X3D/VRML パーサー説明書

2008/11/26

概要

この文書は、国立情報学研究所の「SIGVerse（社会的知能発生シミュレータ）」の研究の一部である X3D/VRML パーサープログラム（以下、X3DParser と書きます）に関する説明書です.

X3DParser は X3D/VRML で記述された形状ファイルをパースすることを目的としています.
X3DParser により、これらの形状ファイルを読み込み、内部構造を抽出することができます.
また、複雑な物体形状から、物理シミュレーションを目的とした単純な形状を生成することもできます.

X3DParser はライブラリの形を取っていますので、他のプログラムに自由に組み込んで使用することができます.

パース対象ファイル

X3DParser でパースできるファイルは以下の通りです.

X3D (VRML 3.0) または VRML97 (VRML 2.0)に準拠したファイル.（テキストファイルのみ）

なお、X3D/VRML 仕様のうち、以下の機能には対応していません.

(1) ExternProto
(2) Inline
(3) サウンド、ナビゲーション、センサー、スクリプト機能等

必要環境

X3DParser の実行には以下の環境が必要です.

Microsoft Windows XP/Vista (32bit)

```
JRE 4.1.2
Xj3D 1.0
```

実行だけでなく、コンパイルも行う場合には、さらに以下の環境が必要です.

```
JDK 1.6.0
eclipse 3.4.0
Visual C++ 2005
```

（注 1）動作確認は上記のバージョン番号にて行っています. 上位バージョンを使う際には互換性の面で問題が発生する可能性があるので注意してください.

（注 2）JDK をインストールする場合、JRE のインストールは必要ありません（JDK に含まれています）

	実行のみ	コンパイル&実行
OS	Microsoft Windows XP/Vista (32bit)	
java 環境	JRE 4.1.2	JDK 1.6.0
ライブラリ	Xj3D 1.0	
コンパイラ	必要なし	eclipse 3.4.0 Visual C++ 2005

ディレクトリ構成

X3DParser のディレクトリ構成を示します

```
├──parser
│   ├──cpp
│   │   ├──X3DParser
│   │   └──X3DParserTest
│   │       └──Debug
│   ├──doc
│   └──java
│       └──X3DParser
│           ├──bin
│           └──src
├──shapes
│   ├──VRML
│   └──X3DSamples
├──tools
    └──Xj3D
```

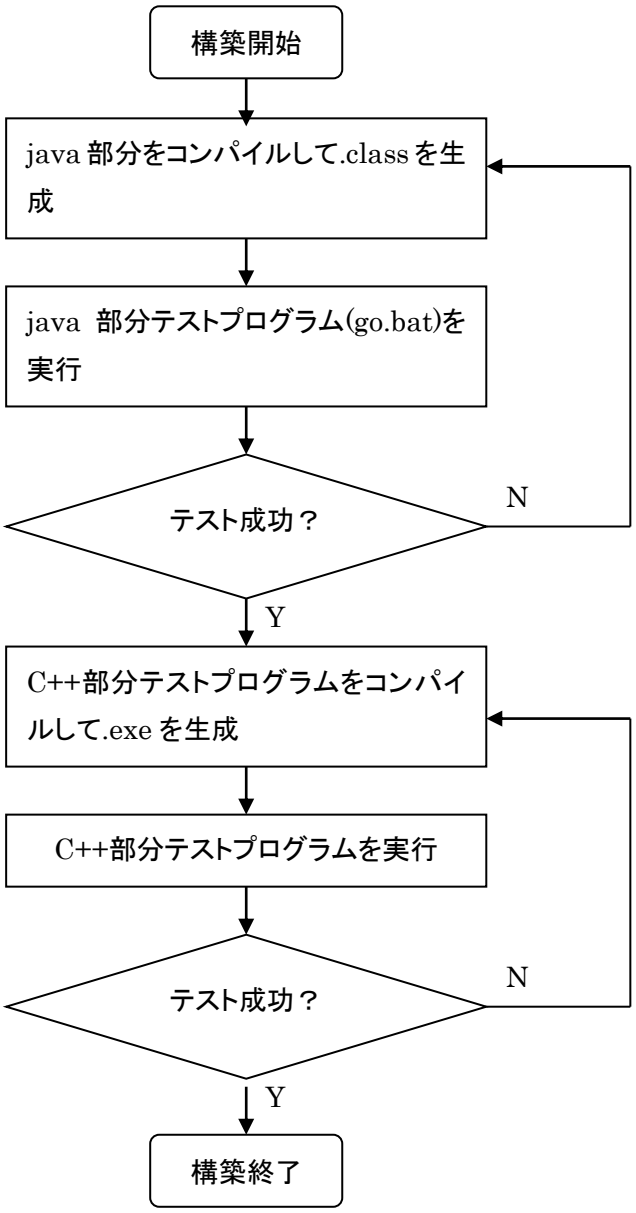
ディレクトリ	内容
parser	X3DParser トップディレクトリ
parser/cpp/X3DParser	X3DParser C++部分ソースコード
parser/cpp/X3DParserTest	X3DParser テストプログラムソースコード
parser/cpp/X3DParserTest/Debug	X3DParser テストプログラム実行形式
parser/doc	ドキュメント
parser/java/X3DParser/bin	X3DParser java クラスファイル
parser/java/X3DParser/src	X3DParser java 部分ソースコード
shapes/VRML	VRML 形状ファイル
shapes/X3DSamples	X3D 形状ファイル
tools/Xj3D	Xj3D ライブラリインストールパッケージ

コンパイル&テスト実行方法

X3DParser は java 部分と C++部分から構成されています.

java 部分は独立していますので、コンパイルは java 部分から行ってください.
C++部分は java 部分に依存していますので、単独でコンパイルすることはできません.

構築フローは以下のようになります.



java 部分コンパイル前の準備

コンパイル前に、Xj3D ライブラリをインストールしておく必要があります.

tools/Xj3D/Xj3D-1-0-windows-full.exe を実行し、Xj3D ライブラリを適当なディレクトリにインストールしてください.
以下では、C:/Xj3D にインストールしたものとして説明します.

コンパイルには Eclipse を使用します.
Eclipse の配布ウェブサイトからダウンロードし、インストールしてください.
（詳しくは Eclipse のウェブサイト(http://www.eclipse.org/)を参照してください）

コンパイルをする前に、Eclipse に以下の設定が必要です.

(1) Xj3D の jar ファイルへの参照設定

C:/Xj3D/jars ディレクトリにある、xj3d-*.jar 全部と、uri.jar を設定してください.

具体的なファイル名は以下の通りです.

```
xj3d-all.jar
xj3d-cefx3d.jar
xj3d-common.jar
xj3d-config.jar
xj3d-core.jar
xj3d-eai.jar
xj3d-ecmascript.jar
xj3d-external-sai.jar
xj3d-images.jar
xj3d-j3d.jar
xj3d-java-sai.jar
xj3d-jaxp.jar
xj3d-jsai.jar
xj3d-net.jar
xj3d-norender.jar
xj3d-ogl.jar
```

```
xj3d-parser.jar
xj3d-render.jar
xj3d-runtime.jar
xj3d-sai.jar
xj3d-sav.jar
xj3d-script-base.jar
xj3d-xml-util.jar
xj3d-xml.jar
uri.jar
```

設定方法に関してはEclipseのマニュアル等を参照してください

(2) クラスパスの設定

C:/Xj3D/jarsディレクトリをクラスパスに設定してください.

java部分コンパイル

java部分のソースは、parser/java/X3DParser/srcにある以下の.javaファイルです.

```
MyWorldRoot.java
MyX3DUtil.java
X3DParseEventHandler.java
X3DParser.java
X3DParserTest.java
```

Eclipseでこのディレクトリにあるソースをimportしたプロジェクトを作り、コンパイルしてください. 成功するとbinディレクトリに.classファイルが生成されます.

java部分テスト実行用バッチファイル(go.bat)の修正

parser/java/X3DParser/binにあるgo.batは、引数に与えたjavaクラスファイルを起動するバッチファイルです.

go.bat の中で、Xj3D ライブラリの場所を指定している箇所がありますので、インストールした Xj3D ライブラリの場所に合わせて書き換えてください.（下記の赤字の部分）

```
@echo off

java     -Xmx450M   -XX:MaxDirectMemorySize=350M   -Xbootclasspath/p:./bin
-Dsun.java2d.noddraw=true                    -Djava.library.path=/Xj3D/bin
-classpath .;/Xj3D/jars/xj3d-all.jar %1 %2 %3 %4 %5 %6
```

go.bat の実行

コマンドプロンプトで以下のように打ち込むと、引数に与えた X3D(VRML)ファイルをパースします.

```
> go X3DParserTest 0 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-1.x3d
```

（注） go X3DParserTest.class 0 list2-1.x3d と打ち込むと正常に実行できませんので注意してください
(.class は不要です)

以下のようなパース結果が表示されれば成功です.（表示は長いので途中省略しています）

```
WorldRoot
  Background
    [ groundAngle ] :
    [ groundColor ] :
    [ skyAngle ] :
    [ skyColor ] : 1.0 1.0 1.0
...
  [ geometry ]
    Box
      [ solid ] true
      [ size ] 4.0 3.0 2.0
  [ bboxSize ] -1.0 -1.0 -1.0
  [ bboxCenter ] 0.0 0.0 0.0
```

go.bat の実行がうまくいかないとき

go.bat でのテストプログラムが正常に実行できない場合は、以下を確認してみてください

JRE（または JDK） 1.4.2 以降はインストールされ、実行できる状態になっていますか？

確認するには、例えば java -version と打ち込んでみます.
以下のような表示がされれば、この点は問題ないでしょう.

```
>java -version
java version "1.6.0_07"
Java(TM) SE Runtime Environment (build 1.6.0_07-b06)
Java HotSpot(TM) Client VM (build 10.0-b23, mixed mode, sharing)
```

go.bat のパス指定は正しいですか？

go.bat 内のパス指定をもう一度確認してみてください.

インターネットに接続できる環境にいますか？

インターネットに接続できない場合、以下のような表示がでることがあります.

```
>go X3DParserTest 0 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-1.x3d
Warning: The file uses an unknown DTD.  This content is not validated against
th
e X3D standard DTD.
java.net.UnknownHostException: www.web3d.org
       at java.net.PlainSocketImpl.connect(Unknown Source)
       at java.net.Socket.connect(Unknown Source)
       at java.net.Socket.connect(Unknown Source)
       at sun.net.NetworkClient.doConnect(Unknown Source)
       at sun.net.www.http.HttpClient.openServer(Unknown Source)
       at sun.net.www.http.HttpClient.openServer(Unknown Source)
       at sun.net.www.http.HttpClient.<init>(Unknown Source)
```

```
        at sun.net.www.http.HttpClient.New(Unknown Source)
        at sun.net.www.http.HttpClient.New(Unknown Source)
...
```

インターネットに接続できない場合、上記のようなエラーメッセージが出ます.

Xj3D ライブラリは、VRML のパース処理中、www.web3d.org にアクセスします.
必ずインターネットに接続できる環境で実行してください.

C++部分のソースコード

C++部分のソースコードは parser/cpp/X3DParser にある.h/.cpp ファイルです.

X3DParser を使う C++プログラムをコンパイルする場合や、ライブラリを作る場合は、このディレクトリのファイルをプロジェクトに含めてください.

C++部分テストプログラムのコンパイル

parser/cpp/X3DParserTest に X3DParserTest.sln ファイルがありますので、Visual C++ 2005 でこのファイルを開いてビルドしてください.
（ただし、インクルードパス、ライブラリパスの設定は、環境に合わせて変える必要があります）

もし最初からプロジェクトを作ってコンパイルしたい場合は、以下のようにしてください.

(1) Visual C++ 2005 で「Win32 コンソール」プロジェクトを作る

ソースファイルとして

parser/cpp/X3DParser
parser/cpp/X3DParserTest

にあるすべての.h/.cpp ファイルをプロジェクトに追加します.

(2) インクルードパスの設定

インクルードパスを以下に設定します.

```
C:¥Java¥jdk1.6.0_07¥include¥win32;C:¥Java¥jdk1.6.0_07¥include;..¥X3DParser
```

（注）上記は jdk1.6.0_07 を C:¥Java¥jdk1.6.0_07 にインストールしてある場合です.
（jdk のインストールディレクトリに合わせて変更してください）

(3) ライブラリパスの設定

ライブラリパスを以下に設定します.

```
C:¥Java¥jdk1.6.0_07¥lib
```

（注）上記は jdk1.6.0_07 を C:¥Java¥jdk1.6.0_07 にインストールしてある場合です.
（jdk のインストールディレクトリに合わせて変更してください）

(4) 参照するライブラリの設定

```
jvm.lib
```

jvm.lib は jvm.dll (JNI 関連 DLL) のインポートライブラリです.

(5) ビルド

コンパイルが成功すると X3DParserTest.exe が parser/cpp/X3DParserTest/Debug ディレクトリに生成されます.

C++部分テストプログラムの実行前の設定

X3DParserTest.exe の実行前にいくつかの設定が必要です.

(1) パスの設定

環境変数 PATH に jvm.dll へのパスを追加してください.

jvm.dll は JNI 用の DLL で、JDK または JRE をインストールするとシステムにインストールされます. jdk1.6.0_07 を C:/jdk1.6.0_07 にインストールした場合、

/jdk1.6.0_07/jre/bin/client

にあります.

(2) クラスパスの設定

parser/cpp/X3DParserTest に X3DParser.cfg というファイルがあります.

ここに、java のクラスパスの設定がありますので、環境に合わせて書き換えてください.
（以下の赤字部分）

```
#
# Library Path
#
java.library.path=/Xj3D/bin

#
# Class Path
#
java.class.path=.;parser/java/X3DParser/bin;/Xj3D/jars/xj3d-all.jar
```

以上で設定は完了です.

C++部分テストプログラムの実行

テストプログラムは X3DParserTest.exe です.
書式は以下の通りです.

```
X3DParserTest.exe testNo VRMLFile [args] [-MFMax nMFMax] [-log logfile]
```

オプション引数：

testNo
テスト番号です.
どのようなテスト項目があるかは、単に X3DParserTest を引数なしで実行すると表示されます.

VRMLFile
パース対象の X3D/VRML ファイル名を与えます.

args
省略可能な引数です. テストによっては、追加の引数が必要なものもありますので、その場合は VRML ファイル名に続けて指定してください.

-MFMax nMFMax
省略可能な引数です. MFInt32, MFDouble 等の MF フィールドを、ログに表示する要素数を制限

します.
MF フィールドに含まれる要素の数が多く、表示が見にくくなる場合に指定してください.
（デフォルトでは、すべて表示します）

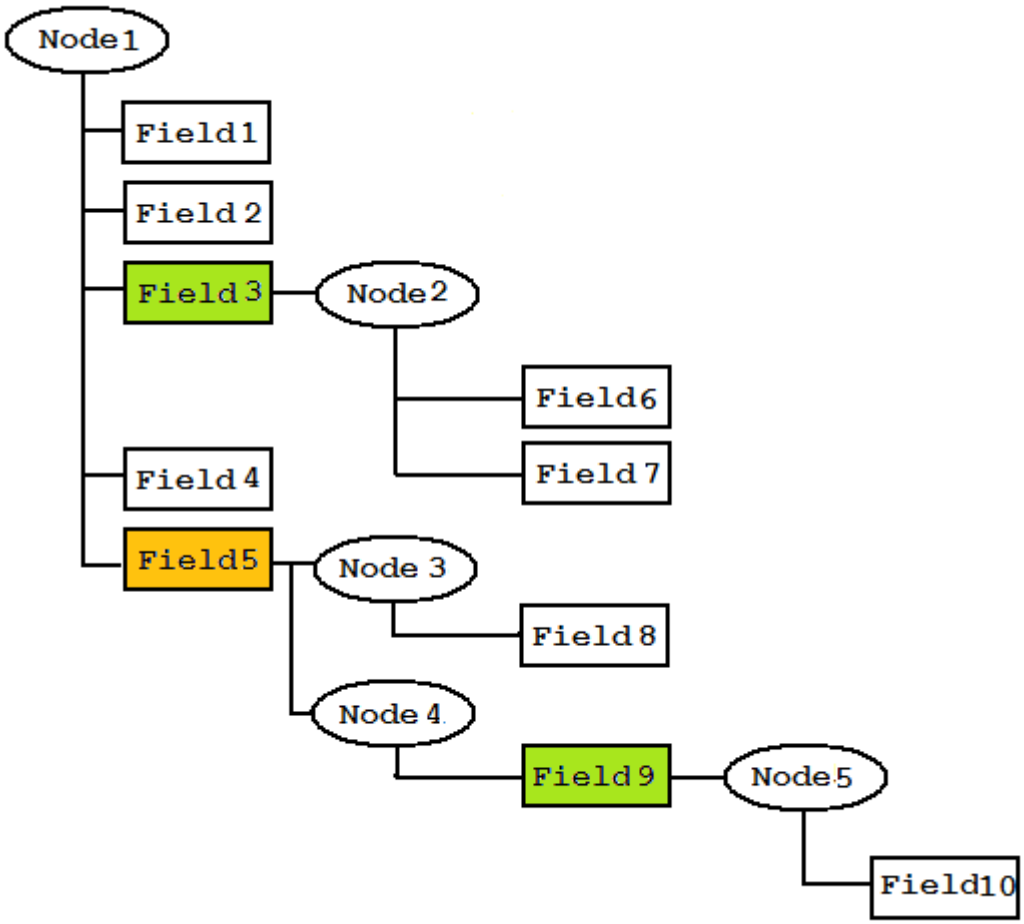
-log logfile
省略可能な引数です．ログの出力先ファイルを指定します.

なお、実行例がこの文書の最後の「サンプルファイル一覧」にありますので、そちらも合わせて参照してください.

X3D/VRML の概要

ノードとフィールド

ノードとフィールドの関係を説明します.
以下の図で、楕円はノード、長方形はフィールドを表しています.



(1) ノードは、0 個以上のフィールドを持ちます.
(2) フィールドは値を持ちます.
(3) フィールドの値の型には、2 種類あります. 一つは単純型フィールドで、もう一つはノード型フィールドです.
(4) 単純型フィールドの値は、整数、実数、文字列、ベクトル、またはそれらの配列、です.
(5) ノード型フィールドの値は、ノードです.

(6) ノード型フィールドには SFNode 型と MFNode 型があります.
(6-1) SFNode 型フィールドは、ノードを 1 個だけ持てます（上図の黄緑色部分）
(6-2) MFNode 型フィールドは、ノードを複数個持てます（上図のオレンジ色部分）

Node1 は 5 つのフィールド(Field1～5)を持っています.
Node2 は 2 つのフィールド(Field6, 7)を持っています.
Node3 は 1 つのフィールド(Field8)を持っています.
Node4 は 1 つのフィールド(Field9)を持っています.
Node5 は 1 つのフィールド(Field10)を持っています.
Field1, 2, 4, 6, 7, 8, 10 は単純型フィールドです.
Field3, 9 はノードを 1 つ持つフィールド(SFNode 型フィールド)です.
Field3 の値は Node2 です.
Field9 の値は Node5 です.
Field5 はノードを複数個持つフィールド(MFNode 型フィールド）です.
Field5 の値は Node3, 4 です.

単純型フィールドの持てる値の型は、以下の通りです.

SFBool	1 個の論理値
SFInt32	1 個の整数
SFFloat	1 個の短精度実数
SFDouble	1 個の倍精度実数
SFTime	1 個の時間
SFVec2f	1 個の 2 次元短精度ベクトル
SFVec3f	1 個の 3 次元短精度ベクトル
SFVec4f	1 個の 4 次元短精度ベクトル
SFVec2d	1 個の 2 次元倍精度ベクトル
SFVec3d	1 個の 3 次元倍精度ベクトル
SFVec4d	1 個の 4 次元倍精度ベクトル
SFRotation	1 個の回転
SFColor	1 個の色情報
SFString	1 個の文字列
MFBool	n 個の論理値 (n>=0)
MFInt32	n 個の整数(n>=0)
MFFloat	n 個の短精度実数(n>=0)
MFDouble	n 個の倍精度実数(n>=0)

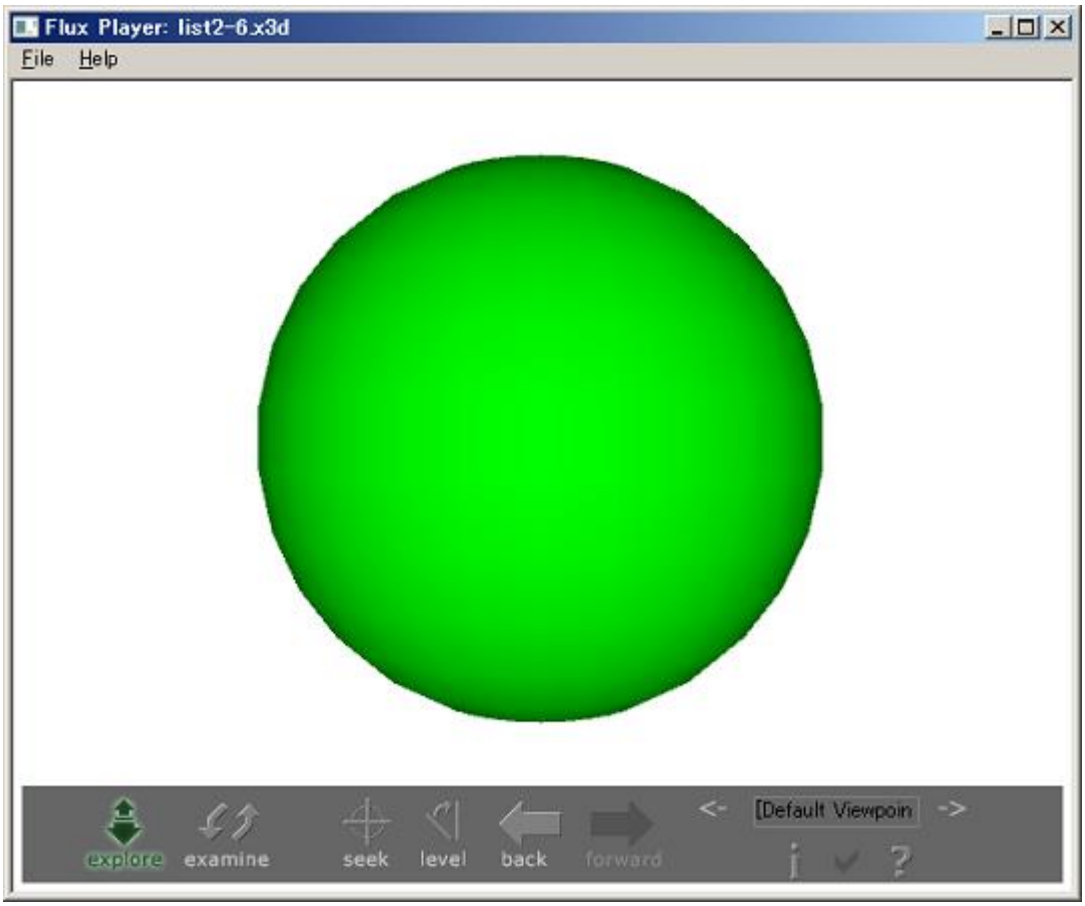
MFVec2f	n 個の 2 次元短精度ベクトル(n>=0)
MFVec3f	n 個の 3 次元短精度ベクトル (n>=0)
MFVec4f	n 個の 4 次元短精度ベクトル (n>=0)
MFVec2d	n 個の 2 次元倍精度ベクトル (n>=0)
MFVec3d	n 個の 3 次元倍精度ベクトル (n>=0)
MFVec4d	n 個の 4 次元倍精度ベクトル (n>=0)
MFColor	n 個の色情報(n>=0)
MFString	n 個の文字列(n>=0)

ノード型フィールドの持てる値の型は、以下の通りです.

SFNode	1 個のノード
MFNode	n 個のノード (n>=0)

X3D/VRML の例

以下の図のような緑色の球を VRML/X3D で表現してみます.



VRML の場合

慣例的にノード名は先頭大文字で表します(Scene, Transform, Material 等)
フィールド名は先頭小文字で表します（translation, rotation, width, height 等）

```
#VRML V2.0 utf8 CosmoWorlds V1.0
Background {
	skyColor 1 1 1
}
Shape {
	geometry Sphere {
		radius 2
	}
	appearance Appearance {
		material Material {
			diffuseColor 0 1 0
		}
	}
}
```

Shape ノードは geometry フィールドと appearance フィールドを持っています.

geometry フィールドの値は Sphere ノードです.

Sphere ノードは radius フィールドを持っています.
radius フィールドの値は 2 です.

appearance フィールドの値は Appearance ノードです.

Appearance ノードは material フィールドを持っています.
material フィールドの値は Material ノードです.

Material ノードは diffuseColor フィールドを持っています.
diffuseColor フィールドの値は 0 1 0 です.

X3D の場合

VRML と同じく、ノード名は先頭大文字で、フィールド名は先頭小文字で表します.

(1) ノードは XML タグで表します.
(2) 単純型フィールドは、XML タグ内の属性で表します.
(3) ノード型フィールドは、XML タグの子として表します. このとき、VRML と異なり、フィールド名の指定はできません.（自動的に判断される）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.2//EN"
  "http://www.web3d.org/specifications/x3d-3.2.dtd">
<X3D profile='Immersive'>
  <head><meta name='filename' content='list2-6.x3d'/></head>
  <Scene>
    <Background skyColor='1 1 1'/>
    <!-- 緑色の球体 -->
    <Shape>
      <Sphere radius='2'/>
      <Appearance>
        <Material diffuseColor='0 1 0'/>
      </Appearance>
    </Shape>
  </Scene>
</X3D>
```

X3DParser でのノードとフィールドの扱い

汎用ノード (CX3DNode)

ノードを表す基本クラスです．どんなノードもこのクラスで表現できます．

専用ノード

使用頻度の高い（と思われる）ノードに関しては、専用のクラスがあります．
これらはすべて、CX3DNode の派生クラスです．

```
CX3DBoxNode
CX3DColorNode
CX3DConeNode
CX3DCoordinateNode
CX3DCylinderNode
CX3DDirectionalLightNode
CX3DGroupNode
CX3DHAnimDisplacerNode
CX3DHAnimHumanoidNode
CX3DHAnimJointNode
CX3DHAnimSegmentNode
CX3DHAnimSiteNode
CX3DImageTextureNode
CX3DIndexedFaceSetNode
CX3DIndexedLineSetNode
CX3DMaterialNode
CX3DNormalNode
CX3DOpenHRPHumanoidNode
CX3DOpenHRPJointNode
CX3DOpenHRPSegmentNode
CX3DPointLightNode
CX3DShapeNode
CX3DSphereNode
CX3DSpotLightNode
```

```
CX3DTextureCoordinateNode
CX3DTextureTransformNode
CX3DTransformNode
CX3DViewpointNode
```

通常の使用に関しては、これらの専用ノードを使うことになるでしょう．万が一これらの専用ノードクラスが存在しないノードを操作する必要が生じた場合は、CX3DNode を使って操作を行ってください．

CX3DTransform

専用ノードの使い方を、CX3DTransformNode を例にして説明します．
（他の専用ノードも基本的には同じですのでそれらについては省略します）

```
SFVec3f *CX3DTransformNode::getTranslation();
```

Transform ノードの translation フィールドの値を返します．
translation フィールドは SFVec3f 型です．
このフィールドは、平行移動成分を表す 3 つの実数値を保持しています．

```
SFRotation *CX3DTransformNode::getRotation();
```

Transform ノードの rotation フィールドの値を返します．
rotation フィールドは SFRotation 型です．
このフィールドは、回転を表す 4 つの実数を保持しています．

```
SFVec3f *CX3DTransformNode::getCenter();
```

Transform ノードの center フィールドの値を返します．
center フィールドは SFVec3f 型です．
このフィールドは移動、回転の中心となる座標を保持しています．

```
SFVec3f *CX3DTransformNode::getScale();
```

Transform ノードの scale フィールドの値を返します．
scale フィールドは SFVec3f 型です．

このフィールドは拡大・縮小のスケールを保持しています.

```
SFRotation *CX3DTransformNode::getScaleOrientation();
```

Transform ノードの scaleOrientation フィールドの値を返します.
scaleOrientation フィールドは SFRotation 型です.

```
MFNode *CX3DTransformNode::getChildren();
```

Transform ノードの children フィールドの値を返します.
children フィールドは MFNode 型です.
このフィールドは、任意個の子ノードを保持します.

（重要な注意）
フィールドへのアクセスメソッドの返り値のポインタは、ノード側に所有権があります.
返り値を delete してはいけません.

（良い例）

```
CX3DTransformNode *trans = ...
SFVec3f *translation = trans->getTranslation();
if (translation)
{
	// ... translation を使う ...
}
```

（悪い例）

```
CX3DTransformNode *trans = ...
SFVec3f *translation = trans->getTranslation();
if (translation)
{
	// ... translation を使う ...
	delete translation;		// *** エラー！これは不要！ ***
}
```

フィールド

フィールドの型に応じた以下のクラスがあります.
これらはすべて、CX3DField の派生クラスです

宣言および定義は CX3DField.h/cpp にあります

```
SFBool
SFInt32
SFFloat
SFDouble
SFTime
SFVec2f
SFVec3f
SFVec4f
SFVec2d
SFVec3d
SFVec4d
SFRotation
SFColor
SFString
SFNode
MFBool
MFInt32
MFFloat
MFDouble
MFVec2f
MFVec3f
MFVec4f
MFVec2d
MFVec3d
MFVec4d
MFColor
MFString
MFNode
```

これらのフィールドクラスの重要メソッドを説明します.

SFBool

```
bool SFBool::getValue();
```

このフィールドの持つ論理値を返します.

SFInt32

```
int SFInt32::getValue();
```

このフィールドの持つ符号付 32bit 整数を返します.

SFFloat

```
float SFFloat::getValue();
```

このフィールドの持つ単精度実数を返します.

SFDouble

```
double SFDouble::getValue();
```

このフィールドの持つ倍精度実数を返します.

SFTime

```
double SFTime::getValue();
```

このフィールドの持つ時間値を倍精度実数で返します.

SFVec2f

```
float SFVec2f::x();
float SFVec2f::y();
```

2 次元ベクトルフィールドの各要素値を単精度実数で返します.

SFVec3f

```
float SFVec3f::x();
float SFVec3f::y();
float SFVec3f::z();
```

3 次元ベクトルフィールドの各要素値を単精度実数で返します.

SFVec4f

```
float SFVec4f::x();
float SFVec4f::y();
float SFVec4f::z();
float SFVec4f::w();
```

4 次元ベクトルフィールドの各要素値を単精度実数で返します.

SFVec2d

```
double SFVec2d::x();
double SFVec2d::y();
```

2 次元ベクトルフィールドの各要素値を倍精度実数で返します.

SFVec3d

```
double SFVec3d::x();
double SFVec3d::y();
double SFVec3d::z();
```

3 次元ベクトルフィールドの各要素値を倍精度実数で返します.

SFVec4d

```
double SFVec4d::x();
double SFVec4d::y();
double SFVec4d::z();
double SFVec4d::w();
```

4 次元ベクトルフィールドの各要素値を倍精度実数で返します.

SFRotation

```
float SFRotation::x();
float SFRotation::y();
float SFRotation::z();
float SFRotation::rot();
```

回転フィールドの各要素値を単精度実数で返します.

SFColor

```
float SFColor::r();
float SFColor::g();
float SFColor::b();
```

色情報を赤成分(r), 緑成分(g), 青成分(b)ごとに返します.

SFString

```
const char *SFString::getValue();
```

フィールドの持つ文字列へのポインタを返します.
（返り値のポインタを解放してはいけません）

SFNode

```
CX3DNode *SFNode::getNode();
```

このフィールドの持つノードへのポインタを返します.
ただし返り値のポインタ（の指すノード）の所有権は SFNode が持っていますので、
返り値を delete してはいけません.

```
CX3DNode *SFNode::releaseNode();
```

getNode()と同じく、このフィールドの持つノードへのポインタを返しますが、同時にノードの所有権を放棄します. すなわち、呼び出し側は、返り値を delete しなければなりません.

MFBool

```
int MFBool::count();
```

このフィールドの持つ論理値の個数を返します.

```
bool MFBool::getValue(int i);
```

このフィールドの持つ i 番目の論理値の値を返します.
i は 0 以上、count()未満です.

MFInt32

```
int MFInt32::count();
```

このフィールドの持つ整数の個数を返します.

```
int MFInt32::getValue(int i);
```

このフィールドの持つ i 番目の整数の値を返します.
i は 0 以上、count()未満です.

MFFloat

```
int MFFloat::count();
```

このフィールドの持つ単精度実数の個数を返します.

```
float MFFloat::getValue(int i);
```

このフィールドの持つ i 番目の単精度実数の値を返します.
i は 0 以上、count()未満です.

MFDouble

```
int MFDouble::count();
```

このフィールドの持つ倍精度実数の個数を返します.

```
double MFDouble::getValue(int i);
```

このフィールドの持つ i 番目の倍精度実数の値を返します.
i は 0 以上、count()未満です.

MFVec2f

```
int MFVec2f::count();
```

このフィールドの持つ単精度 2 次元ベクトルの個数を返します.

```
SFVec2f MFVec2f::getValue(int i);
```

このフィールドの持つ i 番目の単精度 2 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFVec3f

```
int MFVec3f::count();
```

このフィールドの持つ単精度 3 次元ベクトルの個数を返します.

```
SFVec3f MFVec3f::getValue(int i);
```

このフィールドの持つ i 番目の単精度 3 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFVec4f

```
int MFVec4f::count();
```

このフィールドの持つ単精度 4 次元ベクトルの個数を返します.

```
SFVec4f MFVec4f::getValue(int i);
```

このフィールドの持つ i 番目の単精度 4 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFVec2d

```
int MFVec2d::count();
```

このフィールドの持つ倍精度 2 次元ベクトルの個数を返します.

```
SFVec2d MFVec2d::getValue(int i);
```

このフィールドの持つ i 番目の倍精度 2 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFVec3d

```
int MFVec3d::count();
```

このフィールドの持つ倍精度 3 次元ベクトルの個数を返します.

```
SFVec3d MFVec3d::getValue(int i);
```

このフィールドの持つ i 番目の倍精度 3 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFVec4d

```
int MFVec4d::count();
```

このフィールドの持つ倍精度 4 次元ベクトルの個数を返します.

```
SFVec4d MFVec4d::getValue(int i);
```

このフィールドの持つ i 番目の倍精度 4 次元ベクトルを返します.
i は 0 以上、count()未満です.

MFColor

```
int MFColor::count();
```

このフィールドの持つ色情報の個数を返します.

```
SFColor MFColor::getValue(int i);
```

このフィールドの持つ i 番目の色情報を返します.
i は 0 以上、count()未満です.

MFString

```
int MFString::count();
```

このフィールドの持つ文字列の個数を返します.

```
const char *MFString::getValue(int i);
```

このフィールドの持つ i 番目の文字列へのポインタを返します.
（返り値のポインタを解放してはいけません）

i は 0 以上、count()未満です.

MFNode

```
int MFNode::count();
```

このフィールドの持つノードの個数を返します.

```
CX3DNode *MFNode::getNode(int i);
```

このフィールドの持つ i 番目のノードへのポインタを返します.
ただし返り値のポインタ（の指すノード）の所有権は MFNode が持っていますので、
返り値を delete してはいけません.

i は 0 以上、count()未満です.

```
CX3DNode *MFNode::releaseNode(int i);
```

getNode()と同じく、このフィールドの持つ i 番目のノードへのポインタを返しますが、同時にノードの所有権を放棄します. すなわち、呼び出し側は、返り値を delete しなければなりません.

i は 0 以上、count()未満です.

X3DParser(C++ライブラリ)の使い方

ルートノードを得るには？

ルートノード(BaseWorldRoot)自体は、あまり用がないと思われます.
一応ルートノードを得るメソッドは java 側には用意してありますが、ほとんど使うことはないと思われるため、C++のインターフェースには含めていません.

ルートノードの子ノードを得るには？

こちらは重要ですので、頻繁に使うでしょう.

MFNode *CX3DParser::getChildrenOfRootNode()で得ることができます.

返り値の MFNode オブジェクトの解放責任は呼び出し側にありますので注意してください.
使い終わったら必ず delete が必要です.

（例）

```
CX3DParser parser;

parser.parse("MyTestX3D.x3d");
MFNode *children = parser.getChildrenOfRootNode();
if (children)
{
	// ... children を使った処理 ...
	delete children;	// 必ず delete すること
}
```

［参照］
Test_GetChildrenOfRootNode.cpp

ノードの型を知るには？

CX3DNodeType CX3DNode::getNodeType()を使います

ノードの型に応じて、以下の値のいずれかが返ります.

```
enum CX3DNodeType
{
	BASE_NODE,  // 以下のノードではない、何かのノード
	MATERIAL_NODE,
	TRANSFORM_NODE,
	GROUP_NODE,
	APPEARANCE_NODE,
	SCENE_NODE,
	SHAPE_NODE,
	BOX_NODE,
	CONE_NODE,
	CYLINDER_NODE,
	SPHERE_NODE,
	COLOR_NODE,
	COORDINATE_NODE,
	NORMAL_NODE,
	INDEXED_LINE_SET_NODE,
	INDEXED_FACE_SET_NODE,
	DIRECTIONAL_LIGHT_NODE,
	POINT_LIGHT_NODE,
	SPOT_LIGHT_NODE,
	TEXTURE_TRANSFORM_NODE,
	TEXTURE_COORDINATE_NODE,
	IMAGE_TEXTURE_NODE,
	VIEWPOINT_NODE,
	OPENHRP_HUMANOID_NODE,
	OPENHRP_JOINT_NODE,
	OPENHRP_SEGMENT_NODE,
	HANIM_DISPLACER_NODE,
	HANIM_HUMANOID_NODE,
	HANIM_JOINT_NODE,
```

```
        HANIM_SEGMENT_NODE,
        HANIM_SITE_NODE
};
```

CX3DNodeType 型の定義は CX3DNode.h にあります.

通常は、型を調べた後、専用ノードへキャストして使います.

```
CX3DNode *pNode;
switch (pNode->getNodeType())
{
        case TRANSFORM_NODE:
                CX3DTransformNode *pTrans = (CX3DTransformNode *)pNode;
                // pTrans を使う
                break;

        case SHAPE_NODE:
                CX3DShapeNode *pShape = (CX3DShapeNode *)pNode;
                // pShape を使う
                break;

        case MATERIAL_NODE:
                CX3DMaterialNode *pMat = (CX3DMaterial *)pNode;
                // pMat を使う
                break;
}
```

（注意）
BASE_NODE が返るのは、CX3DNodeType 列挙型の、BASE_NODE 以外のどのノード(MATERIAL_NODE,...,HANIM_SITE_NODE)でもなかった場合です. この場合は専用ノードのクラスにキャストすることはできません.

ノード名を知るには？

char *CX3DNode::getNodeName()を使います.

[参照]
Test_GetNodeName.cpp

ノードの持つフィールドの個数を知るには？

int CX3DNode::countFields()を使います.

[参照]
Test_LoopFieldOfNode.cpp

フィールドの型を知るには？

int CX3DNode::getFieldType(int iField)を使います

返り値はCX3DFieldTypeで定義されている値と同じものです.

型名を文字列で得たい場合は
char *getFieldTypeString(int iField);
を使います.
ここで引数のiFieldはフィールドインデックスです.
（getFieldType()の返り値ではありませんので注意してください）

フィールドインデックスとは、フィールドの識別子となる整数値です.
フィールドインデックスは、0以上、countFields()未満の値を取ります.
int getFieldIndex(char *filedName)により、フィールド名からフィールドインデックスを得ることもできます.

[参照]
Test_LoopFieldOfNode.cpp

フィールドの値を得るには？

ノードの型を調べて専用ノードクラスへキャストし、その専用クラスノードのメソッドを使ってください.

[参照]
Test_BoxNode.cpp
Test_TransformNode.cpp

ノードに存在するすべてのフィールドをループするには？

以下のようにします.

```
CX3DNode *pNode;
int n = pNode->countFields();
for (int i=0; i<n; i++)
{
        ... i 番目のフィールドに関する処理 ...
}
```

[参照]
Test_LoopFieldOfNode.cpp

汎用ノード(CX3DNode)に属するフィールドを得るには？

専用ノードが用意されていないノード(Extrusion ノード等) は、createFieldValue()でフィールドの値を得ます.

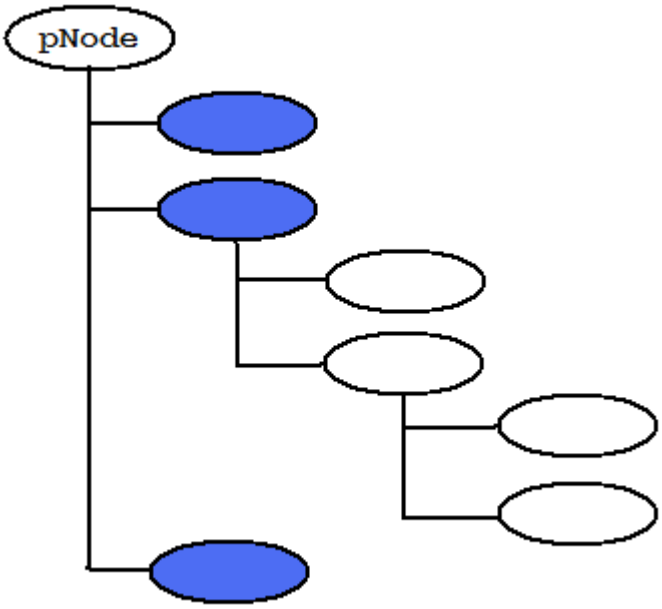
createFieldValue()は new されたメモリ領域を返しますので、使い終わったら delete しなければなりません.

ノードの子ノードを探索するには？

MFNode *CX3DNode::searchNodesFromDirectChildren(char *nodeName);
を使います.

searchNodesFromDirectChildren は、ノードの直下の子ノードの中から、ノード名が nodeName のノードを探索します.

下図において、pNode が探索開始ノードの場合、探索対象は青いノードです.
（pNode 自身は探索対象に入りません）



［注意］
返り値の MFNode オブジェクトの解放責任は呼び出し側にあります.
使い終わったら必ず delete してください.

（例）

```
CX3DNode *pNode;

// pNode 直下の子ノードから、Transform ノードを探す
MFNode *transNodes = pNode->searchNodesFromDirectChildren("Transform");
if (transNodes)
{
        // ... transNodes を使った処理 ...
        delete transNodes;  // 必ず delete すること
}
```

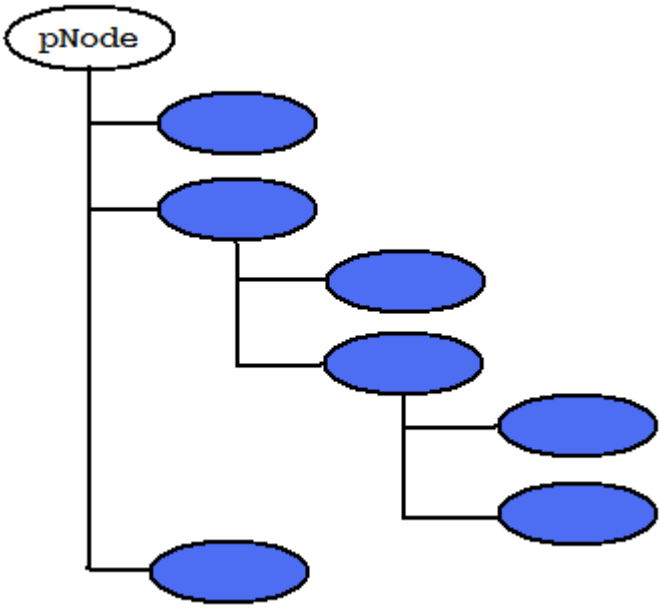
ノードのすべての子ノードを再帰的に探索するには？

```
MFNode *CX3DNode::searchNodesFromAllChildren(char *nodeName);
```

を使います.

searchNodesFromAllChildren は、ノードのすべての子ノードの中から、ノード名が nodeName のノードを再帰的に探索します.

下図において、pNode が探索開始ノードの場合、探索対象は青いノードです.
（pNode 自身は探索対象に入りません）



[注意]
返り値の MFNode オブジェクトの解放責任は呼び出し側にあります.
使い終わったら必ず delete してください.

（例）

```
CX3DNode *pNode;

// pNode のすべての子ノードから、Transform ノードを探す
MFNode *allTransNodes = pNode->searchNodesFromAllChildren("Transform");
if (allTransNodes)
{
        // ... allTransNodes を使った処理 ...
        delete allTransNodes;       // 必ず delete すること
}
```

あるノードの親ノードを得るには？

親ノードをたどることはできません.

あるノードの内容をすべてログに書き出すには？

void CX3DNode::print()を使います.

すべての def ノードの名前を得るには？

std::vector<std::string> CX3DParser::getDefNames()でパースした内容の中で定義されている DEF 名をすべて得ることができます.

[参照]
Test_GetDefNames.cpp

def ノード名から def ノードを得るには？

```
CX3DNode *CX3DParser::getDefNode(char *defName);
```

で DEF 名から対応するノードを得ることができます.

[参照]
Test_GetDefNode.cpp

物理計算用の単純形状を得るには？

```
CSimplifiedShape *CSimplifiedShapeFactory::calc(
        MFNode *shapes,
        CSimplifiedShape::SHAPE_TYPE hint = CSimplifiedShape::NONE);
```

を使って、Shape ノード（の集合）から、1 つの単純形状を求めることができます.

shapes には、Shape ノードを含む MFNode を与えます.
各 Shape ノードの geomertry フィールドには IndexedFaceSet ノードがセットされていることが仮定されています.

hint には、単純形状を求める際のヒントを与えます.
ヒントに従って球、シリンダー、箱（直方体）のいずれかの単純形状を求めます.
何も与えなかった場合は、3 種類の単純形状のうち、最も歪度の小さい単純形状を自動的に求めます.

[参照]
Test_CalcSimplifiedShape.cpp

ログを出力するには？

```
CX3DParser::printLog(char *format, ...);
CX3DParser::printIndentLog(int indentLevel, char *format, ...);
```

を使います.

printLog()はログの出力先にログを書き出します.
printf()と同じフォーマット指定子が使えます.

printIndentLog()はインデント付きでログを出力します.

ログの出力先は、デフォルトでは stderr です.

ログの出力先を変えるには？

```
CX3DParser::openDebugLog(char *fname, bool bAppend=false);
```

でログを出力するファイルを指定できます. または、

```
CX3DParser::setDebugLogFp(FILE *fp);
```

で直接 fp を出力先に設定することもできます.

ログの出力を抑制するには？

```
CX3DParser::setDebugLog(NULL);
```

をコールします.

または、コンパイラの設定で X3DPARSER_DISABLE_DEBUG_LOG マクロを定義して再コンパイルします.

サンプルファイル一覧

Test_BoxNode.cpp

Box ノードのフィールドを表示する

```
>X3DParserTest.exe 4 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-1.x3d
**** Fields of BoxNode ****
size : (4.500000, 3.000000, 20.000000)
solid : TRUE
```

Test_CalcSimpifiedShape.cpp

Shape ノードから単純形状を計算する.

```
>X3DParserTest.exe 6 ..¥..¥..¥..¥shapes¥VRML¥tumiki_car.wrl
** sphere **
       radius  : 2.770568
       center  : (0.790777, 1.217878, -0.000022)
       歪度 : 0.403105
各単純形状の歪度
       球       : 0.403105
       シリンダー : 0.604044
       箱       : 0.489846
```

Test_GetChildrenOfRootNode.cpp

ルートノードの直下にあるすべての子ノードを列挙する.

```
>X3DParserTest.exe 1 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-19.x3d
**** Children of Root Node ****
Background
```

```
Transform
Transform
Transform
Transform
```

Test_GetDefNames.cpp

DEF 指定されたノード名をすべて表示する.

```
>X3DParserTest.exe 8 ..¥..¥..¥..¥shapes¥X3DSamples¥list3-13.x3d
*** def names in list3-13.x3d ***
Body
LeftWing
Wing
```

Test_GetDefNode.cpp

特定の DEF 指定されたノードを探し、ノードの内容を表示する.

```
>X3DParserTest.exe 9 ..¥..¥..¥..¥shapes¥X3DSamples¥list3-13.x3d LeftWing
**** Node contents of defName(LeftWing) ****
Shape (SHAPE_NODE)
  appearance
    Appearance (APPEARANCE_NODE)
      material
        Material (MATERIAL_NODE)
          diffuseColor : (0.700000 0.400000 0.400000)
          specularColor : (0.000000 0.000000 0.000000)
          emissiveColor : (0.000000 0.000000 0.000000)
          shininess : 0.200000
          transparency : 0.000000
          ambientIntensity : 0.200000
      texture
      textureTransform
```

```
    pointProperties
    lineProperties
    fillProperties
    textureProperties
  geomtery
    IndexedFaceSet (INDEXED_FACE_SET_NODE)
      coord
        Coordinate (COORDINATE_NODE)
          point [8]
            (0.000000 1.000000 0.000000)
            (5.000000 1.000000 50.000000)
            (20.000000 1.000000 50.000000)
            (25.000000 1.000000 0.000000)
            (0.000000 -1.000000 0.000000)
            (5.000000 -1.000000 50.000000)
            (20.000000 -1.000000 50.000000)
            (25.000000 -1.000000 0.000000)
      color
      normal
      texCoord
      solid : FALSE
      convex : TRUE
      ccw : TRUE
      creaseAngle : 0.000000
      colorPerVertex : TRUE
      normalPerVertex : TRUE
      coordIndex [30]
        (0 1 2 3 -1 4 5 6 7 -1 0 1 5 4 -1 1 2 6 5 -1 2 3 7 6 -1 3 0 4 7 -1 )
      colorIndex [0]
        ()
      normalIndex [0]
        ()
      texCoordIndex [0]
        ()
```

Test_GetNodeName.cpp

ノード名を表示する.

```
>X3DParserTest.exe 2 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-1.x3d
**** Node names of (list2-1.x3d) ****
Background
Shape
```

Test_LoopFieldOfNode.cpp

ノードの持つフィールドを列挙する.

```
>X3DParserTest.exe 3 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-14.x3d
**** Fields of Node in (list2-14.x3d) ****
Background
       metadata [11:SFNode]
       set_bind [1:SFBool]
       bindTime [9:SFTime]
       isBound [1:SFBool]
       groundAngle [6:MFFloat]
       groundColor [22:MFColor]
       skyAngle [6:MFFloat]
       skyColor [22:MFColor]
       backUrl [28:MFString]
       frontUrl [28:MFString]
       leftUrl [28:MFString]
       rightUrl [28:MFString]
       bottomUrl [28:MFString]
       topUrl [28:MFString]
Shape
       metadata [11:SFNode]
       appearance [11:SFNode]
       geometry [11:SFNode]
       bboxSize [15:SFVec3f]
       bboxCenter [15:SFVec3f]
```

```
Transform
       metadata [11:SFNode]
       children [12:MFNode]
       addChildren [12:MFNode]
       removeChildren [12:MFNode]
       bboxCenter [15:SFVec3f]
       bboxSize [15:SFVec3f]
       center [15:SFVec3f]
       rotation [19:SFRotation]
       scale [15:SFVec3f]
       scaleOrientation [19:SFRotation]
       translation [15:SFVec3f]
```

Test_OpenHRP.cpp

OpenHRP 準拠の VRML ファイルを読み、いくつかの（サーバー側処理に必要と思われる）フィールドの値を取り出す

```
>X3DParserTest.exe 7 ..¥..¥..¥..¥shapes¥X3DSamples¥openHRP_robot_body.x3d
**** OpenHRP field value of (openHRP_robot_body.x3d) ****
name: (BODY)
type: (free)
translation: (0.000000, 0.000000, 0.000000)
rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
  name: (body)
  Shape (SHAPE_NODE)
    appearance
    geomtery
      Box (BOX_NODE)
        solid : TRUE
        size  : (1.000000 1.000000 0.800000)
  name: (NECK)
  type: (fixed)
  translation: (0.000000, 0.500000, 0.000000)
  rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
    name: (HEAD)
```

```
  Shape (SHAPE_NODE)
    appearance
    geomtery
      Sphere (SPHERE_NODE)
        solid  : TRUE
        radius : (0.300000)
name: (R_SHOULDER)
type: (fixed)
translation: (-0.500000, 0.300000, 0.000000)
rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
  name: (RIGHT_U_ARM)
  Shape (SHAPE_NODE)
    appearance
    geomtery
      Box (BOX_NODE)
        solid : TRUE
        size  : (0.300000 0.400000 0.300000)
  name: (R_ELBOW)
  type: (fixed)
  translation: (0.000000, -0.500000, 0.000000)
  rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
    name: (RIGHT_L_ARM)
    Shape (SHAPE_NODE)
      appearance
      geomtery
        Box (BOX_NODE)
          solid : TRUE
          size  : (0.300000 0.400000 0.300000)
name: (L_SHOULDER)
type: (fixed)
translation: (0.500000, 0.300000, 0.000000)
rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
  name: (LEFT_ARM)
  Shape (SHAPE_NODE)
    appearance
    geomtery
      Box (BOX_NODE)
```

```
        solid : TRUE
        size  : (0.400000 1.000000 0.400000)
  name: (J_LBODY0)
  type: (fixed)
  translation: (0.000000, 0.000000, 0.000000)
  rotation:    (0.000000, 0.000000, 1.000000, 0.000000)
    name: (LBODY)
    Shape (SHAPE_NODE)
      appearance
      geomtery
        Box (BOX_NODE)
          solid : TRUE
          size  : (0.900000 1.000000 0.800000)
```

Test_SimpleParse.cpp

引数に与えた VRML を読み込み、パース結果を出力する.

```
>X3DParserTest.exe 0 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-14.x3d
*****  content of list2-14.x3d  *****
Background (BASE_NODE)
  set_bind
  bindTime
    [DOUBLE_DATA]
    0.000000
  isBound
    [BOOLEAN_DATA]
    FALSE
  groundAngle
    [FLOAT_ARRAY_DATA] (n=0)
  groundColor
    [FLOAT_ARRAY_DATA] (n=0)
  skyAngle
    [FLOAT_ARRAY_DATA] (n=0)
  skyColor
    [FLOAT_ARRAY_DATA] (n=3)
```

```
    [0] 1.000000
    [1] 1.000000
    [2] 1.000000
  backUrl
    [STRING_ARRAY_DATA] (n=0)
  frontUrl
    [STRING_ARRAY_DATA] (n=0)
  leftUrl
    [STRING_ARRAY_DATA] (n=0)
  rightUrl
    [STRING_ARRAY_DATA] (n=0)
  bottomUrl
    [STRING_ARRAY_DATA] (n=0)
  topUrl
    [STRING_ARRAY_DATA] (n=0)
Shape (SHAPE_NODE)
  appearance
    Appearance (APPEARANCE_NODE)
      material
        Material (MATERIAL_NODE)
          diffuseColor : (1.000000 0.000000 0.000000)
          specularColor : (0.000000 0.000000 0.000000)
          emissiveColor : (0.000000 0.000000 0.000000)
          shininess : 0.200000
          transparency : 0.000000
          ambientIntensity : 0.200000
      texture
      textureTransform
      pointProperties
      lineProperties
      fillProperties
      textureProperties
  geomtery
    Box (BOX_NODE)
      solid : TRUE
      size  : (2.000000 2.500000 3.000000)
Transform (TRANSFORM_NODE)
```

```
  translation : (4.000000 0.000000 0.000000)
  rotation : (0.000000 0.000000 1.000000)(0.000000)
  scale : (1.000000 1.000000 1.000000)
  center : (0.000000 0.000000 0.000000)
  rotation : (0.000000 0.000000 1.000000)(0.000000)
  children
    Shape (SHAPE_NODE)
      appearance
        Appearance (APPEARANCE_NODE)
          material
            Material (MATERIAL_NODE)
              diffuseColor : (0.000000 1.000000 1.000000)
              specularColor : (0.000000 0.000000 0.000000)
              emissiveColor : (0.000000 0.000000 0.000000)
              shininess : 0.200000
              transparency : 0.000000
              ambientIntensity : 0.200000
          texture
          textureTransform
          pointProperties
          lineProperties
          fillProperties
          textureProperties
      geomtery
        Box (BOX_NODE)
          solid : TRUE
          size  : (2.000000 2.500000 3.000000)
```

Test_TransformNode.cpp

Transform ノードの内容を表示する.

```
>X3DParserTest.exe 5 ..¥..¥..¥..¥shapes¥X3DSamples¥list2-14.x3d
**** Some Field values of Transform Node in (list2-14.x3d) ****
(1/1)
translation : (4.000000, 0.000000, 0.000000)
```

```
rotation : (0.000000, 0.000000, 1.000000, 0.000000)
center : (0.000000, 0.000000, 0.000000)
scale : (1.000000, 1.000000, 1.000000)
scaleOrientation : (0.000000, 0.000000, 1.000000, 0.000000)
```