KiCad 插件

Table of Contents

KiCad 插件系□

版□

□献者

Cirilo Bernardo

翻□人□

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

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

反□

□将任何□□□告、建□或新版本引□到此□:

- □于 KiCad 文档: https://gitlab.com/kicad/services/kicad-doc/issues
- □于 KiCad □件: https://gitlab.com/kicad/code/kicad/issues
- □于 KiCad □件 i18n: https://gitlab.com/kicad/code/kicad-i18n/issues

出版日期和□件版本

2016 年 1 月 29 日出版。

KiCad 插件系□□介

KiCad 插件系□是一个使用共享□□展 KiCad 功能的框架。 使用插件的一个主要□点是在开□插件□没有必要重建 KiCad 套件; 事□上，可以借助 KiCad 源代□□中的一小□□□构建插件。 通□确保开□人□□□□与正在开□的插件直接相□的代□□从而减少每个构建和□□周期所需的□□□在插件开□期□□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□看器开□的，因此可以支持更多□型的 3D 模型，而无需□支持的每种新模型□型的 KiCad 源□行重大更改。 插件框架后来被推广，以便将来开□人□可以□建不同□型的插件。 目前，只有 3D 插件在 KiCad 中□□□但可以想象最□将开□ PCB 插件，以使用□能□□□数据□入器和□出器。

插件□

插件分□插件□□因□每个插件都解决了特定域中的□□□因此需要□域独有的接口。 例如，3D 模型插件从文件加□ 3D 模型数据并将□数据□□□可由 3D □看器□示的格式。 PCB □入/□出插件将□取 PCB 数据并□出□其他□气或机械数据格式，或将外部格式□□□ KiCad PCB。 目前只开□了 3D 插件□□它将成□本文档的重点。

□□插件□需要在 KiCad 源代□□中□建代□来管理插件代□的加□□在 KiCad 源代□□中，文件 ‘plugins/ldr/pluginldr.h’ 声明了所有插件加□器的基□□个□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□器和可用插件之□的版本兼容性的基本□□□□□ ‘plugins/ldr/3d/pluginldr3D.h’ 声明了 3D 插件□的加□器。加□器□□加□□定的插件并使其功能可用于 KiCad。插件加□器的每个□例代表一个□□的插件□□□并充当 KiCad 和插件功能之□的透明□梁。加□器不是 KiCad 中支持插件所需的唯一代□□我□□需要代□来□□插件和代□□以通□插件加□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□新的插件□□否□插件开□人□不需要□心 KiCad 管理插件的内部代□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; □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，□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 );

/*
   如果插件中了版本, 返回 true。
   确定定的插件 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’ 声明了必由所有 3D 插件的函数，并定义了插件所需的多函数以及用不得重新的函数。 用不得重新的已定义函数是：

```
/* 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 );

/*
   行由的开人制行的基本版本。
   PLUGIN_3D 的加器，如果。
   通
   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 );
```

用必的功能如下：

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

/*
   返回□求的□展字符串；有效□□ 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□□器□数 */
/* Return the total number of file filters supported by the plugin */
int GetNFilters( void );

/*
   返回□求的文件□□器；有效□□ 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□种□型的 3D 模型，□返回 true。
    在某些情况下，插件可能尚未提供可□模型。
    并且必□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□指定的模型并返回指向其可□化模型数据的指□ */
/* Load the specified model and return a pointer to its visual model data */
SCENEGRAPH* Load( char const* aFileName );
```

教程：3D插件□

本□包含□ PLUGIN_3D □的两个非常□□的插件的描述，并引□用□完成代□的□置和构建。

基本的 3D 插件

本教程将引□用□开□一个名□ “PLUGIN_3D_DEMO1” 的非常基本的 3D 插件。 本教程的目的只是□了演示一个非常基本的 3D 插件的构造，除了提供一些允□ KiCad 用□在□□ 3D 模型□□□文件名的□□字符串之外什么都不做。 □里演示的代□是任何 3D 插件的□□最低要求，可以用作□建更高□插件的模板。

□了构建演示□目，我□需要以下内容：

- CMake
-

KiCad 插件

- KiCad 景 ‘kicad_3dsg’

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

要开始，让我建一个目目和 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* 的计算结果为 “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_VE
RSION_REVISION} )
        unset( KICAD_SGVERSION CACHE )

    endif()
endif()
```

必□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' □境□量□置□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□目名称
# declare the name of the project
project( PLUGIN_DEMO )

# □□我□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 □□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□ 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□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

// □□□用□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";
}

// □□□用□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□展名数量；在 *NIX 系□上，□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□□器集数量
// number of filter sets supported
#define NFILS 5

// 定义此□展字符串和□□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□足□□ 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□展名和文件□展名□□器列表，以增□ 3D 模型文件□□□□框。 在 KiCad 中，□展字符串用于□□可用于加□指定模型的插件; 例如，如果插件是 ‘wrl’，那么 KiCad 将□用声称支持□展 ‘wrl’ 的每个插件，直到插件返回可□化数据。 每个插件提供的文件□□器将□□到 3D 文件□□器□□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; 如果要加□插件，必□将插件文件□制到 KiCad 的插件目□□

高□ 3D 插件

本教程将引□用□开□名□ “PLUGIN_3D_DEMO2” 的 3D 插件。 本教程的目的是演示 KiCad □□器可以渲染的非常基本的□景□的构造。 □插件声称□理 ‘txt’ □型的文件。 □然文件必□存在，以便□存管理器□用插件，但此插件不□理文件内容; 相反，插件只是□建一个包含一□四面体的□景□□ 本教程假定第一个教程已完成，并且已□建 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□中，我□将□展上一个教程的 CMakeLists.txt 文件来构建本教程。 由于□个插件会□ KiCad □建一个□景□□我□需要□接到 KiCad 的□景□□ ‘kicad_3dsg’。 KiCad 的□景□□提供了一□可用于构建□景□□象的□; □景□□象是 3D □存管理器使用的中□数据可□化格式。 所有支持模型可□化的插件都必□通□此□将模型数据□□□□景□□

第一步是□展 ‘CMakeLists.txt’ 来构建□个教程□目：

```
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□到源目□并□建源文件：

```
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 □景□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□展数量
// number of extensions supported
#ifdef _WIN32
#define NEXTS 1
#else
#define NEXTS 2
#endif

// 支持的□□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□程接口（API）

插件通□□用程序□程接口（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□于 KiCad 源代□□中定义的其他 API; 在 3D 插件的情况下，支持模型可□化的所有插件必□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□□□□文件 ‘kicad_plugin.h’ 中声明的一□基本函数; □些声明称□ Base Kicad 插件□□ 不存在 Base Kicad 插件□的□□; □文件的存在□粹是□了确保插件开□人□在每个插件□□中□□□些定义的函数。

在 KiCad 中，插件加□器的每个□例都□□了插件提供的 API，就像插件加□器是提供插件服□的□一□□ □是通□ Plugin 加□器□□□的，□□提供包含与插件□□的□似的函数名的公共接口; 如果例如没有加□插件，□参数列表可以□化以适□向用□通知可能遇到的任何□□的需要。 在内部，插件加□器使用存□的指□指向每个 API 函数，以代表用□□用每个函数。

API：Base Kicad 插件□

Base Kicad 插件□由□文件 ‘kicad_plugin.h’ 定义。 此□□必□包含在所有其他插件□的声明中; 例如，□参□□文件 ‘3d_plugin.h’ 中的 3D 插件□声明。 □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□器□□□

□了帮助理解基本 KiCad 插件□所需的功能，我□必□□看基本插件 Loader □中□生的情况。 Plugin Loader □声明了一个虚函数 ‘Open()’，它接受要加□的插件的完整路径。 在特定的插件□加□器中□□ ‘Open()’ 函数最初将□用基本插件加□器的受保□的 ‘open()’ 函数; □个基□ ‘open()’ 函数□□找到每个必需的基本插件函数的地址; 一旦□索到每个函数的地址，就会□制□行一些□□□

1. □用插件 ‘GetKicadPluginClass()’，并将□果与插件加□器□□提供的插件□字符串□行比□; 如果□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□索插件□□的插件□ API 版本。
3. 插件加□器虚□ ‘GetLoaderVersion()’ 函数被□用以□索由加□器□□的插件□ API 版本。
4. 插件和加□器□告的插件□ API 版本必□具有相同的主版本号，否□它□被认□是不兼容的。 □是最基本的版本□□□它由基本插件加□器□制□行。
5. 使用插件加□器的插件□ API 版本信息□用插件 ‘CheckClassVersion()’; 如果插件支持□定版本，□返回 “true” 表示成功。 如果成功，加□器根据 ‘GetKicadPluginName()’ 和 ‘GetPluginVersion()’ 的□果□建一个 PluginInfo 字符串，插件加□□程在 Plugin Loader 的 ‘Open()’ □□中□□□

API：3D 插件□

3D 插件□由□文件 ‘3d_plugin.h’ 声明，它□展了所需的插件函数，如《class-plugin-3d□□-插件-3d），Plugin Class（插件□□□PLUGIN_3D（插件_3D）》中所述。 相□的插件加□器在 ‘pluginldr3D.cpp’ 中定义，除了所需的 API 函数之外，加□器□□□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 );

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

/* □索□插件加□器□□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□□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□返回参数。
   包含 GetPluginVersion() 的□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□，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□□□取消□接）插件。

5. □□ ‘KICAD_PLUGIN_LDR_3D’ □例。

□景□□ API

Scenegraph □ API 由□□ ‘ifsg_all.h’ 及其包含的□□定义。 API 由□多□助例程□成，命名空□□ ‘S3D’，在 ‘ifsg_api.h’ 中定义，包装□由各种 ‘ifsg_*.h’ □□定义; 包装器支持底□的□景□□□它□一起形成一个与 VRML2.0 静□□景□兼容的□景□□构。 □□□□构，□及其公共函数如下：

sg_version.h

```
/*
   定义 SceneGraph □的版本信息。
   所有使用 Scenegraph □的插件都□□包含□个□。
   并□照所□告的版本□□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 型；些型。
   与 VRML2.0 点型密切相。
*/
/*
   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 使用的基本数据型的声明。

sg_base.h

```
/*
   □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’ 是所有景点的基 所有 scenegraph 象都此的公共函数，但在某些情况下，特定函数可能特定没有意义。

ifsg_node.h

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

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

    /**
     * 函数附加。
     * 将□定S GNODE* 与此包装器□□。
     */
    /**
     * Function Attach
     * associates a given SGNODE* with this wrapper
     */
    virtual bool Attach( SGNODE* aNode ) = 0;

    /**
     * 函数 NewNode。
     * □建与此包装器□□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□。
     */
    /**
     * Function GetRawPtr()
     * returns the raw internal SGNODE pointer
     */
    SGNODE* GetRawPtr( void );

    /**
     * 函数 GetNodeType。
     * 返回此□点□例的□型。
     */
    /**
     * Function GetNodeType
     * returns the type of this node instance
     */
    S3D::SGTYPES GetNodeType( void ) const;

    /**
     * 函数 GetParent。
     * 返回指向此□象的父 SGNODE 的指□。
```

‘IFSG_TRANSFORM’ □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□景□必□有一个 “IFSG_TRANSFORM” □象作□根。

ifsg_transform.h

```
/**
 * IFSG_Transform □。
 *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’ □似于 VRML2.0 Shape □点; 它必□包含□个子□点或引用 FACESET □点，并且可以包含□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’ 似于 VRML2.0 Appearance 点，但目前它只代表包含 Material 点的 Appearance 点的等价物。

ifsg_appearance.h

```
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□代□
    /* 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’□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 。
 * 是 SGFACESET 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 □□。
 * 是 SGCOORDS 的包装器。
 */
/**
 * Class IFSG_COORDS
 * is the wrapper for SGCOORDS
 */

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。
       □□□点并始□返回失□代□
    /* 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’□似于 VRML2.0 coordIdx[] 集，除了它必□□□描述三角形面，□意味着索引的□数可以被 3 整除。

ifsg_coordindex.h

```
/**
 * IFSG_COORDINDEX 。
 * 是 SGCOORDINDEX 的包装。
 */
/**
 * Class IFSG_COORDINDEX
 * is the wrapper for SGCOORDINDEX
 */

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。
       点并始 返回失 代
    /* 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 。
 * 是 SGNORMALS 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 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 );
     */
};
```

‘IFSG_COLORS’ 似于 VRML2.0 colors[] 集。

ifsg_colors.h

```
/**
 * IFSG_COLOR □。
 * 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：

ifsg_api.h

```
namespace S3D
{
    /**
     * 函数 GetLibVersion 索。
     * kicad_3dsg 。
     */
    /**
     * 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 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 制 存文件。
     *
     * @param aFileName 是要写入的文件的名称。
     * @param overwrite 必 置 true 才能覆盖 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
     * 取二 制 存文件并 建 SGNODE 。
     *
     * @param aFileName 是要 取的二 制 存文件的名称。
     * @return 失 返回 NULL，成功返回 SCENEGRAPH 点指 ；
     * 如果需要，此 点可以通 以下方式与 IFSG_Transform 包装器 。
```

有□ Scenegraph API 的□□使用示例，□参□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。