

KiCad 插件

Table of Contents

KiCad 插件系介绍	2
插件	2
教程：3D插件	4
基本的 3D 插件	4
高级 3D 插件	9
应用程序接口 (API)	11
插件 API	11
场景 API	14

KiCad 插件系

版权

本文档版权所有 © 2016，其贡献者如下所列。您可以根据 GNU 通用公共许可 (<http://www.gnu.org/licenses/gpl.html>)，版本 3 或更高版本，或知识共享署名许可 (<http://creativecommons.org/licenses/by/3.0/>)，版本 3.0 或更高版本的条款分派和/或修改它。

本指南中的所有商标均属于其合法所有者。

贡献者

Cirilo Bernardo

翻印人

taotieren <admin@taotieren.com>, 2019, 2020, 2021.

Telegram 简体中文交流群: https://t.me/KiCad_zh_CN

反馈

请将任何报告、建议或新版本引导到此：

- 关于 KiCad 文档: <https://gitlab.com/kicad/services/kicad-doc/issues>
- 关于 KiCad 软件: <https://gitlab.com/kicad/code/kicad/issues>
- 关于 KiCad 软件 i18n: <https://gitlab.com/kicad/code/kicad-i18n/issues>

出版日期和软件版本

2016 年 1 月 29 日出版。

KiCad 插件系介绍

KiCad 插件系是一个使用共享扩展 KiCad 功能的框架。使用插件的一个主要点是在开发插件时没有必要重建 KiCad 套件;事实上,可以借助 KiCad 源代码中的一小部分构建插件。通过确保开发人员与正在开发的插件直接相关的代码从而减少每个构建和周期所需的在插件开发期间除构建 KiCad 的要求极大地提高了工作效率。

插件最初是 3D 模型查看器开发的,因此可以支持更多类型的 3D 模型,而无需支持的每种新模型类型的 KiCad 源代码行重大更改。插件框架后来被推广,以便将来开发人员可以构建不同类型的插件。目前,只有 3D 插件在 KiCad 中但可以想象最终将开发 PCB 插件,以使用智能数据输入器和输出器。

插件

插件分插件因为每个插件都解决了特定域中的问题因此需要域独有的接口。例如,3D 模型插件从文件加载 3D 模型数据并将数据可由 3D 查看器显示的格式。PCB 输入/输出插件将读取 PCB 数据并输出其他电气或机械数据格式,或将外部格式到 KiCad PCB。目前只开发了 3D 插件它将成为本文档的重点。

插件需要在 KiCad 源代码中建立代来管理插件代码的加载在 KiCad 源代码中,文件 'plugins/ldr/pluginldr.h' 声明了所有插件加载器的基类,它声明了我期望在任何 KiCad 插件(板代中找到的最基本的函数,它的提供了插件加载器和可用插件之间的版本兼容性的基本类 'plugins/ldr/3d/pluginldr3D.h' 声明了 3D 插件的加载器。加载器加载定义的插件并使其功能可用于 KiCad。插件加载器的每个例代表一个插件并充当 KiCad 和插件功能之间的透明桥梁。加载器不是 KiCad 中支持插件所需的唯一代码我需要代码来插件和代码以通过插件加载器用插件的功能。在 3D 插件的情况下,和用功能都包含在 S3D_CACHE 中。

除非正在开发新的插件否则插件开发人员不需要关心 KiCad 管理插件的内部代码的;插件只需要定义其特定插件声明的函数。

include/plugins/kicad_plugin.h 声明了所有 KiCad 插件所需的泛型函数;这些函数插件提供特定插件的名称,提供插件 API 的版本信息,提供特定插件的版本信息,并提供插件 API 的基本版本兼容性。而言之,这些功能是:

```

/* 返回命名插件的 UTF-8 字符串 */
/* Return a UTF-8 string naming the Plugin Class */
char const* GetKicadPluginClass( void );

/* 返回插件API的版本信息 */
/* Return version information for the Plugin Class API */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/*
    如果插件中实现了版本检查返回 true。
    确定给定的插件 API 是否兼容。
    Return true if the version check implemented in the plugin
    determines that the given Plugin Class API is compatible.
*/
bool CheckClassVersion( unsigned char Major,
    unsigned char Minor, unsigned char Patch, unsigned char Revision );

/* 返回具体插件的名称, 例如 "PLUGIN_3D_VRML" */
/* Return the name of the specific plugin, for example "PLUGIN_3D_VRML" */
const char* GetKicadPluginName( void );

/* 返回特定插件的版本信息 */
/* Return version information for the specific plugin */
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

```

插件 `PLUGIN_3D`

`include/plugins/3d/3d_plugin.h` 声明了必须由所有 3D 插件实现的函数，并定义了插件所需的多个函数以及用不得重新实现的函数。用不得重新实现的已定义函数是：

```

/* 返回插件名 "PLUGIN_3D" */
/* Returns the Plugin Class name "PLUGIN_3D" */
char const* GetKicadPluginClass( void );

/* 返回 PLUGIN_3D API 的版本信息 */
/* Return version information for the PLUGIN_3D API */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/*
    由行的开行人控制行的基本版本
    PLUGIN_3D 的加载器，如果。
    通过
    Performs basic version checks enforced by the developers of
    the loader for the PLUGIN_3D class and returns true if the
    checks pass
*/
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
    unsigned char Patch, unsigned char Revision );

```

用必须实现的功能如下：

```

/* 返回插件支持的扩展字符串数 */
/* Return the number of extension strings supported by the plugin */
int GetNExtensions( void );

/*
    返回请求的扩展字符串；有效值 0 到
    GetNExtensions() - 1
    Return the requested extension string; valid values are 0 to
    GetNExtensions() - 1
*/
char const* GetModelExtension( int aIndex );

/* 返回插件支持的文件过滤器数 */
/* Return the total number of file filters supported by the plugin */
int GetNFilters( void );

/*
    返回请求的文件过滤器；有效值 0 到。
    GetNFilters() - 1
    Return the file filter requested; valid values are 0 to
    GetNFilters() - 1
*/
char const* GetFileFilter( int aIndex );

/*
    如果插件可以渲染某种类型的 3D 模型，返回 true。
    在某些情况下，插件可能尚未提供可视模型。
    并且必须返回 false。
    Return true if the plugin can render this type of 3D model.
    In some cases a plugin may not yet provide a visual model
    and must return false.
*/
bool CanRender( void );

/* 加载指定的模型并返回指向其可视化模型数据的指针 */
/* Load the specified model and return a pointer to its visual model data */
SCENEGRAPH* Load( char const* aFileName );

```

教程：3D 插件

本文包含 `PLUGIN_3D` 的两个非常简单的插件的描述，并引用完成代码的放置和构建。

基本的 3D 插件

本教程将引用并打开一个名为“`PLUGIN_3D_DEMO1`”的非常基本的 3D 插件。本教程的目的只是演示了一个非常基本的 3D 插件的构造，除了提供一些允许 KiCad 用户在 3D 模型中指定文件名的字符串之外什么都不做。这里演示的代码是任何 3D 插件的最低要求，可以用作构建更高插件的模板。

为了构建演示项目，我需要以下内容：

- [CMake](#)
-

KiCad 插件

- KiCad 场景 'kicad_3dsg'

要自己 KiCad 和 我将使用 CMake FindPackage 脚本; 如果相关的文件安装到 '\${KICAD_ROOT_DIR}/kicad' 并且 KiCad Scene Graph 安装在 '\${KICAD_ROOT_DIR}/lib' 中, 本教程中提供的脚本适用于 Linux 和 Windows。

要开始, 让我建一个目录和 FindPackage 脚本 :

```

mkdir demo && cd demo
export DEMO_ROOT=${PWD}
mkdir CMakeModules && cd CMakeModules
cat > FindKICAD.cmake << _EOF
find_path( KICAD_INCLUDE_DIR kicad/plugins/kicad_plugin.h
    PATHS ${KICAD_ROOT_DIR}/include $ENV{KICAD_ROOT_DIR}/include
    DOC "Kicad plugins header path."
)

if( NOT ${KICAD_INCLUDE_DIR} STREQUAL "KICAD_INCLUDE_DIR-NOTFOUND" )

    # 从 sg_version.h 中提取版本信息
    # attempt to extract the version information from sg_version.h
    find_file( KICAD_SGVERSION sg_version.h
        PATHS ${KICAD_INCLUDE_DIR}
        PATH_SUFFIXES kicad/plugins/3dapi
        NO_DEFAULT_PATH )

    if( NOT ${KICAD_SGVERSION} STREQUAL "KICAD_SGVERSION-NOTFOUND" )

        # 提取 "#define KICADSG_VERSION*" 行
        # extract the "#define KICADSG_VERSION*" lines
        file( STRINGS ${KICAD_SGVERSION} _version REGEX "^#define.*KICADSG_VERSION.*" )

        foreach( SVAR ${_version} )
            string( REGEX MATCH KICADSG_VERSION_[M,A,J,O,R,I,N,P,T,C,H,E,V,I,S]* _VARNAME
                ${SVAR} )
            string( REGEX MATCH [0-9]+ _VALUE ${SVAR} )

            if( NOT ${_VARNAME} STREQUAL "" AND NOT ${_VALUE} STREQUAL "" )
                set( ${_VARNAME} ${_VALUE} )
            endif()

        endforeach()

        # 确保 NOT SG3D_VERSION* 的计算结果 "0"
        # ensure that NOT SG3D_VERSION* will evaluate to '0'
        if( NOT _KICADSG_VERSION_MAJOR )
            set( _KICADSG_VERSION_MAJOR 0 )
        endif()

        if( NOT _KICADSG_VERSION_MINOR )
            set( _KICADSG_VERSION_MINOR 0 )
        endif()

        if( NOT _KICADSG_VERSION_PATCH )
            set( _KICADSG_VERSION_PATCH 0 )
        endif()

        if( NOT _KICADSG_VERSION_REVISION )
            set( _KICADSG_VERSION_REVISION 0 )
        endif()

        set( KICAD_VERSION
            ${_KICADSG_VERSION_MAJOR}.${_KICADSG_VERSION_MINOR}.${_KICADSG_VERSION_PATCH}.${_KICADSG_VERSION_REVISION} )
        unset( KICAD_SGVERSION CACHE )

    endif()
endif()

```

必须安装 Kicad 及其插件; 如果将它安装到用目或 Linux 上的 ‘/opt’ 下, 或者您使用的是 Windows 需要将 ‘KICAD_ROOT_DIR’ 环境变量指向包含 KiCad ‘include’ 和 ‘lib’ 目的目。于 OS X, 此示的 FindPackage 脚本可能需要行一些整。

要配置和构建教程代我将使用 CMake 并建一个 ‘CMakeLists.txt’ 脚本文件 :

```
cd ${DEMO_ROOT}
cat > CMakeLists.txt << _EOF
# 声明项目名称
# declare the name of the project
project( PLUGIN_DEMO )

# 我们是否有具有所有必需功能的 CMake 版本
# check that we have a version of CMake with all required features
cmake_minimum_required( VERSION 2.8.12 FATAL_ERROR )

# 通知 CMake 何可以找到 FindKICAD 脚本
# inform CMake of where to find the FindKICAD script
set( CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/CMakeModules )

# 已安装的 kicad 头和
# attempt to discover the installed kicad headers and library
# 并置量: (and set the variables:)
#     KICAD_INCLUDE_DIR
#     KICAD_LIBRARY
find_package( KICAD 1.0 REQUIRED )

# 将 kicad include 目添加到器的搜索路径中
# add the kicad include directory to the compiler's search path
include_directories( ${KICAD_INCLUDE_DIR}/kicad )

# 建名为 s3d_plugin_demo1 的插件
# create a plugin named s3d_plugin_demo1
add_library( s3d_plugin_demo1 MODULE
    src/s3d_plugin_demo1.cpp
)

_EOF
```

第一个演示目非常基; 它由一个文件成, 除了器默认之外没有外部接依。我首先建一个源目

```
cd ${DEMO_ROOT}
mkdir src && cd src
export DEMO_SRC=${PWD}
```

在我自己建插件源 :

s3d_plugin_demo1.cpp

```
#include <iostream>

// 3d_plugin.h 定义了 3D 插件所需的功能
// the 3d_plugin.h header defines the functions required of 3D plugins
#include "plugins/3d/3d_plugin.h"

// 定义一个插件的版本信息, 不要混淆
// define the version information of this plugin; do not confuse this
// 使用在 3d_plugin.h 中定义的插件版本
// with the Plugin Class version which is defined in 3d_plugin.h
#define PLUGIN_3D_DEMO1_MAJOR 1
#define PLUGIN_3D_DEMO1_MINOR 0
#define PLUGIN_3D_DEMO1_PATCH 0
#define PLUGIN_3D_DEMO1_REVNO 0

// 提供一个提供此插件名称的函数
// implement the function which provides users with this plugin's name
const char* GetKicadPluginName( void )
{
    return "PLUGIN_3D_DEMO1";
}

// 提供一个提供此插件版本的功能
// implement the function which provides users with this plugin's version
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision )
{
    if( Major )
        *Major = PLUGIN_3D_DEMO1_MAJOR;

    if( Minor )
        *Minor = PLUGIN_3D_DEMO1_MINOR;

    if( Patch )
        *Patch = PLUGIN_3D_DEMO1_PATCH;

    if( Revision )
        *Revision = PLUGIN_3D_DEMO1_REVNO;

    return;
}

// 支持的扩展名数量; 在 *NIX 系上, 扩展名
// number of extensions supported; on *NIX systems the extensions are
// 提供两次-一次小写, 一次大写
// provided twice - once in lower case and once in upper case letters
#ifdef _WIN32
    #define NEXTS 7
#else
    #define NEXTS 14
#endif

// 支持的过滤器集数量
// number of filter sets supported
#define NFILS 5

// 定义此扩展字符串和过滤器字符串
// define the extension strings and filter strings which this
// 插件将提供用
// plugin will supply to the user
```


此源文件是 3D 插件的所有最低要求。插件不会渲染模型生成任何数据，但它可以 KiCad 提供支持的模型文件扩展名和文件扩展名列表，以增强 3D 模型文件对话框。在 KiCad 中，扩展字符串用于可用于添加指定模型的插件；例如，如果插件是 ‘wrl’，那么 KiCad 将用声称支持扩展 ‘wrl’ 的每个插件，直到插件返回可序列化数据。每个插件提供的文件管理器将到 3D 文件管理器对话框，以改善 UI。

要构建插件：

```
cd ${DEMO_ROOT}
# export KICAD_ROOT_DIR if necessary
mkdir build && cd build
cmake .. && make
```

插件将被构建但未安装；如果要添加插件，必须将插件文件复制到 KiCad 的插件目录

高级 3D 插件

本教程将引用一个名为 “PLUGIN_3D_DEMO2” 的 3D 插件。本教程的目的是演示 KiCad 管理器可以渲染的非常基本的场景的构造。插件声称处理 ‘txt’ 类型的文件。虽然文件必须存在，以便存管理器用插件，但此插件不处理文件内容；相反，插件只是建立一个包含一个四面体的场景。本教程假定第一个教程已完成，并且已建立 CMakeLists.txt 和 FindKICAD.cmake 脚本文件。

将新的源文件放在与上一个教程的源文件相同的目录中，我将扩展上一个教程的 CMakeLists.txt 文件来构建本教程。由于这个插件会 KiCad 建立一个场景，我需要连接到 KiCad 的场景 ‘kicad_3dsg’。KiCad 的场景提供了一组可用于构建场景对象的；场景对象是 3D 存管理器使用的中间数据可序列化格式。所有支持模型可序列化的插件都必须通过此将模型数据到场景

第一步是扩展 ‘CMakeLists.txt’ 来构建这个教程目录：

```
cd ${DEMO_ROOT}
cat >> CMakeLists.txt << _EOF
add_library( s3d_plugin_demo2 MODULE
    src/s3d_plugin_demo2.cpp
)

target_link_libraries( s3d_plugin_demo2 ${KICAD_LIBRARY} )
_EOF
```

在我切换到源目录并建立源文件：

```
cd ${DEMO_SRC}
```

s3d_plugin_demo2.cpp

```
#include <cmath>
// 3D 插件声明
// 3D Plugin Class declarations
#include "plugins/3d/3d_plugin.h"
// KiCad 场景图形接口
// interface to KiCad Scene Graph Library
#include "plugins/3dapi/ifsg_all.h"

// 插件的版本信息
// version information for this plugin
#define PLUGIN_3D_DEMO2_MAJOR 1
#define PLUGIN_3D_DEMO2_MINOR 0
#define PLUGIN_3D_DEMO2_PATCH 0
#define PLUGIN_3D_DEMO2_REVNO 0

// 提供此插件的名称
// provide the name of this plugin
const char* GetKicadPluginName( void )
{
    return "PLUGIN_3D_DEMO2";
}

// 提供此插件的版本
// provide the version of this plugin
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision )
{
    if( Major )
        *Major = PLUGIN_3D_DEMO2_MAJOR;

    if( Minor )
        *Minor = PLUGIN_3D_DEMO2_MINOR;

    if( Patch )
        *Patch = PLUGIN_3D_DEMO2_PATCH;

    if( Revision )
        *Revision = PLUGIN_3D_DEMO2_REVNO;

    return;
}

// 支持的扩展数量
// number of extensions supported
#ifdef _WIN32
#define NEXTS 1
#else
#define NEXTS 2
#endif

// 支持的过滤器集数量
// number of filter sets supported
#define NFILS 1

static char ext0[] = "txt";

#ifdef _WIN32
static char fil0[] = "demo (*.txt)|*.txt";
#else
```

用程序接口（API）

插件通用程序接口（API）每个插件都有其特定的 API，在 3D 插件教程中，我已看到了由“3d_plugin.h”声明的 3D 插件 API 的示例。插件也可能依赖于 KiCad 源代码中定义的其他 API；在 3D 插件的情况下，支持模型可视化的所有插件必须与“ifsg_all.h”及其包含的中声明的 Scene Graph API 交互。

本描述了插件可能需要的插件 API 和其他 KiCad API 的信息。

插件 API

目前只有一个 KiCad 声明的插件 3D 插件 所有 KiCad 插件都必须包含文件“kicad_plugin.h”中声明的一些基本函数；一些声明称 Base Kicad 插件不存在 Base Kicad 插件的文件的存在纯粹是为了确保插件开发人员可以在每个插件中定义一些函数。

在 KiCad 中，插件加载器的每个示例都使用了插件提供的 API，就像插件加载器是提供插件服务的唯一接口是通用 Plugin 加载器提供的，它提供包含与插件类似的函数名的公共接口；如果例如没有加载插件，参数列表可以简化以适应通用通知可能遇到的任何需要。在内部，插件加载器使用存储的指针指向每个 API 函数，以代表使用每个函数。

API : Base Kicad 插件

Base Kicad 插件由文件“kicad_plugin.h”定义。此必须包含在所有其他插件的声明中；例如，参考文件“3d_plugin.h”中的 3D 插件声明。一些函数的原型在《plugin-classes（插件-Plugin Classes（插件）中要描述。API 由“pluginldr.cpp”中定义的基本插件加载器

为了帮助理解基本 KiCad 插件所需的功能，我必须看看基本插件 Loader 中发生的情况。Plugin Loader 声明了一个虚函数“Open()”，它接受要加载的插件的完整路径。在特定的插件加载器中“Open()”函数最初将基本插件加载器的受保护的“open()”函数；一个基“open()”函数找到每个必需的基本插件函数的地址；一旦索引到每个函数的地址，就会执行一些

1. 用插件“GetKicadPluginClass()”，并将结果与插件加载器提供的插件字符串行比较；如果一些字符串不匹配，打开的插件不适用于 Plugin Loader 例。
2. 用插件“GetClassVersion()”来索引插件的插件 API 版本。
3. 插件加载器虚“GetLoaderVersion()”函数被用以索引由加载器的插件 API 版本。
4. 插件和加载器报告的插件 API 版本必须具有相同的主版本号，否则它被认为是不兼容的。是最基本的版本它由基本插件加载器执行。
5. 使用插件加载器的插件 API 版本信息用插件“CheckClassVersion()”；如果插件支持特定版本，返回“true”表示成功。如果成功，加载器根据“GetKicadPluginName()”和“GetPluginVersion()”的结果建立一个 PluginInfo 字符串，插件加载器在 Plugin Loader 的“Open()”中

API : 3D 插件

3D 插件由文件“3d_plugin.h”声明，它展示了所需的插件函数，如《class-plugin-3d-插件-3d），Plugin Class（插件-PLUGIN_3D（插件_3D）》中所述。相关的插件加载器在“pluginldr3D.cpp”中定义，除了所需的 API 函数之外，加载器定义了以下公共函数：

```

/* 打开完整路径 "aFullFileName" 指定的插件 */
/* Open the plugin specified by the full path "aFullFileName" */
bool Open( const wxString& aFullFileName );

/* 关闭当前打开的插件 */
/* Close the currently opened plugin */
void Close( void );

/* 检索插件加载器实现的插件 API 版本 */
/* Retrieve the Plugin Class API Version implemented by this Plugin Loader */
void GetLoaderVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Revision, unsigned char* Patch ) const;

```

所需的 3D 插件功能通以下功能公开：

```

/* 如果没有加插件, 返回插件或 NULL */
/* returns the Plugin Class or NULL if no plugin loaded */
char const* GetKicadPluginClass( void );

/* 如果没有加插件, 返回 FALSE */
/* returns false if no plugin loaded */
bool GetClassVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/* 如果版本号丢失或未加插件, 返回 FALSE */
/* returns false if the class version check fails or no plugin is loaded */
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
    unsigned char Patch, unsigned char Revision );

/* 返回插件名称, 如果没有加插件, 返回 NULL */
/* returns the Plugin Name or NULL if no plugin loaded */
const char* GetKicadPluginName( void );

/*
    如果未加任何插件, 返回 False, 否则返回参数。
    包含 GetPluginVersion() 的结果。
*/
/*
    returns false if no plugin is loaded, otherwise the arguments
    contain the result of GetPluginVersion()
*/
bool GetVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/*
    如果没有加插件, 将 aPluginInfo 置空字符串,
    否则将 aPluginInfo 置以下形式的字符串:
    [名称]:[主要].[次要].[修程序].[修订版本]其中。
    name = 由 GetKicadPluginClass() 提供的名称。
    主要、次要、修程序、修订=版本信息来自。
    GetPluginVersion()。
*/
/*
    sets aPluginInfo to an empty string if no plugin is loaded,
    otherwise aPluginInfo is set to a string of the form:
    [NAME]:[MAJOR].[MINOR].[PATCH].[REVISION] where
    NAME = name provided by GetKicadPluginClass()
    MAJOR, MINOR, PATCH, REVISION = version information from
    GetPluginVersion()
*/
void GetPluginInfo( std::string& aPluginInfo );

```

在典型情况下, 用将行以下操作:

1. 建 'KICAD_PLUGIN_LDR_3D' 的例。
2. 用 'Open("/path/to/myplugin.so")' 来打开一个特定的插件。必返回以确保根据需要加插件。
3. 用由 'KICAD_PLUGIN_LDR_3D' 公开的任何 3D 插件用。
- 4.

用 'Close()' 来关闭（取消连接）插件。

5. 例：'KICAD_PLUGIN_LDR_3D' 例。

场景 API

Scenegraph API 由 'ifsg_all.h' 及其包含的库定义。API 由多个辅助例程组成，命名空间 'S3D'，在 'ifsg_api.h' 中定义，包装器由各种 'ifsg_*.h' 定义；包装器支持底层的场景，它一起形成一个与 VRML2.0 静态场景兼容的场景结构。结构，及其公共函数如下：

sg_version.h

```
/*
   定义 SceneGraph 库的版本信息。
   所有使用 Scenegraph 库的插件都必须包含这个头文件
   并对照所报告的版本库版本信息。
   S3D::GetLibVersion() 以确保兼容性。
*/
/*
   Defines version information of the SceneGraph Classes.
   All plugins which use the scenegraph class should include this header
   and check the version information against the version reported by
   S3D::GetLibVersion() to ensure compatibility
*/

#define KICADSG_VERSION_MAJOR      2
#define KICADSG_VERSION_MINOR     0
#define KICADSG_VERSION_PATCH     0
#define KICADSG_VERSION_REVISION  0
```

sg_types.h

```
/*
    定义 SceneGraph 数据类型；有些类型。
    与 VRML2.0 数据类型密切相关。
*/
/*
    Defines the SceneGraph Class Types; these types
    are closely related to VRML2.0 node types.
*/

namespace S3D
{
    enum SGTYPES
    {
        SGTYPE_TRANSFORM = 0,
        SGTYPE_APPEARANCE,
        SGTYPE_COLORS,
        SGTYPE_COLORINDEX,
        SGTYPE_FACESET,
        SGTYPE_COORDS,
        SGTYPE_COORDINDEX,
        SGTYPE_NORMALS,
        SGTYPE_SHAPE,
        SGTYPE_END
    };
};
```

‘sg_base.h’ 包含 scenegraph 使用的基本数据类型声明。

```

/*
   是相当于 VRML2.0 的 RGB 色模型。
   RGB 模型，其中每种色可能在。
   范 [0..1]。
*/
/*
   This is an RGB color model equivalent to the VRML2.0
   RGB model where each color may have a value within the
   range [0..1].
*/

class SGCOLOR
{
public:
    SGCOLOR();
    SGCOLOR( float aRVal, float aGVal, float aBVal );

    void GetColor( float& aRedVal, float& aGreenVal, float& aBlueVal ) const;
    void GetColor( SGCOLOR& aColor ) const;
    void GetColor( SGCOLOR* aColor ) const;

    bool SetColor( float aRedVal, float aGreenVal, float aBlueVal );
    bool SetColor( const SGCOLOR& aColor );
    bool SetColor( const SGCOLOR* aColor );
};

class SGPOINT
{
public:
    double x;
    double y;
    double z;

public:
    SGPOINT();
    SGPOINT( double aXVal, double aYVal, double aZVal );

    void GetPoint( double& aXVal, double& aYVal, double& aZVal );
    void GetPoint( SGPOINT& aPoint );
    void GetPoint( SGPOINT* aPoint );

    void SetPoint( double aXVal, double aYVal, double aZVal );
    void SetPoint( const SGPOINT& aPoint );
};

/*
   SGVECTOR 有 3 个分量(x, y, z)似于一个点；但是。
   向量确保存的 是 范化的，并且。
   防止直接操作 件 量。
*/
/*
   A SGVECTOR has 3 components (x,y,z) similar to a point; however
   a vector ensures that the stored values are normalized and
   prevents direct manipulation of the component variables.
*/

class SGVECTOR
{
public:

```


'IFSG_NODE' 是所有场景点的基类。所有 scenegraph 对象都继承此基类的公共函数，但在某些情况下，特定函数可能对特定对象没有意义。

```

class IFSG_NODE
{
public:
    IFSG_NODE();
    virtual ~IFSG_NODE();

    /**
     * 功能
     * 除此包装器持有的 Scenegraph 对象。
     */
    /**
     * Function Destroy
     * deletes the scenegraph object held by this wrapper
     */
    void Destroy( void );

    /**
     * 函数附加。
     * 将给定 SGNODE* 与此包装器关联
     */
    /**
     * Function Attach
     * associates a given SGNODE* with this wrapper
     */
    virtual bool Attach( SGNODE* aNode ) = 0;

    /**
     * 函数 NewNode。
     * 创建与此包装器关联的新点。
     */
    /**
     * Function NewNode
     * creates a new node to associate with this wrapper
     */
    virtual bool NewNode( SGNODE* aParent ) = 0;
    virtual bool NewNode( IFSG_NODE& aParent ) = 0;

    /**
     * 函数 GetRawPtr()。
     * 返回原始内部SGNODE指针
     */
    /**
     * Function GetRawPtr()
     * returns the raw internal SGNODE pointer
     */
    SGNODE* GetRawPtr( void );

    /**
     * 函数 GetNodeType。
     * 返回此点例的类型。
     */
    /**
     * Function GetNodeType
     * returns the type of this node instance
     */
    S3D::SGTYPES GetNodeType( void ) const;

    /**
     * 函数 GetParent。
     * 返回指向此对象的父 SGNODE 的指针

```

‘IFSG_TRANSFORM’ 类似于 VRML2.0 Transform 点；它可以包含任意数量的子 IFSG_SHAPE 和 IFSG_TRANSFORM 点以及任意数量的引用的 IFSG_SHAPE 和 IFSG_TRANSFORM 点。有效的场景必须有一个“IFSG_TRANSFORM” 象作根。

ifsg_transform.h

```
/**
 * IFSG_Transform 点
 * 是 VRML 兼容 SCENEGRAPH 的包装。
 */
/**
 * Class IFSG_TRANSFORM
 * is the wrapper for the VRML compatible TRANSFORM block class SCENEGRAPH
 */

class IFSG_TRANSFORM : public IFSG_NODE
{
public:
    IFSG_TRANSFORM( bool create );
    IFSG_TRANSFORM( SGNODE* aParent );

    bool SetScaleOrientation( const SGVECTOR& aScaleAxis, double aAngle );
    bool SetRotation( const SGVECTOR& aRotationAxis, double aAngle );
    bool SetScale( const SGPOINT& aScale );
    bool SetScale( double aScale );
    bool SetCenter( const SGPOINT& aCenter );
    bool SetTranslation( const SGPOINT& aTranslation );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */
};
```

‘IFSG_SHAPE’ 类似于 VRML2.0 Shape 点；它必须包含一个子点或引用 FACESET 点，并且可以包含一个子点或引用 APPEARANCE 点。

ifsg_shape.h

```
/**
 * IFSG_SHAP E
 * 是 SGSHAPE 的包装。
 */
/**
 * Class IFSG_SHAPE
 * is the wrapper for the SGSHAPE class
 */

class IFSG_SHAPE : public IFSG_NODE
{
public:
    IFSG_SHAPE( bool create );
    IFSG_SHAPE( SGNODE* aParent );
    IFSG_SHAPE( IFSG_NODE& aParent );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */
};
```

‘IFSG_APPEARANCE’ 类似于 VRML2.0 Appearance 点，但目前它只代表包含 Material 点的 Appearance 点的等价物。

```

class IFSG_APPEARANCE : public IFSG_NODE
{
public:
    IFSG_APPEARANCE( bool create );
    IFSG_APPEARANCE( SGNODE* aParent );
    IFSG_APPEARANCE( IFSG_NODE& aParent );

    bool SetEmissive( float aRVal, float aGVal, float aBVal );
    bool SetEmissive( const SGCOLOR* aRGBColor );
    bool SetEmissive( const SGCOLOR& aRGBColor );

    bool SetDiffuse( float aRVal, float aGVal, float aBVal );
    bool SetDiffuse( const SGCOLOR* aRGBColor );
    bool SetDiffuse( const SGCOLOR& aRGBColor );

    bool SetSpecular( float aRVal, float aGVal, float aBVal );
    bool SetSpecular( const SGCOLOR* aRGBColor );
    bool SetSpecular( const SGCOLOR& aRGBColor );

    bool SetAmbient( float aRVal, float aGVal, float aBVal );
    bool SetAmbient( const SGCOLOR* aRGBColor );
    bool SetAmbient( const SGCOLOR& aRGBColor );

    bool SetShininess( float aShininess );
    bool SetTransparency( float aTransparency );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       外点，并始返回失败代码
    /* the following functions make no sense within an
       appearance node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
    */
};

```

‘IFSG_FACESET’ 类似于包含 IndexedFaceSet 点的 VRML2.0 Geometry 点。它必包含个子点或引用 COORDS 点，个子 COORDINDEX 点以及个子点或引用 NORMALS 点；另外可能有一个子点或引用 COLORS 点。提供化的法计算功能以帮助用将正常分配表面。与 VRML2.0 模的偏差如下：

1. 法始是每个点。
2. 色是每个点。
3. 坐索引集必描述三角形面。

ifsg_faceset.h

```
/**
 * IFSG_FACESET 类
 * 是 SGFACESET 类的包装。
 */
/**
 * Class IFSG_FACESET
 * is the wrapper for the SGFACESET class
 */

class IFSG_FACESET : public IFSG_NODE
{
public:
    IFSG_FACESET( bool create );
    IFSG_FACESET( SGNODE* aParent );
    IFSG_FACESET( IFSG_NODE& aParent );

    bool CalcNormals( SGNODE** aPtr );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */
};
```

ifsg_coords.h

```
/**
 * IFSG_COORDS 类
 * 是 SGCORDS 的包装器。
 */
/**
 * Class IFSG_COORDS
 * is the wrapper for SGCORDS
 */

class IFSG_COORDS : public IFSG_NODE
{
public:
    IFSG_COORDS( bool create );
    IFSG_COORDS( SGNODE* aParent );
    IFSG_COORDS( IFSG_NODE& aParent );

    bool GetCoordsList( size_t& alistSize, SGPOINT*& aCoordsList );
    bool SetCoordsList( size_t alistSize, const SGPOINT* aCoordsList );
    bool AddCoord( double aXValue, double aYValue, double aZValue );
    bool AddCoord( const SGPOINT& aPoint );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       点开始并返回失败代码
       the following functions make no sense within a
       coords node and always return a failure code

       bool AddRefNode( SGNODE* aNode );
       bool AddRefNode( IFSG_NODE& aNode );
       bool AddChildNode( SGNODE* aNode );
       bool AddChildNode( IFSG_NODE& aNode );
    */
};
```

‘IFSG_COORDINDEX’ 类似于 VRML2.0 coordIdx[] 集，除了它必须描述三角形面，这意味着索引的数可以被 3 整除。

```

/**
 * IFSG_COORDINDEX 类
 * 是 SGCORDINDEX 的包装。
 */
/**
 * Class IFSG_COORDINDEX
 * is the wrapper for SGCORDINDEX
 */

class IFSG_COORDINDEX : public IFSG_INDEX
{
public:
    IFSG_COORDINDEX( bool create );
    IFSG_COORDINDEX( SGNODE* aParent );
    IFSG_COORDINDEX( IFSG_NODE& aParent );

    bool GetIndices( size_t& nIndices, int*& aIndexList );
    bool SetIndices( size_t nIndices, int* aIndexList );
    bool AddIndex( int aIndex );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       点开始并返回失败代码
       the following functions make no sense within a
       coordindex node and always return a failure code

       bool AddRefNode( SGNODE* aNode );
       bool AddRefNode( IFSG_NODE& aNode );
       bool AddChildNode( SGNODE* aNode );
       bool AddChildNode( IFSG_NODE& aNode );
    */
};

```

‘IFSG_NORMALS’ 相当于 VRML2.0 Normals 点。

ifsg_normals.h

```
/**
 * IFSG_Normal 类
 * 是 SGNORMALS 类的包装。
 */
/**
 * Class IFSG_NORMALS
 * is the wrapper for the SGNORMALS class
 */

class IFSG_NORMALS : public IFSG_NODE
{
public:
    IFSG_NORMALS( bool create );
    IFSG_NORMALS( SGNODE* aParent );
    IFSG_NORMALS( IFSG_NODE& aParent );

    bool GetNormalList( size_t& aListSize, SGVECTOR*& aNormalList );
    bool SetNormalList( size_t aListSize, const SGVECTOR* aNormalList );
    bool AddNormal( double aXValue, double aYValue, double aZValue );
    bool AddNormal( const SGVECTOR& aNormal );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       法点并始返回失败代码
       the following functions make no sense within a
       normals node and always return a failure code

       bool AddRefNode( SGNODE* aNode );
       bool AddRefNode( IFSG_NODE& aNode );
       bool AddChildNode( SGNODE* aNode );
       bool AddChildNode( IFSG_NODE& aNode );
    */
};
```

‘IFSG_COLORS’ 类似于 VRML2.0 colors[] 集。

ifsg_colors.h

```
/**
 * IFSG_COLOR 类
 * 是 SGCOLORS 的包装器。
 */
/**
 * Class IFSG_COLORS
 * is the wrapper for SGCOLORS
 */

class IFSG_COLORS : public IFSG_NODE
{
public:
    IFSG_COLORS( bool create );
    IFSG_COLORS( SGNODE* aParent );
    IFSG_COLORS( IFSG_NODE& aParent );

    bool GetColorList( size_t& aListSize, SGCOLOR*& aColorList );
    bool SetColorList( size_t aListSize, const SGCOLOR* aColorList );
    bool AddColor( double aRedValue, double aGreenValue, double aBlueValue );
    bool AddColor( const SGCOLOR& aColor );

    /* 此未示的各种基函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       法点并始返回失代
       /* the following functions make no sense within a
          normals node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
    */
};
```

其余的 API 函数在 'ifsg_api.h' 中定义如下：

```

namespace S3D
{
    /**
     * 函数 GetLibVersion 函数。
     * kicad_3dsg 库
     */
    /**
     * Function GetLibVersion retrieves version information of the
     * kicad_3dsg library
     */
    SGLIB_API void GetLibVersion( unsigned char* Major, unsigned char* Minor,
                                unsigned char* Patch, unsigned char* Revision );

    //从 SGNODE 指针提取信息的函数
    // functions to extract information from SGNODE pointers
    SGLIB_API S3D::SGTYPES GetSGNodeType( SGNODE* aNode );
    SGLIB_API SGNODE* GetSGNodeParent( SGNODE* aNode );
    SGLIB_API bool AddSGNodeRef( SGNODE* aParent, SGNODE* aChild );
    SGLIB_API bool AddSGNodeChild( SGNODE* aParent, SGNODE* aChild );
    SGLIB_API void AssociateSGNodeWrapper( SGNODE* aObject, SGNODE** aRefPtr );

    /**
     * 函数 CalcTriNorm
     * 返回点 p1, p2, p3 描述的三角形的法向量。
     */
    /**
     * Function CalcTriNorm
     * returns the normal vector of a triangle described by vertices p1, p2, p3
     */
    SGLIB_API SGVECTOR CalcTriNorm( const SGPOINT& p1, const SGPOINT& p2, const SGPOINT& p3
    );

    /**
     * 函数 WriteCache
     * 将 SGNODE 写入二进制缓存文件。
     *
     * @param aFileName 是要写入的文件的名称。
     * @param overwrite 必须置为 true 才能覆盖已有文件。
     * @param 阳极是要写入的点中的任何点。
     * @return 成功返回 true。
     */
    /**
     * Function WriteCache
     * writes the SGNODE tree to a binary cache file
     *
     * @param aFileName is the name of the file to write
     * @param overwrite must be set to true to overwrite an existing file
     * @param aNode is any node within the node tree which is to be written
     * @return true on success
     */
    SGLIB_API bool WriteCache( const char* aFileName, bool overwrite, SGNODE* aNode,
                              const char* aPluginInfo );

    /**
     * 函数 ReadCache
     * 读取二进制缓存文件并构建 SGNODE 树
     *
     * @param aFileName 是要读取的二进制缓存文件的名称。
     * @return 失败返回 NULL, 成功返回 SCENEGRAPH 点指针;
     * 如果需要, 此点可以通过以下方式与 IFSG_Transform 包装器关联
     */

```

有关 Scenegraph API 的使用示例, 参看上面的《advanced-3d-plugin (高3d-插件), Advanced 3D Plugin tutorial (高3D 插件教程)》, 以及 KiCad VRML1, VRML2 和 X3D 解析器。