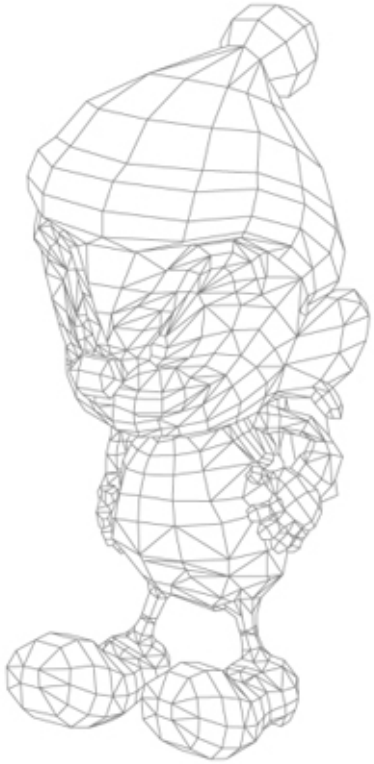
MASCOT CAPSULE®
REAL-TIME 3DRENDERING ENGINE

BAC File Ver.6.0
Data Format Specification

MascotCapsule V3

English version



Ver.2.1

HI CORP.

— Revision History —

October 8, 2003
- Ver.1.0 Created

June 15, 2005
- Ver.1.0

Updated
- Corrected descriptions and page layout.

March 20, 2007
- Ver.2.0

Updated
- Corrected descriptions and page layout.
- Added sample code.

August 24, 2007
- Ver.2.1

Updated
- Corrected descriptions and page layout.
- Changed descriptions of the chunks that define bone coordinates.

— Table of Contents —

1. Introduction

This document provides explanation of the **BAC file** (we call it **BAC** from now on), which is an intermediate file of MascotCapsule V2/V3 (**V2**/**V3** from now on).
BAC is a data file that contains model information to be used in **V2**/**V3**; there are versions **5.0** and **6.0** (**BAC5**/**BAC6** from now on). **BAC5** is a data file that contains model information to be used in **V2**. On the other hand, **BAC6** is a data file that contains model information to be used in **V3**. This document provides detailed description of the **BAC6** data format.

1.1. What is BAC6?

BAC6 is a text format file (**BAC5** is a binary format file); using text strings, **BAC6** describes material attributes or bone information applied for the model data to be used in **V3**.
On the other hand, **TRA4** file defines animation information. For more details, please refer to the separate document, "**TRA File Ver.4.0 Data Format Specification**." **BAC6** and **TRA4** are a pair of corresponding intermediate files to be used in **V3**.
BAC6 has "**.bac**" file name extension. Text format characters are supported within "**ASCII code**" range.

2. About macros

In order to simplify the explanation and description of **BAC6**, this document uses macro definitions. The following section provides explanation of each macro.

2.1. Macro definition

Parameter portion of **BAC6** is defined by the macros described in following subsections:

2.1.1. [STRING]

The **[STRING]** macro signifies a null-terminated text string.
Text string must be enclosed in double quotation marks (**""**), and must not exceed 255 characters (byte).

2.1.2. [A|B]

The **[A | B]** macro signifies **A** or **B**.
A or **B** is a null-terminated text string; it must not exceed 255 characters (byte).

2.1.3. [INT]

The **[INT]** macro signifies a text string that represents integer values.
Text string (numeric value) must fit within the bit range of **int** type.

2.1.4. [FLOAT]

The **[FLOAT]** macro signifies a text string that represents floating point values.
Text string (numeric value) must fit within the bit range of **float** type.

2.1.5. …

The **…** macro signifies arbitrary number of repetition.

2.1.6. ; (Semicolon)

From the semicolon to the end of line, it is a comment text string.

3. File structure

This chapter provides details of each chunk to be defined, and the basic coding convention of **BAC6**.

3.1. BAC6 file structure

BAC6 consists of "**;BAC**" at the top as a file identifier, and two chunks: the **(Head)** and **(Figure)** chunks. After defining the **(Head)** chunk signifying the header, you must define the **(Figure)** chunk specifying the model information. The order of these definitions should not be altered.
The **(Head)** chunk and **(Figure)** chunk respectively contain a series of subchunks; and each of subchunk includes dependent subchunks.
Please keep these in mind when you look at **Figure 1** "**BAC6** file structure."

Figure 1 **BAC6** file structure

```
;BAC

( Head
        ( bacVersion [FLOAT] )
)
( Figure
        ( name [STRING] )

        ( Textures
                ( i2 [INT] [INT] )
                …
        )

        ( Colors
                ( f3 [FLOAT] [FLOAT] [FLOAT] )
                …
        )

        ( Materials
                ( material
                        ( blendMode      [normal | add | sub | half] )
                        ( doubleFace     [true | false] )
                        ( transparent    [true | false] )
                        ( lighting       [true | false] )
                        ( textureIndex   [INT] )
                        ( colorIndex     [INT] )
                        ( specular       [FLOAT] )
                        ( alpha          [FLOAT] )
                        ( shininess      [FLOAT] )
                )
                …
        )

        ( Vertices
                ( coords
                        ( pnt [FLOAT] [FLOAT] [FLOAT] )
                        …
                )
                ( normals
                        ( vct [FLOAT] [FLOAT] [FLOAT] )
                        …
                )
        )
```

```
    ( Bones
          ( bone
                        ( name           [STRING] )
                        ( hasChild       [true | false] )
                        ( hasBrother     [true | false] )
                        ( translate      [FLOAT] [FLOAT] [FLOAT] )
                        ( rotate         [FLOAT] [FLOAT] [FLOAT] )
                        ( handle         [FLOAT] [FLOAT] [FLOAT] )
                        ( vertexIndices
                                [INT] [INT] …
                        )
           )
          …
    )

    ( TextureCoords
           ( f2 [FLOAT] [FLOAT] )
           …
    )

    ( Polygons
           ( face [INT] ( i3 [INT] [INT] [INT] )( i3 [INT] [INT] [INT] ))
           ( face [INT] ( i4 [INT] [INT] [INT] [INT] )( i4 [INT] [INT] [INT] [INT] ))
           …
    )
    ( DynamicPolygons
       ( group
           ( name [STRING] )
           ( face [INT] ( i3 [INT] [INT] [INT] )( i3 [INT] [INT] [INT] ))
           ( face [INT] ( i4 [INT] [INT] [INT] [INT] )( i4 [INT] [INT] [INT] [INT] ))
           …
       )
       …
    )
)
```

4. About each chunk

This chapter provides details of each chunk to be defined in **BAC6**.

4.1. (Head) chunk

Specify the version number of the **BAC** in the **(bacVersion [FLOAT])** chunk, in order to define the file header. **BAC6** is version 6.0; therefore, fill in the text string **6.0** in **[FLOAT]** as definition.
[See also] **Figure 2** ”Example of **(Head)** chunk definition”

```
( Head
        ( bacVersion 6.0 )
)
```

Figure 2 ”Example of **(Head)** chunk definition”

4.2. (Figure) chunk

Specify the model data (polygon mesh) definition in the **(Figure)** chunk.
Use subchunks within the **(Figure)** chunk, in order to define specific details of model data.
[See also] **Figure 3** ”Example of **(Figure)** chunk definition”

```
( Figure
        ( name … )
        ( Textures … )
        ( Materials … )
        ( Vertices … )
        ( Bones … )
        ( Polygons … )
        ( DynamicPolygons … )
)
```

Figure 3 ”Example of **(Figure)** chunk definition”

Use subchunks within the **(Figure)** chunk, in order to define the details of model data.
The following subsections provide details on each subchunk.

4.2.1. (name) chunk

In the **(name [STRING])** chunk within the **(Figure)** chunk, define the name of model data using a text string not exceeding 255 characters (byte).
You do not necessarily have to define the model name, but when you do, you must enclose it in double quotation marks as in **(name "")**.
[See also] **Figure 4** ”Example of **(name)** chunk definition”

```
( name "figure" )
```

Figure 4 ”Example of **(name)** chunk definition”

[Related links] 4.2.6.1.1. (name) chunk
4.2.9.1.1. (name) chunk

4.2.2. (Textures) chunk

In the **(Textures)** chunk, define the pixel size (width and height) of texture image to be used for the model data.
For the model data that does not have texture information (polygon mesh), you don't need to define this chunk within the **(Figure)** chunk.
When you define this chunk, you must always define it within the **(Figure)** chunk.

[Related links] 4.2.7. (TextureCoords) chunk

4.2.2.1. (i2 [INT] [INT]) chunk

Define the pixel size (width and height) of texture image.
Index ID is automatically assigned to this chunk from the top; it is a sequential ID starting from **0**.
This index ID is referred by the **(textureIndex [INT])** chunk within the **(material)** chunk.
Also, the **(i2)** chunk must always be defined within **(Textures)** chunk, and cannot be used in other chunks.
[See also] **Figure 5** ”Example of **(Textures)** chunk definition”

```
( Textures
       ( i2 256 256 ) ; index 0
       ( i2 128 128 ) ; index 1
       …
)
```

Figure 5 ”Example of **(Textures)** chunk definition”

[Related links] 4.2.4.1.5. (textureIndex) chunk

4.2.3. (Colors) chunk

In the **(Colors)** chunk, define the color table for color polygons to be used for the model data.
For the model data without color polygons, you don't need to define this chunk within the **(Figure)** chunk. When you define this chunk, you must always define it within the **(Figure)** chunk.

4.2.3.1. (f3 [FLOAT] [FLOAT] [FLOAT]) chunk

This chunk defines color tables for the color polygons. Index ID is automatically assigned to this chunk from the top; it is a sequential ID starting from **0**. This index ID is referred by the **(colorIndex [INT])** chunk within the **(material)** chunk.
Valid range for the color is described as follows: from **black=RGB (0.0, 0.0, 0.0)** to **white=RGB (1.0, 1.0, 1.0)**.
[See also] **Figure 6** "Example of **(Colors)** chunk definition"

```
( Colors
       ( f3 0.000 0.000 0.000 ) ; index 0  [Black]
       ( f3 1.000 1.000 1.000 ) ; index 1  [White]
       …
)
```

Figure 6 "Example of **(Colors)** chunk definition"

[Related links] 4.2.4.1.6. (colorIndex) chunk

4.2.4. (Materials) chunk

In the **(Materials)** chunk, define the arbitrary display attributes to be specified for the polygons in the model data.
When you define this chunk, you must always define it within the **(Figure)** chunk.
Details of each subchunk within the **(Materials)** chunk are as follows:
[See also] **Figure 7** "Example of **(Materials)** chunk definition"

```
( Materials
       ( material
              ( blendMode   normal )
              ( doubleFace  true )
              ( transparent false )
              ( lighting    true )
              ( textureIndex -1 )
              ( colorIndex    0 )
              ( specular      0.000 )
              ( alpha         0.000 )
              ( shininess     0.000 )
       )
       …
)
```

Figure 7 "Example of **(Materials)** chunk definition"

4.2.4.1. (material) chunk

In the **(material)** chunk, define the attributes to be specified for the polygons. At least one **(material)** chunk must be defined within the **(Materials)** chunk.
Index ID is automatically assigned to the **(material)** chunk from the top; it is a sequential ID starting from **0**. This index ID is referred by the **(face)** chunk within the **(Polygons)** chunk.
[Related links] 4.2.8.1. (face) chunk

Please read the description in **4.2.4.1.1** to **4.2.4.1.9** for each subchunk to be defined within the **(material)** chunk. There are no rules for the order of defining each subchunk within the **(material)** chunk.

4.2.4.1.1. (blendMode [normal | add | sub | half]) chunk

Specify one of the following attributes in the **(blendMode [normal | add | sub | half])** chunk, in order to define the fragment process for displaying polygons.

Note: d_c = destination color; s_c = source color

normal	Performs the standard fragment process (Substitution).	$d_c = s_c$
add	Adds and blends the polygon colors (Additive blend).	$d_c = d_c + s_c$
sub	Subtracts the color value of the polygon (Subtractive blend).	$d_c = d_c - s_c$
half	Overlays the color using the half values of the polygon colors (Semi-transparency).	$d_c = 0.5\ d_c + 0.5\ s_c$

4.2.4.1.2. (doubleFace [true | false]) chunk

Use the **(doubleFace [true | false])** chunk to specify whether both sides of polygons are rendered.

true	Enables the **double face rendering** for polygons.
false	Enables rendering for the front side of polygons, but not for the backside.

4.2.4.1.3. (transparent [true | false]) chunk

Use the **(transparent [true | false])** chunk to specify whether the **BMP** color palette **0** for the texture color should be transparent or not.

true	Enables the transparency effect.
false	Disables the transparency effect.

4.2.4.1.4. (lighting [true | false]) chunk

Use the **(lighting [true | false])** chunk to enable/disable the lighting effect.

true	Enables the lighting effect.
false	Disables the lighting effect.

4.2.4.1.5. (textureIndex [INT]) chunk

In the **(textureIndex [INT])** chunk, specify the index ID that was automatically assigned in the **(Textures)** chunk. Specify **-1** if you do not want a textured polygon.

[Related links] 4.2.2. (Textures) chunk

4.2.4.1.6. (colorIndex [INT]) chunk

In the **(colorIndex [INT])** chunk, specify the color ID registered in the **(Colors)** chunk.
You must specify the index ID that was automatically assigned to the **(f3)** chunk defined within the **(Colors)** chunk. Specify **-1** if you do not want color polygons.

[Related links] 4.2.3. (Colors) chunk

4.2.4.1.7. (specular [FLOAT]) chunk

Define the fragment value of specular reflection in the **(specular [FLOAT])** chunk, in order to set up environment mapping. Valid range of the value is **0.00** to **1.00**; when **0.25** or larger values are defined, environment mapping is enabled for the specular reflection process.

4.2.4.1.8. (alpha [FLOAT]) chunk

Define the **semi-transparent alpha value** for the texture in the **(alpha [FLOAT])** chunk. Valid range of the value is **0.0** to **1.0**.

V3 does not support this functionality. Therefore, when it is not necessary, you do not have to define this chunk in the **(material)** chunk.

4.2.4.1.9. (shininess [FLOAT]) chunk

Define the **specular reflection coefficient** for environment mapping in the **(shininess [FLOAT])** chunk. Valid range of the value is **0.0** to **1.0**.

V3 does not support this functionality. Therefore, when it is not necessary, you do not have to define this chunk in the **(material)** chunk.

4.2.5. (Vertices) chunk

In the **(Vertices)** chunk, define the vertex position that comprises the model data, and the normal vector for that vertex.

[See also] **Figure 8** ”Example of **(Vertices)** chunk definition”

```
( Vertices
        ( coords
                ( pnt -1.000 -1.000 0.000 ) ; index 0
                ( pnt 1.000 -1.000 0.000 )  ; index 1
                ( pnt 1.000 1.000 0.000 )   ; index 2
                ( pnt -1.000 0.000 0.000 )  ; index 3
                …
        )
        ( normals
                ( vct 0.000 0.000 1.000 )  ; index 0
                ( vct 0.000 0.000 1.000 )  ; index 1
                ( vct 0.000 0.000 1.000 )  ; index 2
                ( vct 0.000 0.000 1.000 )  ; index 3
                …
        )
)
```

You must define the vertex coordinates in the **(pnt)** chunk index **0**, so that they retain the correspondence with the normal vector values defined in the **(vct)** chunk index **0**.
The number of vertices must match the number of corresponding pairs: vertex coordinates and normal vector definitions.

Figure 8 ”Example of **(Vertices)** chunk definition”

4.2.5.1. (coords) chunk

In the **(coords)** chunk, define the vertex coordinates of polygons that comprise the model data.
You must always define this chunk within the **(Vertices)** chunk.

4.2.5.1.1. (pnt [FLOAT] [FLOAT] [FLOAT]) chunk

In the **(pnt)** chunk, use model coordinate system to define the three-dimensional vertex position **(XYZ)** that comprises polygons.
Index ID is automatically assigned to this chunk from the top; it is a sequential ID starting from **0**. This index ID must match the index ID of the **(vct)** chunks listed within the **(normals)** chunk.
Also, this index ID is referred by the **(face)** chunk that is used by the **(Polygons)** chunk, etc.

[Related links] 4.2.8. (Polygons) chunk

4.2.5.2. (normals) chunk

In the **(normals)** chunk, define the normal vector of polygons that comprise the model data.
You must always define this chunk within the **(Vertices)** chunk.

4.2.5.2.1. (vct [FLOAT] [FLOAT] [FLOAT]) chunk

In the **(vct)** chunk, use model coordinate system to define the normal vector **(XYZ)** of vertex that comprises polygons.
Index ID is automatically assigned to this chunk from the top; it is a sequential ID starting from **0**. This index ID must match the index ID of the **(pnt)** chunks listed within the **(coords)** chunk.
Also, this index ID is referred by the **(face)** chunk that is used by the **(Polygons)** chunk, etc.

[Related links] 4.2.8. (Polygons) chunk

4.2.6. (Bones) chunk

In the **(Bones)** chunk, define the hierarchical structure of bones that are assigned to the model data, as well as each underlying bone coordinate system.
At least one bone must be defined within the **(Figure)** chunk in order to construct a model data. Also, the first **(bone)** chunk to be defined within the **(Bones)** chunk represents a root bone, which is the highest in the bone hierarchy structure. Only one root bone must be defined.
This subsection provides explanation of the **(bone)** chunk contained in the **(Bones)** chunk.
[See also] **Figure 9** “Example of bone definition”, and **Figure 10** “Example of bone structure”

```
( Bones
     ( bone
        ( name            "parent" )
        ( hasChild        true )
        ( hasBrother      false )
        ( translate       0.000 2.000 0.000 )
        ( rotate          0.000 2.000 1.000 )
        ( handle          0.000 3.000 0.000 )
        ( vertexIndices
           0 1 2 3
        )
     )
     ( bone
        ( name            "child_1" )
        ( hasChild        false )
        ( hasBrother      true )
        ( translate       2.000 0.500 0.000 )
        ( rotate          2.000 0.500 1.000 )
        ( handle          2.000 2.000 0.000 )
        ( vertexIndices
           4 5 6 7
        )
     )
     ( bone
        ( name            "child_2" )
        ( hasChild        false )
        ( hasBrother      false )
        ( translate       1.000 0.500 0.000 )
        ( rotate          1.000 0.500 1.000 )
        ( handle          1.500 1.500 0.000 )
        ( vertexIndices
           8 9 10 11
        )
     )
)
```

Figure 9 ”Example of bone definition”

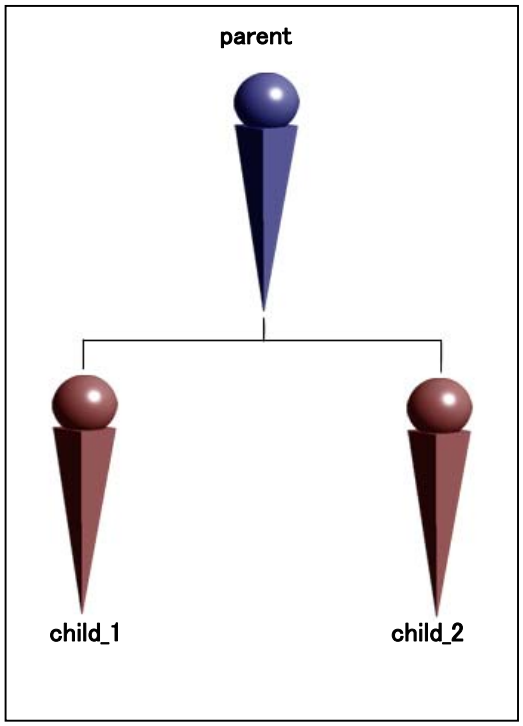


Figure 10 ”Example of bone structure”

Note: Bone animation is defined in **TRA** file as a translation from each bone coordinate system.

4.2.6.1. (bone) chunk

The **(bone)** chunk is the parent of each chunk that defines a single bone. In the **(Bones)** chunk, you must define the same number of **(bone)** chunks as the assigned bones in the model data.

The **(bone)** chunk has a child chunk that defines relationship such as hierarchy in relation to other bones, etc.

Please read the description in **4.2.6.1.1** to **4.2.6.1.7** for each chunk that defines the detail of the bone.

4.2.6.1.1. (name [STRING]) chunk

In the **(name)** chunk, define the name of the target bone.

You do not necessarily have to define the target bone name, but when you do, you must enclose it in double quotation marks as in **(name "")**.

[Related links] 4.2.1. (name) chunk

4.2.6.1.2. (hasChild [true | false]) chunk

In the **(hasChild)** chunk, define whether the target bone has a child bone or not.

true	Defines a child bone for the target bone
false	Does not define a child bone for the target bone

When **true** is defined, it means that the next defined **(bone)** chunk is defined as target bone's child.

[See also] **Figure 11** ”Parent-child relationship of bones”

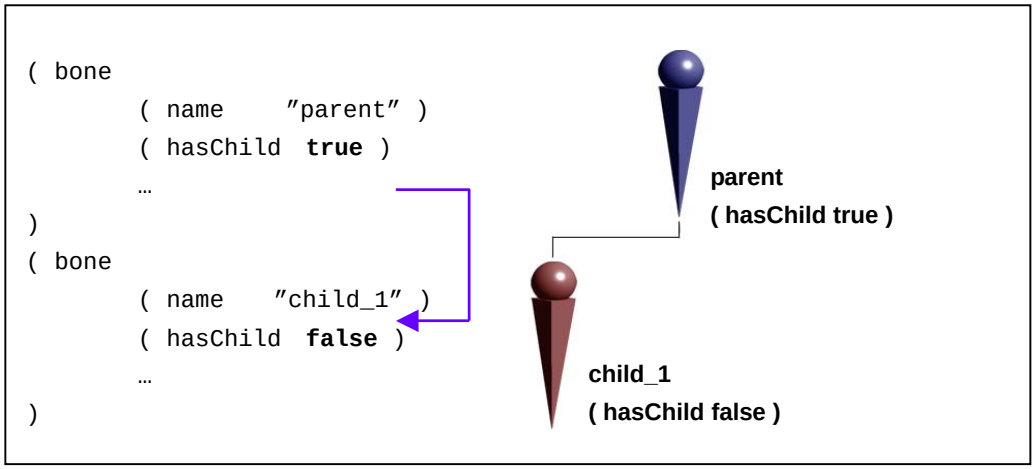


Figure 11 “Parent-child relationship of bones”

[Related links] 4.2.6.2. Additional notes on bone structure

4.2.6.1.3. (hasBrother [true | false]) chunk

In the **(hasBrother [true | false])** chunk, define whether a brother bone (in the same hierarchy) exists or not for the target bone.

true	This bone has a brother bone in the same hierarchy.
false	This bone does not have a brother bone in the same hierarchy.

When **true** is defined, it means that the target bone has a brother bone in the same hierarchy. In order to define the last bone in the same hierarchy, you must always define **false** for the **(hasBrother)** chunk.

[See also] **Figure 12** “Bones in the same hierarchy”

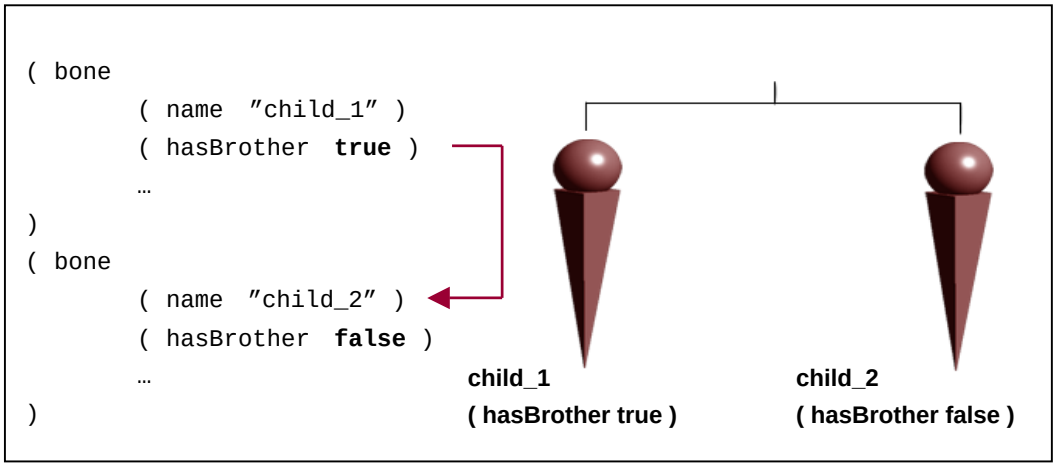


Figure 12 “Bones in the same hierarchy”

[Related links] 4.2.6.2. Additional notes on bone structure

4.2.6.1.4. (translate [FLOAT] [FLOAT] [FLOAT]) chunk

In the **(translate)** chunk, define the target bone coordinates' origin position in the model coordinate system.

When the target bone coordinate system's origin position is the same as the model coordinate system's origin position, you can define the value as **(translate 0.0 0.0 0.0)**.

[See also] **Figure 13** “Correlation of the bone coordinate system”

[Related links] 4.2.6.1.5. (handle) chunk , and 4.2.6.1.6 (rotate) chunk

4.2.6.1.5. (handle [FLOAT] [FLOAT] [FLOAT]) chunk

In the **(handle)** chunk, specify the target bone coordinate system's +Y-axis position using the model coordinate system, as well as the origin position defined in the **(translate)** chunk.

The direction vector of +Y-axis in the model coordinate system can be obtained as shown below:

$$\overrightarrow{\text{(handle)}} - \overrightarrow{\text{(translate)}}$$

Therefore, the direction vector of +Y-axis (shown below) must be. . .

$$\overrightarrow{\text{(handle)}} - \overrightarrow{\text{(translate)}}$$

orthogonal to the direction vector of +Z-axis (shown below).

$$\overrightarrow{\text{(rotate)}} - \overrightarrow{\text{(translate)}}$$

When the **(translate)** chunk is **(translate 0.0 0.0 0.0)**, in order to match the bone coordinate system's Y-axis and the model coordinate system's Y-axis, you can define the value as **(handle 0.0 1.0 0.0)**.

[See also] **Figure 13** “Correlation of the bone coordinate system”

[Related links] 4.2.6.1.4. (translate) chunk , and 4.2.6.1.6 (rotate) chunk

4.2.6.1.6. (rotate [FLOAT] [FLOAT] [FLOAT]) chunk

In the **(rotate)** chunk, specify the target bone coordinate system's +Z-axis position using the model coordinate system, as well as the origin position defined in the **(translate)** chunk.

The direction vector of +Z-axis in the model coordinate system can be obtained as shown below:

$$\overrightarrow{(\text{rotate})} - \overrightarrow{(\text{translate})}$$

Therefore, the direction vector of +Z-axis (shown below) must be. . .

$$\overrightarrow{(\text{rotate})} - \overrightarrow{(\text{translate})}$$

orthogonal to the direction vector of +Y-axis (shown below).

$$\overrightarrow{(\text{handle})} - \overrightarrow{(\text{translate})}$$

When the **(translate)** chunk is **(translate 0.0 0.0 0.0)**, in order to match the bone coordinate system's Z-axis and the model coordinate system's Z-axis, you can define the value as **(rotate 0.0 0.0 1.0)**.

[See also] **Figure 13** “Correlation of the bone coordinate system”

[Related links] 4.2.6.1.4. (translate) chunk , and 4.2.6.1.5 (handle) chunk

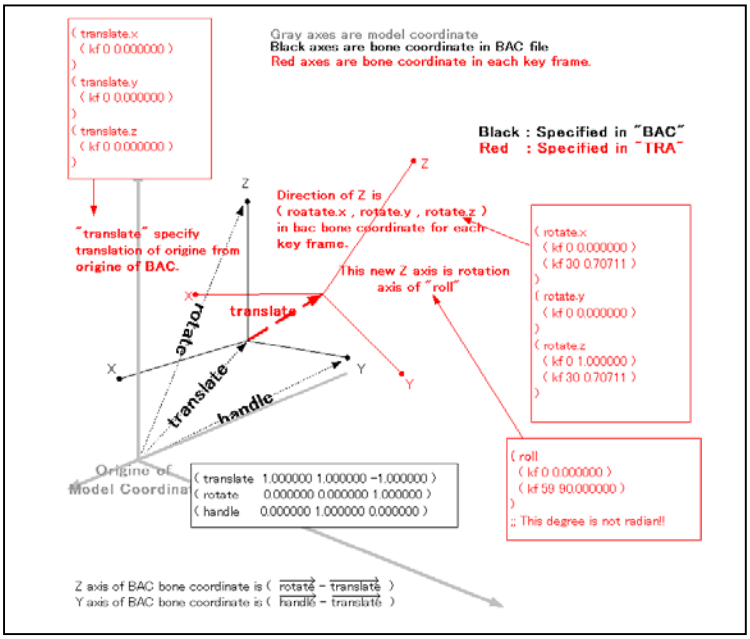


Figure 13 “Correlation of the bone coordinate system”

4.2.6.1.7. (vertexIndices [INT] [INT] … …) chunk

In the **(vertexIndices)** chunk, define the vertex ID that belongs to the target bone. This vertex ID defines the index number, which was defined in the **(coords)** chunk. This index ID is assigned from the top; it is a sequential ID starting from **0**.

If there is no vertex that belongs to the target bone, you don't need to define this; however, polygons containing a vertex that does not belong to any bones will not be displayed.

[Related links] 4.2.5.1. (coords) chunk

4.2.6.2. Additional notes on bone structure

As stated before, the bone hierarchy structure of **BAC6** is defined by the **(hasChild)** chunk and **(hasBrother)** chunk.

Explanation of the relationship between these two chunks in defining the hierarchy structure is provided here. In terms of bone hierarchy structure, you need to define the **(hasChild)** chunk as a priority, then specify each chunk definition. In other words, decide the order of defining the **(bone)** chunk, depending on whether it has a child bone or not. Please refer to **Figure 15** "Bone hierarchy" for details.

When you want to define a model data that has bone hierarchy structure as shown in **Figure 15**, specify the definition as shown in **Figure 14** "**BAC6** bone definition.

After defining the **(bone)** chunk of "**child_1**," define the **(bone)** chunk of "**child_1_1**," which is a child bone of "**child_1**." There is no child bone for the "**child_1_1**" bone; therefore, define **false** for the **(hasChild)** chunk. This completes hierarchy definition of the "**child_1**" bone. The "**child_1**" bone's **(hasBrother)** chunk is **true**; therefore, define the "**child_2**" bone in the same hierarchy.

As stated above, give priority to defining child bones, and continue defining until there are no undefined child bones. After completing child bone definitions, define the bone in the same hierarchy.

[Related links] 4.2.6.1.2. (hasChild) chunk
4.2.6.1.3. (hasBrother) chunk

```
( bone
        ( name  ″parent″ )
        ( hasChild    true )
        ( hasBrother  false )
        …
)
( bone
        ( name  ″child_1″ )
        ( hasChild    true )
        ( hasBrother  true )
        …
)
( bone
        ( name  ″child_1_1″ )
        ( hasChild    false )
        ( hasBrother  false )
        …
)
( bone
        ( name  ″child_2″ )
        ( hasChild    false )
        ( hasBrother  false )
        …
)
```

Figure 14 "**BAC6** bone definition"

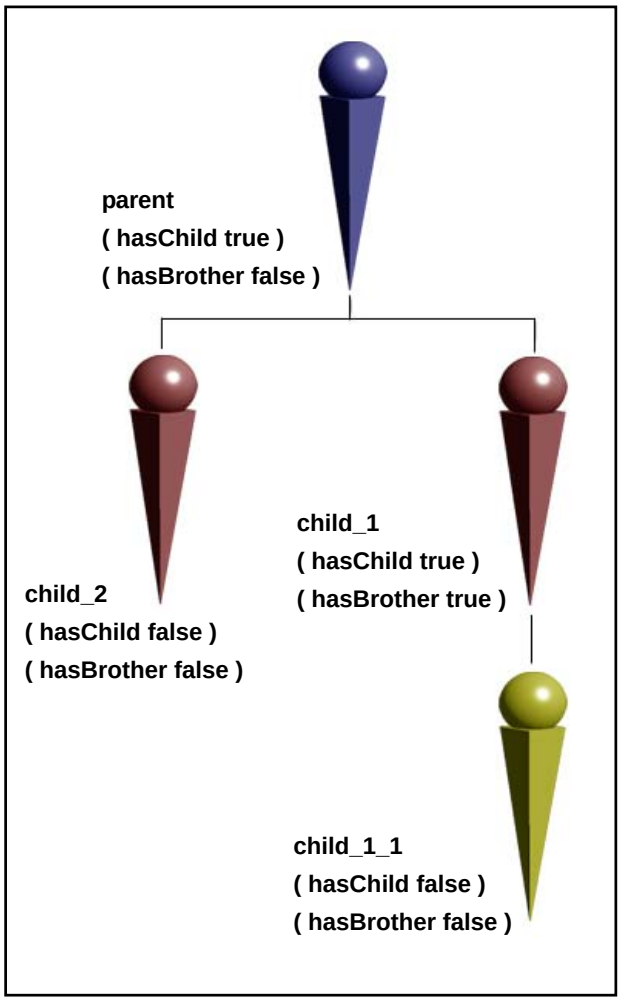


Figure 15 "Bone hierarchy"

4.2.7. (TextureCoords) chunk

In the **(TextureCoords)** chunk, define UV coordinate values of the texture specified in the model data.
When there is no texture input in the model data, you don't need to define this chunk within the **(Figure)** chunk.
UV coordinate values of **V3** range from (**0**, **0**) to (**1**, **1**). Therefore, for the values outside this range, use padding within (**0**, **0**) to (**1.0**, **1.0**).
[See also] **Figure 16** "**V3** texture coordinate system"

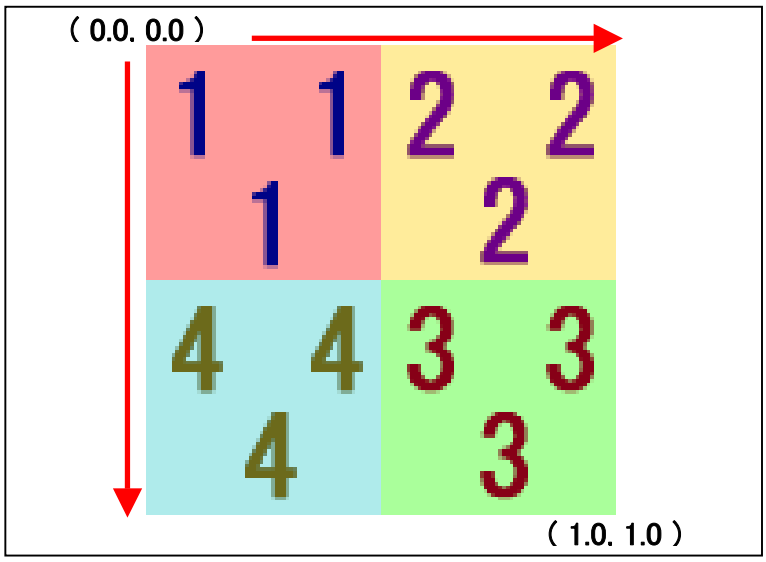


Figure 16 "**V3** texture coordinate system"

[Related links] 4.2.2. (Textures) chunk 4.2.8. (Polygons) chunk

The child chunk that defines UV coordinate values within the **(TextureCoords)** chunk is as follows:

4.2.7.1. (f2 [FLOAT] [FLOAT]) chunk

For UV coordinate values of the texture, define U coordinate value at left, then define V coordinate value.
Index ID is automatically assigned to this chunk from the top; it is a sequential ID starting from **0**.
This index ID is referred by the **(face)** chunk that belongs to the **(Polygons)** chunk.
[See also] **Figure 17** "**(TextureCoords)** chunk"

```
( TextureCoords
        ( f2 0.000 0.000 ) ; index 0
        ( f2 0.000 1.000 ) ; index 1
        ( f2 1.000 0.000 ) ; index 2
        ( f2 1.000 1.000 ) ; index 3
)
```

Figure 17 "**(TextureCoords)** chunk"

[Related links] 4.2.8.1. (face) chunk

4.2.8. (Polygons) chunk

In the **(Polygons)** chunk, define each polygon input that comprises the model data.
You must always define the **(Polygons)** chunk within the **(Figure)** chunk.

4.2.8.1. (face) chunk

In the **(face)** chunk, use the index to define the order of connecting vertices that comprise a polygon surface, the materials associated with the polygon, and the association with texture UV coordinates, etc.
When a polygon surface is comprised of three vertices, use the **(i3)** chunk. When it is comprised of four vertices, use the **(i4)** chunk. A polygon surface with more than 5 vertices is not supported.
Details of the **(i3)** chunk and the **(i4)** chunk are as follows:

[Related links] 4.2.5. (Vertices) chunk
4.2.4. (Materials) chunk
4.2.7. (TextureCoords) chunk

4.2.8.1.1. (i3) chunk

In the **(i3)** chunk, define the vertex coordinates and texture coordinates of a polygon surface comprised of three vertices.
[See also] **Figure 18** "**(i3)** chunk definition"

```
( face 0 ( i3 0 1 2 ) ( i3 0 1 2 ) )  ; Triangular polygon
         Vertex coordinate  Texture UV coordinate
```

Figure 18 "**(i3)** chunk definition"

In the vertex coordinate portion, define the index number automatically assigned to the **(pnt)** chunk within the **(coords)** chunk.
And in the texture UV coordinate index portion, define the index number automatically assigned to the **(f2)** chunk within the **(TextureCoords)** chunk.
The **(i3)** chunk can only be defined within the **(face)** chunk.

[Related links] 4.2.5.1.1. (pnt) chunk
4.2.7.1. (f2) chunk

4.2.8.1.2. (i4) chunk

In the **(i4)** chunk, define the vertex coordinates and texture coordinates of a polygon surface comprised of four vertices.
[See also] **Figure 19** "**(i4)** chunk definition"

```
( face 0 ( i4 0 1 2 3 ) ( i4 0 1 2 3 ) )  ; Quadrilateral polygon
Vertex coordinate     Texture UV coordinate
```

Figure 19 "**(i4)** chunk definition"

In the vertex coordinate portion, define the index number automatically assigned to the **(pnt)** chunk within the **(coords)** chunk.
And in the texture UV coordinate index portion, define the index number automatically assigned to the **(f2)** chunk within the **(TextureCoords)** chunk.
The **(i4)** chunk can only be defined within the **(face)** chunk.

[Related links] 4.2.5.1.1. (pnt) chunk
4.2.7.1. (f2) chunk

4.2.8.2. Additional notes on the (face) chunk definition

Examples of polygon surface definitions using the **(face)** chunk are as follows:

4.2.8.2.1. Associating the material

In order to associate the material, specify the index number automatically assigned to the **(material)** chunk within the **(Materials)** chunk, in the **(face)** chunk.
[See also] **Figure 20** "Associating the material"

```
( Polygons
    ( face 0 ( i3 0 1 2 )  ( i3 0 1 2 ))    ; Index 0 material
    ( face 1 ( i4 3 4 5 6 )( i4 3 4 5 6 )) ; Index 1 material
)    Index number of the ( material ) chunk
```

Figure 20 "Associating the material"

[Related links] 4.2.4.1. (material) chunk

4.2.8.2.2. Associating the polygon

In order to define the connecting order of the polygon, use the **(face)** chunk and the **(i3)** chunk for the polygon that has three vertices.
For a polygon that has four vertices, use the **(face)** chunk and the **(i4)** chunk to define it.
[See also] **Figure 21** "Associating the polygon"

```
( Polygons
    ( face 0 ( i3 0 1 2 )  ( i3 0 1 2 ) )   ; Triangular polygon
    ( face 1 ( i4 3 4 5 6 )( i4 3 4 5 6 ) ); Quadrilateral polygon
)        Building a polygon using the vertex index number
```

Figure 21 "Associating the polygon"

[Related links] 4.2.8.1.1. (i3) chunk
4.2.8.1.2. (i4) chunk

4.2.8.2.3. Texture coordinate input

In order to define the texture UV coordinate applied to the polygon, use the **(face)** chunk and the **(i3)** chunk for the polygon that has three vertices. For a polygon that has four vertices, use the **(face)** chunk and the **(i4)** chunk to define it.
For the color polygon without applied texture, define **-1** as index ID.
[See also] **Figure 22** "Associating the texture coordinate"

```
( Polygons
    ( face 0 ( i3 0 1 2  ) ( i3 -1 -1 -1 ) ) ; (Not using) Texture polygon
    ( face 1 ( i4 3 4 5 6 )( i4 3 4 5 6  ) ) ; (Using) Texture polygon
)
```

Figure 22 "Associating the texture coordinate"

[Related links] 4.2.8.1.1. (i3) chunk 4.2.8.1.2. (i4) chunk

4.2.9. (DynamicPolygons) chunk

Dynamic polygons are animation patterns that change texture UV coordinates or materials for the polygons defined in the **(face)** chunk.
Prior to defining the patterns for dynamic polygons that change texture UV coordinates, you must always define the **(Textures)** and **(TextureCoords)** chunks. This is because the **(face)** chunk refers to the texture coordinate index; the **(face)** chunk belongs to the **(group)** chunk, which is described later.
Please refer to the following figures for detail: **Figure 23** "**Example 1— (DynamicPolygons)** chunk definition," and **Figure 24** "**Example 2— (DynamicPolygons)** chunk definition."
Figure 23 shows a definition example of dynamic polygons' patterns that change textures.

```
( DynamicPolygons
    ( group      ; Index 0
        ( name "pattern0" )
        ( face 0 ( i4 3 2 0 1 )( i4 3 4 1 0 ) )
    )

    ( group      ; Index 1
        ( name "pattern1" )
        ( face 0 ( i4 3 2 0 1 )( i4 6 7 4 3 ) )
    )
)
```

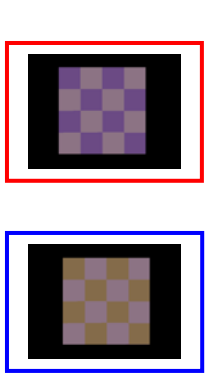


Figure 23"**Example 1— (DynamicPolygons)** chunk definition"

Figure 24 shows a definition example of dynamic polygons that change materials.

```
( DynamicPolygons
    ( group      ; Index 0
        ( name "pattern0" )
        ( face 0 ( i4 3 2 0 1 )( i4 -1 -1 -1 -1 ) )
    )

    ( group      ; Index 1
        ( name "pattern1" )
        ( face 1 ( i4 3 2 0 1 )( i4 -1 -1 -1 -1 ) )
    )
)
```

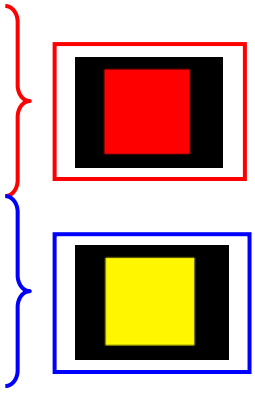


Figure 24"**Example 2— (DynamicPolygons)** chunk definition"

[Related links] 4.2.1. (name) chunk
4.2.8.1. (face) chunk

For the data that does not use dynamic polygons, you don't have to define the **(DynamicPolygons**) chunk within the **(Figure)** chunk.
Details of the **(name)** chunk and the **(face)** chunk to be defined in the **(DynamicPolygons)** chunk are as follows:

4.2.9.1. (group) chunk

Define the animation pattern of dynamic polygons in the **(group)** chunk.
Index ID is assigned to this chunk from the top; it is a sequential ID starting from **0**. The number of this **(group)** chunk index equals to the number of defined animation patterns for dynamic polygons. This index ID is referred by the **(kgf)** chunk described in the separate document, "**TRA File Ver.4.0 Data Format Specification**."
Please refer to the **Figure 23** "**Example 1— (DynamicPolygons)** chunk definition," and **Figure 24** "**Example 2— (DynamicPolygons)** chunk definition" for details.

4.2.9.1.1. (name) chunk

Specify arbitrary name for dynamic polygons' animation pattern in the **(name)** chunk.
Please refer to the **Figure 23** "**Example 1— (DynamicPolygons)** chunk definition," and **Figure 24** "**Example 2— (DynamicPolygons)** chunk definition" for details.

[Related links] 4.2.1. (name) chunk

4.2.9.1.2. (face) chunk

Define the texture UV coordinates and materials for each pattern of dynamic polygons in the **(face)** chunk within the **(group)** chunk.
Please refer to the **Figure 23** "**Example 1— (DynamicPolygons)** chunk definition," and **Figure 24** "**Example 2— (DynamicPolygons)** chunk definition" for details.

[Related links] 4.2.8.1. (face) chunk

5. Sample code

5.1. BAC6 sample 1

The following code is a model data output when the texture was applied for **BAC6** file.

[Sample01.bac]

```
;BAC

( Head
	( bacVersion 6.0 )
)
( Figure
	( name "TexturePolygonSample" )
	( Textures
		 ( i2 256 256 )
	)
	( Colors
		( f3 0.500 0.500 0.500 )
	)
	( Materials
		( material
			( blendMode   normal )
			( doubleFace  true )
			( transparent false )
			( lighting    true )
			( textureIndex 0 )
			( colorIndex  -1 )
			( specular    1.000 )
		)
	)
	( Vertices
		( coords
			( pnt -1.500 0.000 0.000 )
			( pnt -1.500 3.000 0.000 )
			( pnt 1.500 0.000 0.000 )
			( pnt 1.500 3.000 0.000 )
		)
		( normals
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
		)
	)
```

```
	( Bones
		( bone
			( name		"bone" )
			( hasChild	false )
			( hasBrother	false )
			( translate	0.000 0.000 0.000 )
			( rotate	0.000 0.000 1.000 )
			( handle	0.000 1.000 0.000 )
			( vertexIndices
				0 1 2 3
			)
		)
	)
	( TextureCoords
		( f2 0.000 0.000 )
		( f2 0.000 1.000 )
		( f2 1.000 0.000 )
		( f2 1.000 1.000 )
	)
	 ( Polygons
		( face 0 ( i4 3 2 0 1 ) ( i4 2 3 1 0 ) )
	)
)
```

5.2. BAC6 sample 2

The following code is a model data output when dynamic polygons were applied for **BAC6** file.

[Sample02.bac]

```
;BAC

( Head
	( bacVersion 6.0 )
)
( Figure
	( name "DynamicPolygonSample" )
	( Textures
		( i2 128 128 )
	)
	( Colors
		 ( f3 1.000 0.500 0.500 )
	)
	( Materials
		( material
			( blendMode   normal )
			( doubleFace  true )
			( transparent false )
			( lighting    true )
			( textureIndex 0 )
			( colorIndex  -1 )
			( specular    1.000 )
		)
		( material
			( blendMode    normal )
			( doubleFace   true )
			( transparent  false )
			( lighting     true )
			( textureIndex -1 )
			( colorIndex   0 )
			( specular     1.000 )
		)
	)
	( Vertices
		( coords
			( pnt -400.000 -200.000 0.000 )
			( pnt -400.000 200.000 -0.000 )
			( pnt 0.000 -200.000 0.000 )
			( pnt 0.000 200.000 -0.000 )
			( pnt 400.000 -200.000 0.000 )
			( pnt 400.000 200.000 -0.000 )
		)
```

```
		( normals
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
			( vct 0.000 0.000 1.000 )
		)
	)
	( Bones
		( bone
			( name        "bone" )
			( hasChild    false )
			( hasBrother  false )
			( translate   0.000 0.000 0.000 )
			( rotate      0.000 0.000 100.000 )
			( handle      0.000 100.000 0.000 )
			( vertexIndices
				0 1 2 3 4 5
			)
		)
	)
	( TextureCoords
		( f2 0.000 0.000 )
		( f2 0.000 0.500 )
		( f2 0.000 1.000 )
		( f2 0.500 0.000 )
		( f2 0.500 0.500 )
		( f2 0.500 1.000 )
		( f2 1.000 0.000 )
		( f2 1.000 0.500 )
		( f2 1.000 1.000 )
	)
	( Polygons
		( face 1 ( i4 3 2 0 1 ) ( i4 –1 –1 –1 -1 ) )
	)
	( DynamicPolygons
		( group
			( name "pattern0" )
			( face 0 ( i4 5 4 2 3 ) ( i4 3 4 1 0 ) )
		)
		( group
			( name "pattern1" )
			( face 0 ( i4 5 4 2 3 ) ( i4 6 7 4 3 ) )
		)
	)
)
```

BAC File Ver.6.0 Data Format Specification

MascotCapsule V3

English version

Version: 2.1

Publication date: August 24, 2007

Publisher: HI CORPORATION
Meguro Higashiyama Bldg. 5F
1-4-4 Higashiyama
Meguro-ku, Tokyo 153-0043 Japan
http://www.hicorp.co.jp/