusThin.obj*
18	*:maleOrig:maleOrig.obj*
19	*:maleNudeHi:maleNudeHi.obj*
20	*:maleNudeLo:maleNudeLo.obj*
21	*:maleSuitHi:maleSuitHi.obj*
22	*:maleSuitLo:maleSuitLo.obj*
23	*:maleCasHi:maleCasHi.obj*
24	*:maleCasLo:maleCasLo.obj*
25	*:maleSkeleton:maleSkeleton.obj*
26	*:maleStick:maleStick.obj*
27	*:femaleOrig:femaleOrig.obj*
28	*:femaleNudeHi:femaleNudeHi.obj*
29	*:femaleNudeLo:femaleNudeLo.obj*
30	*:femaleSuitHi:femaleSuitHi.obj*

32	*:femaleCasHi:femaleCasHi.obj*
33	*:femaleCasLo:femaleCasLo.obj*
34	*:femaleSkeleton:femaleSkeleton.obj*
35	*:femaleStick:femaleStick.obj*
36	*:childNudeHi:childNudeHi.obj*
37	*:childNudeLo:childNudeLo.obj*
38	*:childCasHi:childCasHi.obj*
39	*:childCasLo:childCasLo.obj*
40	*:childStick:childStick.obj*
41	UNUSED
42	*:manikin:manikin.obj*
43	UNUSED
44	*:props:ball.obj*

actor | prop | hairProp (Declaratory)

The *actor*, *prop*, and *hairProp* sections declare each part of the figure based on the geometry file's groups (.obj 'g' specifer) or other geometry. In order to avoid repetition, as fields are defined, they will not be redefined in the alternative formats. The general formats are as follows:

1. The *BODY* actor, since it does not reference any geometry, is an empty *actor* section as shown above. You may also find an actor named *neckDummy* in older Poser figures that is also empty. Other situations in which these appears are when clothes, hair, props, etc. are defined as characters using the default Poser figure hierarchy but which reference unused actors.

```
actor <part name>:<character number>
   {
   }

prop [<part name>:<character number>]
   {
   }

hairProp [<part name>:<character number>]
   {
   }
```

Following the *actor*, *prop*, or *hairProp* section type are two colon-separated fields, optionally used for *prop* and *hairProp* when they are parented to a figure. The first specifies a part name and the second a *figure* number that should be the same for every *actor* reference for a figure throughout the file. This number just reflects which figure number it was when saved to the library within Poser. The first actor should always have the *<part>, BODY*, which defines the overall figure parameters.

2. Each group within a referenced .obj file should have a corresponding *actor* section (both declaratory and definitive). *prop* sections do not reference into the *figureResFile* geometry file and are thus limited to one of the two alternative formats below, (**3.**) and (**4.**).

```
actor <part name>:<character number>
   {
   storageOffset <real> <real> <real>
   geomHandlerGeom <int> <groupname>
   }
```

storageOffset defines the (x,y,z) coordinates to move the *actor/prop* geometry from its default location. In reality, this parameter seems to be unused except as a way to differentiate between *actor* and *prop* sections. The usual parameter when *actor* is used is:

```
storageOffset 0 0 0
```

while prop uses this:

```
storageOffset 0 0.3487 0
```

geomHandlerGeom has two fields. The first, again, seems to be a constant number (maybe for MacOS) with a value of *13*. The

second, *<groupname>*, is text that matches the qualifying name after a 'g' in the .obj file referenced by *figureResFile.* So, if there is a group within the .obj file:

g name1 name2 name3

then any matching name can be referenced for this parameter.

3. This format is used mainly for *prop* sections (as is the next format), but can be used for *actor* sections. The only difference between this format and (***2.***) is that the group reference by way of *geomHandlerGeom* has been replaced by a file reference using *objFileGeom.*

objFileGeom has three fields, the first two again being constant numbers suspiciously appearing to support MacOS. The third is a *<file specifier>* in the same format as *figureResFile*'s data field:

:Runtime:Geometries:[<paths>]:<file>.obj

where *<paths>* is any allowable number of colon-separated directories that lead to *<file>*.obj and where *<file>* is the name of an existing geometry file.

```
actor|prop|hairProp <part name>:<character number>
   {
   storageOffset <real> <real> <real>
   objFileGeom <int> <int> <file specifier>
   }
```

4. This format embeds the .obj file geometry into the CR2 file as a *geomCustom* section. Its format is indistinguishable from a .obj file's content except for prefixed count parameters. Click on any of the link for more information.

```
actor|prop|hairProp <part name>:<character number>
   {
   geomCustom
      {
      }
   }
```

5. This format specifies a number to a Poser internal object resource, such as the Ground, light, or camera geometries.

```
prop
   {
   geomResource <int>
   }
```

geomCustom

The *geomCustom* section embeds a Wavefront .obj file geometry into a Poser file as follows:

```
geomCustom
   {
   numbVerts <long>
   numbTVerts <long>
   numbTSets <long>
   numbElems <long>
   numbSets <long>
   <.obj file with v,vt,f,g,usemtl fields>
   }
```

Each of the *numbXXX* parameters enumerates the number of expected fields as follows:

- *numbVerts*: total number of vertices (v)
- *numbTVerts*: total number of texture vertices (vt)
- *numbTSets*: total number of texture sets (total vt's referenced in all f's) (usemtl)
- *numbElems*: total number of facets (f)
- *numbSets*: total number of vertex sets (total v's referenced in all f's) (g)

This is followed by an completely embedded polygonal object definition in standard Wavefront polygonal .obj file format. The details of this format will not be covered here as these details can be found in many places already. For those who do not have this information, here is a PDF reference (Right click and Save Link Target As... if you don't want it to open in a browser window).

actor | prop | hairProp (Declaratory)

The **Definitive** *actor, prop,* and *hairProp* sections define many parameters for each part of the figure, including channels which configure dials, morphs, transformations, and joint parameters. If all of your geometry references reside in external files, the **Definitive** area makes up the bulk of a CR2 file.

The general format is as follows:

```
actor|prop|hairProp <part name>:<character number>
   {
   name <name|GetStringRes(1024,<int>)>
   <on|off>
   bend <1|0>
   dynamicsLock <1|0>
   hidden <1|0>
   addToMenu <1|0>
   castsShadow <1|0>
   includeInDepthCue <1|0>
   parent|smartparent<UNIVERSE|part>
   [inkyParent <part>]
   [nonInkyParent <part>]
   [conformingTarget <part>]
   [alternateGeom <name>
      {
      }
   ... more alternateGeom sections
   defaultGeomName GetStringRes(1037,N)]
   channels
      {
      }
   endPoint <real:x> <real:y> <real:z>
   origin <real:x> <real:y> <real:z>
   orientation <real:alpha> <real:beta> <real:gamma>
   displayOrigin <0|1>
   displayMode <TYPE>
   customMaterial <0|1>
   [material <name>
      {
      }
   ... more material sections
    ]
   locked <0|1>
   backfaceCull <0|1>
   visibleInReflections <1|0>
   visibleInRender <1|0>
   displacementBounds <real>
   shadingRate <real>
   smoothPolys <1|0>
   morphPutty
      {
      inactiveGroup <group name>
      ... more inactiveGroup parameters
      }
   }
```

The *<part>* (*<part name>:<character number>*) must match one and only one of those defined in the **Declaratory** area.

name defines the part name used within Poser. If this is a standard part name, *GetStringRes(1024,N)* will be used. The first argument within *GetStringRes()* defines the type of string resource pool from which to get the information while the second is an index into that pool, for whatever language is defined, for a specific string. The number *1024* denotes a string resource pool of part name strings. The argument *<int>* takes any of the following known values resolving to these known specifiers:

<int>	**String Resource**

2	Head
3	Neck
4	Chest
5	Abdomen
6	Hip
7	Left Thigh
8	Left Shin
9	Left Foot
10	Right Thigh
11	Right Shin
12	Right Foot
13	Left Shoulder
14	Left Forearm
15	Left Hand
16	Right Shoulder
17	Right Forearm
18	Right Hand
19	Left Collar
20	Right Collar
21	Jaw
22	Left Pinky 1
23	Left Pinky 2
24	Left Pinky 3
25	Right Pinky 1
26	Right Pinky 2
27	Right Pinky 3
28	Left Ring 1
29	Left Ring 2
30	Left Ring 3
31	Right Ring 1
32	Right Ring 2
33	Right Ring 3
34	Left Mid 1
35	Left Mid 2
36	Left Mid 3
37	Right Mid 1
38	Right Mid 2
39	Right Mid 3
40	Left Index 1
41	Left Index 2
42	Left Index 3
43	Right Index 1
44	Right Index 2
45	Right Index 3
46	Left Thumb 1
47	Left Thumb 2
48	Left Thumb 3
49	Right Thumb 1
50	Right Thumb 2
51	Right Thumb 3
52	Left Toe
53	Right Toe
54	Neck 1

56	Right Finger 1
57	Right Claw 1
58	Right Finger 2
59	Right Claw 2
60	Right Finger 3
61	Right Claw 3
62	Left Finger 1
63	Left Claw 1
64	Left Finger 2
65	Left Claw 2
66	Left Finger 3
67	Left Claw 3
68	Right Toe 1
69	Right Toe 2
70	Left Toe 1
71	Left Toe 2
72	Tail 1
73	Tail 2
74	Tail 3
75	Tail 4
76	Left Up Arm
77	Right Up Arm
78	Left Wrist
79	Right Wrist
80	Lower Neck
81	Upper Neck
82	Left Leg
83	Right Leg
84	Left Ankle
85	Right Ankle
86	Waist
87	Left Ear 1
88	Right Ear 1
89	Left Ear 2
90	Right Ear 2
91	Left Ear 3
92	Right Ear 3
93	Tail Fins
94	Right Pect Fin
95	Left Pect Fin
96	Body 1
97	Body 2
98	Body 3
99	Body 4
100	Body 5
101	Left Eye
102	Right Eye
103	innerMatSphere
104	outerMatSphere
105	nullMatSphere

<*on/off*> determines whether or not the part is both visible and selectable in the Document window in Poser. It will still be listed in the Hierarchy Editor, Parameters Palette, and Elements menu.

bend determines whether or not to bend part when moved.

dynamicsLock's defines whether or not body part is deformed by joint movement.

hidden determines whether or not part is selectable in Document Window.

addToMenu determines whether or not part is displayed in Body Parts list.

castsShadow determines whether or not part casts shadow during render.

includeInDepthCue determines whether or not part is included in display when Depth Cueing is enabled.

parent specifies the parent <*part*> of this body part. In the case of *BODY* (or the *root* <*part*> as defined in *figure)* it is *UNIVERSE. smartparent* is a way to parent props to the active figure while adding to a scene.

inkyParent specifies an IK chain parent <*part*> and is only necessary if being used as a link or goal in an *inkyChain.*

nonInkyParent specifies the <*part*> that is parent of this part. In most cases, this should be the same as *parent.*

conformingTarget specifies the target body part on the receiving figure for a conforming item (such as clothing).

alternateGeom sections define alternative geometries for the part in question. Click on the links for more information.

defaultGeomName is used only in conjunction with *alternateGeom* and defines the default geometry name in contrast to those defined by the *alternateGeom sections.* If this is a standard part name, *GetStringRes(1037,N)* will be used. The first argument within *GetStringRes()* defines the type of string resource pool from which to get the information while the second is an index into that pool, for whatever language is defined, for a specific string. The number *1037* denotes a string resource pool of geometry name strings. The argument <*int*> takes any of the following known values resolving to these known specifiers:

<int>	**String Resource**
UNKNOWN	UNKNOWN

The *channels* section simply encloses a set of sections used as channels within Poser. Its simplicity ends there. Full information can be found at the *channels* section and links below it in the Contents.

The next three parameters work together to describe the extent of the part with respect to its rigging:

endPoint effects spherical blend zones and specifies the endpoint coordinate of the part in relation to them.

origin is the center of rotation coordinate of the part.

orientation specifies three angles which determine the relative angle of the part with respect to the coordinate axes.

displayOrigin determines whether or not the origin should be displayed (as crosses in Poser).

displayMode sets the Poser mode in which to display the part. The possible settings are:

- USEPARENT
- CARTOONNOLINE
- FLATLINED
- SMOOTHLINED
- SHADEDOUTLINE
- SKETCHSHADED
- FLATSHADED
- TEXTURESHADED
- SHADED
- HIDLINE
- SILHOUETTE
- EDGESONLY

- WIREFRAME

customMaterial determines whether there are custom materials associated with this part. If it is non-zero, then expect *material* sections to follow.

material sections are only included here if *customMaterial* is non-zero. This is usually only found in *prop* sections. Click on the links for more information.

locked determines whether or not the part is locked (unable to be moved) by default.

backfaceCull determines whether or not this part will have backface culling performed (the removal of facets (polygons) prior to display or render because they are facing away from the camera).

visibleInReflections determines whether or not this part is visible in reflections during render.

visibleInRender determines whether or not this part is visible during render (i.e.: Visible in Raytracing).

displacementBounds defines the part's displacement boundary in the displacement map material properties.

shadingRate defines the part's shading rate (the level to which the polygons of this part will be subdivided into micropolygons during shading calculations).

smoothPolys determines whether or not polygon smoothing is used on this part (i.e.: hard edges between polygons are rounded during render).

morphPutty section defines which morph groups to ignore during use of the Morph Putty tool in Poser 5. It contains a list of *inactiveGroups* whose names match the morph groups effected.

alternateGeom

The *alternateGeom* section defines, as expected, an alternative geometry for the part in question. This is mainly used for replacing the default *Hip* with one that has no genitalia, but can also be used for replacing other geometries.

The format is as follows:

```
alternateGeom <name>
   {
   name <name|GetStringRes(1037,<int>)>
   objFile <long> <file specifier>
   }
```

The <*name*> is an alternate name for the <*part*> name. For example, *hip* is replaced with *hip_1* when the alternate is used and so named.

name is a name used by Poser internally for referencing geometries. If the name is a standard one, *GetStringRes(1037,N)* can be used. The first argument within *GetStringRes()* defines the type of string resource pool from which to get the information while the second is an index into that pool, for whatever language is defined, for a specific string. The number *1037* denotes a string resource pool of geometry name strings. The argument <*int*> takes any of the following known values resolving to these known file specifiers:

<int>	**String Resource**
UNKNOWN	UNKNOWN

objFile references a .obj file that contains the geometry with which to replace the original. Needless to say, the replacement geometry should contain the same edge boundaries in order to seamlessly integrate with neighbor parts. The first field appears to be another MacOS-specific number. The second is a <*file specifier*> which takes a relative file path specifier in Mac format (i.e.: using ':' instead of '\' for path separation). The general format of this field is:

:Runtime:Geometries:[<paths>]:<file>.obj

where <*paths*> is any allowable number of colon-separated directories that lead to <*file*>.obj and where <*file*> is the name of an existing geometry file.

channels

The *channels* section configures dials, morphs, transformations, and joint parameters for each part. In itself, it just encloses more sections and has no parameters of its own, but there are many types of sections within it, 37 of them in all. Each section will be handled on a separate page with each type being defined on that page.

As far as can be determined, channel sections are all optional but most are included (at least once) in the *channels* section. In particular, the *nOffsetA|B* and transformation sections *(translate, rotate, scale, taper, propagatingScale,* and *smoothingScale)* are almost always present. Also, the inclusion and ordering below are irrelevant to what one will find in the actual *channels* section. For simplicity, each type has been listed once for reference.

The general format is as follows:

```
channels
  {
  groups
    {
    }
  targetGeom <name>
    {
    }
  valueParm <name>
    {
    }
  geomChan <name>
    {
    }
  xOffsetA|yOffsetA|zOffsetA <name>
    {
    }
  xOffset|yOffsetB|zOffsetB <name>
    {
    }
  taperX|Y|Z <name>
    {
    }
  scale|X|Y|Z <name>
    {
    }
  propagatingScale|X|Y|Z <name>
    {
    }
  translateX|Y|Z <name>
    {
    }
  rotateX|Y|Z <name>
    {
    }
  smoothScaleX|Y|Z <name>
    {
    }
  twistX|Y|Z <name>
    {
    }
  jointX|Y|Z <name>
    {
    }
  curveX|Y|Z <name>
    {
    }
  handGrasp|thumbGrasp|handSpread <name>
    {
    }
  pointAtParm <name>
    {
    }
  hairDynamicsParm <name>
    {
    }
  }
```

Common Channel Fields

Each type of channel section, except the new *groups* section, has a common set of fields. To avoid constant repetition, they have been grouped and defined here with links from each of the channel categories effected.

The common fields are as follows:

<channel section> <name>
{
// Common Fields
name <name|GetStringRes(1028,<int>)>
initvalue <real:dial>
hidden <0|1>
forceLimits <4|1|0>
min <real>
max <real>
trackingScale <real>
keys
interpStyleLocked <0|1>
// End Common Fields
... other fields determined by channel section type
}

name defines the name used in Poser, especially on the dials of the Parameter Palette. If this is a standard parameter dial name, *GetStringRes(1028,N)* will be used. The first argument within *GetStringRes()* defines the type of string resource pool from which to get the information while the second is an index into that pool, for whatever language is defined, for a specific string. The number *1028* denotes a string resource pool of parameter dial name strings. The argument *<int>* takes any of the following known values resolving to these known specifiers:

<int>	**String Resource**
1	Taper
2	Twist
3	Side-Side
4	Bend
5	Scale
6	xScale
7	yScale
8	zScale
9	xRotate
10	yRotate
11	zRotate
12	xTrans
13	yTrans
14	zTrans
15	Focal
16	Pitch
17	Yaw
18	Roll
19	Red
20	Green
21	Blue
22	Intensity
23	Turn
24	Front-Back
25	DollyX
26	DollyY
27	DollyZ
28	Shadow
29	Map Size
30	PanX

31	PanY
32	Zoom
33	xTranB
34	yTranB
35	zTranB
36	xOffset
37	yOffset
38	zOffset
39	xOrbit
40	yOrbit
41	zOrbit
42	Hand Type
43	BreastSize
44	OriginX
45	OriginY
46	OriginZ
47	Fatness
48	Grasp
49	Thumb Grasp
50	Spread
51	Curve
52	curve
53	Up-Down

initvalue specifies the initial value at which the Parameter Dial was set when the initial pose was 'memorized' in Poser.

hidden determines whether or not this channel is hidden in Poser. Especially useful for the *twistN, jointN,* and *smoothScaleN* sections.

forceLimits is a tristate variable whose states have the following properties:

- 0: do not use limits - can be overridden by Poser.
- 1: use limits - can be overriden by Poser.
- 4: use the limits specified by *min* and *max* - cannot be overriden by Poser.

min defines the minimum allowable dial setting.

max defines the maximum allowable dial setting.

trackingScale defines the "sensitivity" of the dial, or how fine or coarse the discreet transitions are from one value to the next as the dial is turned.

keys is a section which defines key frames for static poses and animations.

interpStyleLocked determines a fixed style of interpolation for the part (related to geometry swapping).

keys

This section defines animation key frames, but it is included in all *channels*.

The general format is as follows:

```
keys
   {
   static <0|1>
   k <long> <real>
   [sl <0|1>
   spl|lin|con
```

sm|br]
}

static determines whether this channel is animating (=0) or static (=1).

k denotes the key information. The first values is the frame, starting at 0. The second value is the channel value at this frame.

sl determines whether or not to loop animation.

spl|lin|con denote the section interpolation type: spline, linear, constant.

sm|br represent either a smooth transition or break in the transition.

groups

New to Poser 5 is the ability to group parameters in the Parameters Palette. This feature allows better organization of the dials as well as the ability to collapse groups for easier access to other groups (less scrolling). This feature is facilitated by the new *group* section. Like the *channel* section in which it resides, this section just encloses more sections and has no parameters of its own. These enclosed sections are of the type *groupNode.*

The general format is as follows:

```
groups
   {
   groupNode <name>
      {
      collapsed <0|1>
      groupNode <name>
         {
         ... and so on
         }
      ... more groupNodes
      parmNode <name>
      ... more parmNodes
      }
   ... more groupNodes
   }
```

The *groupNode* section defines a new collapsable/expandable group within the Parameter Palette. Each group has a name which is used in Poser and placed next to the collapse/expand [+/-] box for that group. As can be seen in the format description, it appears that *groupNode*s may be nested, but although I have not yet seen this construct, Poser 5 appears to allow nesting. Therefore, the assumption is that, indeed, such constructs are possible.

collapsed determines whether or not this group starts out in a collapsed or expanded state.

parmNode is a final member of the groupNode and declares a Parameter Dial within it. Note that the <*name*> must be an existing channel section name defined for the part in which it resides.

targetGeom

This section defines a morph target for a group of polygons within the part *(actor/prop)* being defined.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general format is as follows:

```
targetGeom <name>
   {
   <common fields>
   [valueOpDeltaAdd
      figureName <name>
      partName <part>
      morphName <name>
```

deltaAddDelta <real>]
indexes <long>
numDeltas <long>
deltas
{
d <long> <real:dx> <real:dy> <real:dz>
... many more d parameters
}
}

valueOpDeltaAdd is an optional command that sets up a Slave dial, which is a dial controlled by another dial. The following four parameters will only appear if this parameter is used.

figureName names the figure, usually matching the *figure* section's *name* parameter, which should be something unique to avoid Poser renaming.

partName names the part defined by the figure in *figureName.* This is the *actor* <*part*> field.

morphName names the dial or morph target in *partName* that controls this one. This is the internal name.

deltaAddDelta defines the modifier for this dial. Here, I'll quote directly from Kattman's "CR2 Autopsy" to avoid causing conflicting information:

"A further explanation of this is required so you know why this value is not equal to 1.000 like all other variations on this. In order to actually figure out the value to add to the deltaAddDelta parameter we need to do a little algebriac math. Do a little addition to find the full range, for example if the min is 0 and the max is 12 then the range is 12, if it is -2 and 12 then the range is 14. Lets say the Parent range is 12 and the Child range is 2, build the equation and solve for x using this formula (x beng the unknown deltaAddDelta): 1/Parent Range = x/Child Range. This would give us 1/12 = x/2. To solve for x we have to multiply both sides by 2 giving us 2(1/12) = x. solving this equation we get the deltaAddDelta value of 0.167 (always round to the nearest thousandth). To simplify this we could have simply stated: Child Range * (1/Parent Range) = deltaAddDelta."*

indexes specifies the number of deltas (changes) in the *deltas* section that follows.

numDeltas specifies the total number of vertices in the geometry group. Seems to me that this and the previous parameters are backwards, but this is how they work.

deltas contains the vertex indices and delta values for a morph target. It only contains one type of parameter, which has the format:

d <long> <real:dx> <real:dy> <real:dz>

where <*long*> specifies the vertex index into the part's group geometry that is to be modified and <*real:dx*> <*real:dy*> <*real:dz*> specify the target translation on each coordinate axis of the vertex. These represent the target when the Parameter Dial is set to 1.0.

valueParm

valueParm section defines a custom master dial used to control other, slaved dials. It is sometimes associated with a *targetGeom* section.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

valueParm <name>
{
<*common fields*>
}

geomChan

geomChan denotes a dial for swapping alternate geometries.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general format is as follows:

```
geomChan <name>
   {
   <common fields>
   staticValue <real>
   [uniqueInterp <0|1>]
   }
```

staticValue takes a real number and is used in conjunction with *static* in the *keys* section. If *static* is 1, then channel value (dial setting) is stored in *staticValue*.

uniqueInterp determines whether or not the channel dial has unique interpolation. If set, the dial will only move through integral values.

nOffsetA|B

Offset channels represent changes of body part origin. They are almost always used in *nOffsetA-nOffsetB* pairs where A offsets the origin for such things as scaling and rotation, then B offsets return it to its original location. For each set in the pair, each axis is defined so that: (xOffsetA, yOffsetA, zOffsetA) and (xOffsetB, yOffsetB, zOffsetB).

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general format is as follows:

```
<x|y|z>Offset<A|B> <name>
   {
   <common fields>
   staticValue <real>
   [uniqueInterp <0|1>]
   }
```

staticValue takes a real number and is used in conjunction with *static* in the *keys* section. If *static* is 1, then channel value (dial setting) is stored in *staticValue*.

uniqueInterp determines whether or not the channel dial has unique interpolation. If set, the dial will only move through integral values.

taper

taper sections define the Taper dial(s) on the Parameter Palette and control end-scaling of the part. Tapering operates on the non-twist axes. The end tapered is that which is most distant from the center of the figure's body.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

```
taper<X|Y|Z> <name>
   {
   <common fields>
   }
```

scale

scale sections are obviously the body part Scale, xScale, yScale, and zScale dials on the Parameter Palette and control the overall, X-axis, Y-axis, and Z-axis scaling, respectively.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

scale<|X|Y|Z> <name>
{
<common fields>
}

propagatingScale

propagatingScale sections will be found within the body part whose scaling affects its children. Usually found on the BODY part. The corresponding scales are passed down to the children.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

propagatingScale<|X|Y|Z> <name>
{
<common fields>
}

translate

translate sections define the xTran, yTran, and zTran dials on the Parameter Palette and control translation (linear motion) along the local coordinate axes. These are not to be confused with *nOffsetA|B* sections which set the body part's origin.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

translate<X|Y|Z> <name>
{
<common fields>
}

rotate

rotate sections define the xRotate, yRotate, and zRotate dials on the Parameter Palette and control rotation about the local coordinate axes. These should always follow the *twist/joint/joint* sections defining the joint parameters and are related to the rotation order specified by them.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

rotate<X|Y|Z> <name>
{
<common fields>
}

smoothScale

This is one of the sections that defines the joints between the parts (or bones for the parts, in 3D CG lingo). *smoothScale*'s function in reference to this is to define scaling of polygons at the joints (intersection of body parts) during bending.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

smoothScale<X|Y|Z> <name>
{
<common fields>
smoothZones <real> <real> <real> <real>
otherActor <part>
[jointMult <0|1>]
[flipped]
[calcWeights]
}

smoothZones represents the Inclusion and Exclusion angles used to determine the blending zone at the joint wherein polygons are deformed (scaled) during bending.

otherActor names the other part connected at this joint.

jointMult seems to be related to chaining body part JPs together. When a Chain Break is specified on a body part, this section is removed.

flipped will only appear in joint parameters listed for child body parts in a body part's channel list and most likely alerts Poser that the channel is a reference to a child and not the actual body part.

calcWeights has an unknown use. If included, it is assumed that weights are calculated for the body part and its children.

twist

This is one of the sections that defines the joints between the parts (or bones for the parts, in 3D CG lingo). It defines the axis along the length of which the body part twists.

One important note about *twist* and *joint* sections. Poser always defines a part's bones using one twist and two joints, each representing its own coordinate axis. The order and axes of these in the *channels* section is important as it determines the Rotation Orders for the joint. The *twist* joint parameter is nearly always placed first, but need not be first in the *channels* section since its position doesn't change. The two remaining *joint* sections must be in the correct order. Each joint triplet, especially if more than one is defined, should be grouped together.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

twist<X|Y|Z> <name>
{
<common fields>
otherActor <part>
matrixActor <part>
center <real:x> <real:y> <real:z>
startPt <real>
endPt <real>
[jointMult <0|1>]
[sphereMatsRaw
<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>

<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>]
[doBulge <0|1>]
[posBulgeLeft <real>
posBulgeRight <real>
negBulgeLeft <real>
negBulgeRight <real>]
[flipped]
[calcWeights]

}

otherActor names the other part connected at this joint.

matrixActor names another part or is set to *NULL.* Its use is unknown.

center defines the center for the joint rotation.

startPt defines the start point of the joint's twist area.

endPt defines the end point of the joint's twist area.

jointMult seems to be related to chaining body part JPs together. When a Chain Break is specified on a body part, this section is removed.

sphereMatsRaw defines two right-handed 4x4 matrices in the lines following it. These are the joint parameter settings for the innerSphere and outerSphere used for blending zones. They represent transformation matrices to be applied to the innerMatSphere and outerMatSphere, respectively, each starting as a unit sphere (R=1.0) at the origin (0,0,0).

doBulge determines whether or not the part's geometry is deformed by joint bending. If this is set to 1, the following four parameters are included. If this body part contains a Chain Brake, this is also removed (see *jointMult*).

posBulgeLeft sets the amount of positive deformation (away from the center) when the joint is moved to the left (however this is to be interpreted).

posBulgeRight sets the amount of positive deformation (away from the center) when the joint is moved to the right (however this is to be interpreted).

negBulgeLeft sets the amount of negative deformation (toward from the center) when the joint is moved to the left (however this is to be interpreted).

negBulgeRight sets the amount of negative deformation (toward from the center) when the joint is moved to the right (however this is to be interpreted).

flipped will only appear in joint parameters listed for child body parts in a body part's channel list and most likely alerts Poser that the channel is a reference to a child and not the actual body part.

calcWeights has an unknown use. If included, it is assumed that weights are calculated for the body part and its children.

joint

This is one of the sections that defines the joints between the parts (or bones for the parts, in 3D CG lingo). It is used to define the remaining two axes of rotation not covered by the *twist* channel.

One important note about *twist* and *joint* sections. Poser always defines a part's bones using one twist and two joints, each representing its own coordinate axis. The order and axes of these in the *channels* section is important as it determines the Rotation Orders for the joint. The *twist* joint parameter is nearly always placed first, but need not be first in the *channels* section since its position doesn't change. The two remaining *joint* sections must be in the correct order. Each joint triplet, especially if more than one is defined, should be grouped together.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

```
joint<X|Y|Z> <name>
   {
   <common fields>
   otherActor <part>
   matrixActor <part>
   center <real:x> <real:y> <real:z>
   angles <real> <real> <real> <real>
   [jointMult <0|1>]
   [sphereMatsRaw
```

<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>

<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>
<real> <real> <real> <real>]
[doBulge <0|1>]
[posBulgeLeft <real>
posBulgeRight <real>
negBulgeLeft <real>
negBulgeRight <real>]
[flipped]
[calcWeights]
}

otherActor names the other part connected at this joint.

matrixActor names another part or is set to *NULL.* Its use is unknown.

center defines the center for the joint rotation.

angles defines four angles of the joint. They appear as a set of four intersecting lines in the Joint Parameter Window. These angles determine Inclusion, Exclusion, and blending zones between the part and its children.

jointMult seems to be related to chaining body part JPs together. When a Chain Break is specified on a body part, this section is removed.

sphereMatsRaw defines two right-handed 4x4 matrices in the lines following it. These are the joint parameter settings for the innerSphere and outerSphere used for blending zones. They represent transformation matrices to be applied to the innerMatSphere and outerMatSphere, respectively, each starting as a unit sphere (R=1.0) at the origin (0,0,0).

doBulge determines whether or not the part's geometry is deformed by joint bending. If this is set to 1, the following four parameters are included. If this body part contains a Chain Brake, this is also removed (see *jointMult*).

posBulgeLeft sets the amount of positive deformation (away from the center) when the joint is moved to the left (however this is to be interpreted).

posBulgeRight sets the amount of positive deformation (away from the center) when the joint is moved to the right (however this is to be interpreted).

negBulgeLeft sets the amount of negative deformation (toward from the center) when the joint is moved to the left (however this is to be interpreted).

negBulgeRight sets the amount of negative deformation (toward from the center) when the joint is moved to the right (however this is to be interpreted).

flipped will only appear in joint parameters listed for child body parts in a body part's channel list and most likely alerts Poser that the channel is a reference to a child and not the actual body part.

calcWeights has an unknown use. If included, it is assumed that weights are calculated for the body part and its children.

curve

The *curve* section, containing only common fields, defines a body part as being curved (tail, tentacle, and so on). It will create an associated dial to control the amount of curve for the part.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

curve <name>
{
<common fields>
}

handGrasp, handSpread, thumbGrasp

These three sections control macro manipulation hands. Each represents a similarly named dial in the Parameters Palette. *handGrasp* closes all of the fingers of the hand. *handSpread* spreads all of the fingers away from one another. *thumbGrasp* closes the thumb, bringing it in towards the palm.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

handGrasp <name>
{
<common fields>
}

handSpread<name>
{
<common fields>
}

thumbGrasp<name>
{
<common fields>
}

pointAtParm and pointAtTarget

This section is directly linked to the "Point At" feature of Poser and is supposedly coupled to a *pointAtTarget* line, but mainly shows up in Poser scene files (pz3). It creates a dial in the Parameters Palette for the controlled object.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general formats are as follows:

pointAtParm <name>
{
<common fields>
}

hairDynamicsParm

This section is new to Poser 5 and is only found in *hairProp* sections using Dynamic Hair.

Note that there is a common set of fields to all but one of the channel sections. Since this is the case, these common fields are defined on a separate page linked from each category page within the format description.

The general format is as follows:

hairDynamicsParm <name>
{
<common fields>
hairCacheFile <_NONE_|<path>>
}

hairCacheFile seems to be a parameter which points to a file, if any, that contains information for Dynamic simulation using the hair.

figure

The *figure* section is an important section in that it brings all of the actors and props together, defining their hierarchical relationships, IK chains, and material properties. Here is the format:

```
figure
    {
    name <default name>
    root <part>
    addchild <part:child>
        <part:parent>
    ... more addchild commands
    [inkyChain <name>
        {
        }
    ... more inkyChain sections]
    defaultPick <part>
    displayOn <1|0>
    weld <part:child>
        <part:parent>
    ... more weld commands
    [linkParms <part>
        <name>
        <part>
        <name>
    ... more linkParm commands]
    allowsBending <1|0>
    figureType <int>
    origFigureType <int>
    canonType <int>
    conforming <0|1>
    material <name>
        {
        }
    ... more material sections
    presetMaterial <name>
        {
        }
    ... more presetMaterial sections
    displayMode <TYPE>
    locked <0|1>
    }
```

The *name* parameter specifies the default name for Poser to display for this figure when inserted into a document. I say "default" because if, for instance, "Figure 1" is already in the document, Poser will change this to "Figure 2" and so forth in order to retain unique names.

root defines the root part in the figure hierarchy, which is usually *BODY:<N>.* All other parts are children of this one.

addchild is a command which sets a hierarchical relationship between two parts. The first field, *<part:child>*, is the *<name>:<number>* of the part to be made a child of the second field, *<part:parent>*, which is always on a separate line.

The *inkyChain* sections create ordered chains of parts for use with inverse kinematics. Click on its name in the format or above to get details of its contents.

defaultPick defines the *<part>* that is to be selected by default within Poser.

displayOn determines whether or not the figure is visible when added to a document.

weld is the second command and specifies which joints between two parts need to be welded so that the mesh does not break apart while flexing them. The fields are identical to *addchild*'s.

linkParms is an archaic master/slave dial system rarely seen these days. Click on the link to see the format.

allowsBending determines whether or not mesh deforms or breaks when bending.

figureType and *origFigureType* are obsolete Poser 1 parameters.

canonType specifies the number of heads high the figure is set to.

conforming states whether or not this is a conforming figure.

material and *presetMaterial* sections define the materials for each texture defined within the various .obj data used by the figure. Click on the links for more information.

displayMode sets the Poser mode in which to display the figure. The possible settings are:

- USEPARENT
- CARTOONNOLINE
- FLATLINED
- SMOOTHLINED
- SHADEDOUTLINE
- SKETCHSHADED
- FLATSHADED
- TEXTURESHADED
- SHADED
- HIDLINE
- SILHOUETTE
- EDGESONLY
- WIREFRAME

locked determines whether or not the figure can be moved or posed.

inkyChain

The *inkyChain* section defines the parts that make up and parameters of an inverse kinematic chain of bones, as follows:

```
inkyChain <name>
   {
   <on|off>
   name <name>
   addLink <part>
   ... more addLinks
   goal <part>
   linkWeight <int:Link> <real:Weight>
   ... more linkWeights addLinks
   }
```

The <*name*> field defines the internal name of the IK.

<*on*|*off*> is a parameter that takes one of these two values, determining whether the IK chain is initially active within Poser.

name defines the name of the IK within Poser, which is displayed in the **Figure->Use Inverse Kinematics** menu. Spaces are allowed.

addLink adds a <*part*> (in standard <*name:number*> format) to the IK chain list. The order here is important and should start at the end opposite the *goal.*

The end of the chain is the *goal*, which is the part which effects the others linked to it.

After this are a series of *linkWeight* parameters, which must equal the number of *addLink* parameters. Each specifies the link, by number in the *addLink* list, starting at 0, and the weight of that link to the next link as a value from 0.0 to 1.0. Typical value-series are 1.0, 0.2, 0.04, 0.008, etc. which are inverse powers of 5: 1/(5^0), 1/(5^1), 1/(5^2), 1/(5^3), etc.

linkParms

The *linkParms* section defines a master-slave relationship between two parts' channels. The format is:

```
linkParms <part>
   <name>
   <part>
   <name>
```

The first *<part>,<name>* pair define the master body part and its channel using internal names.

The second pair define the slave body part and its channel using internal names.

material and presetMaterial

The *material* and *presetMaterial* sections define material parameters and maps for the material groups of each figure. Since the *presetMaterial* section is identical to the *material* section, it is included here:

```
material|presetMaterial <name>
   {
   KdColor <real:R> <real:G> <real:B> <real:Strength>
   KaColor <real:R> <real:G> <real:B> <real:Strength>
   KsColor <real:R> <real:G> <real:B> <real:Strength>
   TextureColor <real:R> <real:G> <real:B> <real:Strength>
   NsExponent <real>
   bumpStrength <real>
   ksIgnoreTexture <0|1>
   [ *** THESE PARAMETERS ARE OPTIONAL
   tMin <real>
   tMax <real>
   tExpo <real>
   reflectThruLights <1|0>
   reflectThruKd <0|1>
   textureMap <NO_MAP|quoted file specifier>
      <[0 0]>
   bumpMap <NO_MAP|quoted file specifier>
      <[0 0]>
   reflectionMap <NO_MAP|quoted file specifier>
      <[0 0]>
   transparencyMap <NO_MAP|quoted file specifier>
      <[0 0]>
   ReflectionColor <real:R> <real:G> <real:B> <real:Strength>
   reflectionStrength <real>
   *** ]
   shaderTree
      {
      }
   }
```

Each *material* has a *name*, usually corresponding to the material zone (group of polygons) onto which it is applied. These names must match the *usemtl* fields defined within the Wavefront .obj file.

KdColor, KaColor, KsColor, TextureColor and *ReflectionColor* each specify RGB values with a strength setting tagged on. The values for each field are from 0.0 to 1.0.

KdColor specifies the material's diffuse color.

KaColor specifies the material's ambient color (for ambient lighting).

KsColor specifies the material's specular color (for specular highlights).

TextureColor specifies a color, but the RGB is always 1,1,1. The strength setting affects the *textureMap* image.

NsExponent is the highlight size for specular highlighting. Its value ranges from 0 to 100.

tMin is the transparency minimum value: 0.0 to 1.0.

tMax is the transparency maximum value: 0.0 to 1.0.

tExpo is the transparency falloff: 0.0 to 1.0.

bumpStrength is the strength of the bump map (how much apparent elevation is added to the bump). This takes both positive and negative values which range from -1.0 to 1.0.

ksIgnoreTexture determines whether or not to apply texture to specular highlight.

reflectThruLights determines whether or not to multiply reflection through lights.

reflectThruKd determines whether or not to multiply reflection through object color.

ReflectionColor specifies the material's reflection color.

reflectionStrength is the amount or "depth and clarity" of the reflection: 0.0 to 1.0.

textureMap, bumpMap, reflectionMap, and *transparencyMap* each specify images to be used for material mapping. If no map is being used, *NO_MAP* is specified in the data field. If a map is specified, it is a quoted file specifier in the format:

"[:Runtime:textures]:[<paths>]:<image file>"

where *<paths>* is any allowable number of colon-separated directories that lead to *<image file>*. Note that *:Runtime:textures* is optional as Poser will automatically start here when seeking texture images. An extra line, *0 0*, is added when a file is specified.

shaderTree is a new Poser 5 material addition which is backward compatible with the previous format while adding new procedural functionality.

shaderTree

In Poser 5, a new "node-based" material system was created. And in light of this, new sections were created to codify this system.

```
shaderTree
   {
   node <type> <name>
      {
      }
   ... more nodes
   }
```

Each *shaderTree* contains *nodes*, the main node called "Poser Surface". This corresponds to the main shader control for the material. Any other nodes are plugged into this main node.

node

In Poser 5, a new "node-based" material system was created. And in light of this, new sections were created to codify this system. A *node* contains information for each *node* tied to a material or other *nodes*.

```
node <type> <name>
   {
   name <string>
   pos <x y coordinates>
   showPreview <0|1>
   nodeInput <name>
      {
      }
   ... more nodeInputs
   }
```

type and *name* are both strings which identify the *node*.

pos specifies the (x,y) location of the *node* in the Material workspace of the Material Room.

showPreview determines whether or not the preview image is displayed for the *node*.

nodeInputs specify the values for the settings in the *node*.

nodeInput

In Poser 5, a new "node-based" material system was created. And in light of this, new sections were created to codify this system. A *nodeInput* specifies particular characteristics of a *node,* such as color, strength, procedural values, or an image map.

```
nodeInput <name>
   {
   name <string>
   value <RGB>
   parmR NO_PARM
   parmG NO_PARM
   parmB NO_PARM
   node <NO_NODE|string>
   file <""|<path>>
   }
```

name is the name of this *nodeInput*.

value contains RGB values ranging from 0.0 to 1.0.

parmR, *parmG*, *parmB* are also RGB values, but are always set to *NO_PARM*. As Bloodsong states in her book, these may exist only for Renderman compliance.

node specifies another *node* that inputs into this one. Either *NO_NODE* or the name of the input *node*.

file specifies a material or image map. If none is used, double-quotes ("") are present.

setGeomHandlerOffset

This is the last field of a CR2 file, just before the file-level closing brace. The *setGeomHandlerOffset* parameter section is formatted as follows:

```
setGeomHandlerOffset <real> <real> <real>
```

The three fields define real values supposedly representing (x,y,z) coordinates, but as is the case for *storageOffset*, their function is unknown.