

# 北航数字导航中心

## C/C++源代码及其文档书写规范



北航数字导航中心

2009 年 05 月

# 目 录

<b>1 前言</b>	<b>- 1 -</b>
<b>2 DNC 前缀和名空间</b>	<b>- 1 -</b>
2.1 C 代码使用 DNC 做前缀	- 1 -
2.2 C++代码使用 DNC 做名空间	- 1 -
<b>3 C/C++源代码注释及书写规范</b>	<b>- 2 -</b>
3.1 C/C++源代码注释	- 2 -
3.1.1 总体注释要求	- 3 -
3.1.2 说明性文件头部的注释	- 3 -
3.1.3 源文件头部的注释	- 4 -
3.1.4 函数头部的注释	- 5 -
3.1.5 对于 if、while、do 等其大括号内嵌代码块的注释	- 5 -
3.1.6 某个代码段的注释	- 6 -
3.1.7 注释的位置	- 6 -
3.1.8 对于所有物理含义的变量、常量及宏的注释	- 7 -
3.1.9 数据结构声明(包括数组、结构、类、枚举等)的注释	- 7 -
3.1.10 全局变量的注释	- 7 -
3.1.11 注释与所描述内容缩进的要求	- 8 -
3.2 C/C++源代码注释书写与帮助文档的自动生成	- 8 -
3.2.1 块注释	- 8 -
3.2.2 行注释	- 9 -
3.2.3 函数的摘要	- 9 -
3.2.4 函数的形参	- 9 -
3.2.5 函数的返回值	- 9 -
3.2.6 函数的注意事项 (Remark)	- 10 -
3.2.7 补充说明	- 10 -
3.3 C/C++代码书写规范	- 10 -
3.3.1 每行代码长度	- 10 -

3.3.2 合并行的问题 .....	- 11 -
3.3.3 指针中*号的位置 .....	- 11 -
3.3.4 全局函数的调用 .....	- 12 -
3.3.5 关于 if...else if .....	- 12 -
3.3.6 与“{”、“}”有关的规定 .....	- 12 -
3.3.6.1 与“{”、“}”有关代码的缩进格式 .....	- 12 -
3.3.6.2 “{”、“}”的省略 .....	- 12 -
3.3.7 与空格有关的各项规定 .....	- 13 -
3.3.7.1 对两目、三目运算符两边的空格的规定 .....	- 13 -
3.3.7.2 对 for、while、if 等关键字之后空格的规定 .....	- 13 -
3.3.7.3 对调用函数、宏时的空格规定 .....	- 13 -
3.3.7.4 对类型强制转换时的空格规定 .....	- 14 -
3.3.8 代码缩进/对齐使用空格的规定 .....	- 14 -
3.3.8.1 代码缩进一个 4 个空格 .....	- 14 -
3.3.8.2 代码不缩进情况 .....	- 14 -
3.3.9 与类相关的.h 文件与.cpp 文件 .....	- 15 -
3.3.9.1 类的定义与实现 .....	- 15 -
3.3.9.2 类的合并 .....	- 15 -
3.3.10 对变量的要求 .....	- 15 -
3.3.10.1 对变量的要求 .....	- 15 -
3.3.10.2 对变量类型的要求 .....	- 16 -
3.3.11 对结构的要求 .....	- 16 -
3.3.11.1 结构的功能要单一 .....	- 16 -
3.3.11.2 不同结构间的关系要简单 .....	- 17 -
3.3.12 函数的编写要求 .....	- 18 -
3.3.12.1 函数编写的总体要求 .....	- 18 -
3.3.12.2 函数名的要求 .....	- 19 -
3.3.12.3 函数参数的要求 .....	- 19 -
3.3.12.4 函数功能的要求 .....	- 20 -

3.3.12.5 可重入函数编写 .....	21 -
3.3.13 C/C++程序效率的要求 .....	22 -
3.3.14 C/C++代码质量的要求 .....	23 -
3.3.14.1 代码质量保证优先原则 .....	23 -
3.3.14.2 对内存及句柄操作的要求 .....	23 -
3.3.14.3 其他要求 .....	25 -
3.3.15 C/C++代码测试与维护 .....	26 -
<b>4 C/C++源代码程序命名规范 .....</b>	<b>27 -</b>
4.1 总体命名要求 .....	27 -
4.2 具体命名规范 .....	28 -
4.2.1 工程名 .....	28 -
4.2.2 文件名 .....	28 -
4.2.3 函数名 .....	28 -
4.2.4 变量名 .....	28 -
4.2.5 类名 .....	29 -
4.2.6 结构、宏、枚举及联合的名字 .....	29 -
4.2.7 具有互斥意义的变量或相反动作的函数的命名 .....	29 -
4.2.8 对命名中下划线的使用规定 .....	30 -
<b>5 程序开发中相关文档撰写规定 .....</b>	<b>30 -</b>
5.1 程序设计与开发文档 .....	30 -
5.1.1 软件的开发背景 .....	30 -
5.1.2 软件的结构设计 .....	30 -
5.1.3 数据及资源文件说明 .....	31 -
5.1.3.1 数据用途 .....	31 -
5.1.3.2 数据格式 .....	31 -
5.1.4 程序结构设计与开发 .....	31 -
5.2 程序使用文档 .....	31 -
5.2.1 软件系统的程序组成 .....	31 -
5.2.2 开发环境要求、软件安装及环境配置 .....	31 -

5.2.3 软件系统的使用方法.....	- 34 -
5.3 程序测试文档.....	- 36 -
5.3.1 测试环境.....	- 37 -
5.3.2 软件测试项目和功能.....	- 37 -
5.3.3 软件测试数据及格式.....	- 37 -
5.3.4 软件测试过程.....	- 37 -
5.4 软件程序调试文档.....	- 37 -
<b>6 程序及文档备份.....</b>	<b>- 39 -</b>
6.1 源程序及其文档备份.....	- 39 -
6.2 第三方软件（开发包或源码）及其文档备份.....	- 39 -
<b>7 词语缩写表.....</b>	<b>- 39 -</b>
7.1 词语缩写的总体要求.....	- 39 -
7.2 通用词缩写表.....	- 39 -
7.3 专业词缩写表.....	- 44 -
<b>8 中心代码实例.....</b>	<b>- 46 -</b>
8.1 *.h 的编写实例.....	- 46 -
8.2 *.CPP 的编写实例.....	- 49 -

## 1 前言

本规范规定了 C/C++ 源代码及其相应文档的书写规范。

制定本规范的目的是统一北航数字导航中心研发部的 C/C++ 语言源代码的书写风格、注释内容和相应文档的撰写，以便于代码的阅读、维护、管理、修订及使用。

从制定完成即日起，程序员在编写 C/C++代码的时候，需要严格遵守制定的《C/C++源代码及其文档书写规范》。

此外，利用其他语言（如 VB、Delphi 和 Matlab 等）编写的程序的程序员同样要遵守本规范。

## 2 DNC 前缀和名空间

### 2.1 C 代码使用 DNC 做前缀

C 代码的接口函数一律使用 DNC(Digital Navigation Center,数字导航中心英文缩写)做前缀。

例如

//C 语言代码在函数前加 DNC\_前缀即可，例

```
int DNC_FunName();  
  
{  
    return 0;  
}
```

### 2.2 C++代码使用 DNC 做名空间

C++代码使用 DNC 做名空间。

具体示例如下

/\*\*.h 文件

```
namespace DNC  
{  
    class CDncDemo  
    {  
    public:
```

```
        CDncDemo();
        ~CDncDemo();
    };
}
/**.cpp 文件
namespace DNC
{
    CDncDemo::CDncDemo()
    {

    }

    CDncDemo::~~CDncDemo()
    {

    }
}

//在使用 CDncDemo 的文件中加入:
using namespace DNC;
//然后定义对象
CDncDemo dncobj;
//或者直接
DNC::CDncDemo dncobj;
```

### 3 C/C++源代码注释及书写规范

#### 3.1 C/C++源代码注释

除非极其简单，否则对函数应有注释说明。内容包括：功能、入口/出口参数，必要时还可有备注或补充说明。推荐采取特定的格式(见第 3.2 条)，以方便使用软件自动

生成帮助文档。

### 3.1.1 总体注释要求

- 一般情况下，源程序有效注释量必须在 20% 以上；
- 注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁；
- 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性，不再有用的注释要删除；
- 注释的内容要清楚、明了，含义准确，防止注释二义性，错误的注释不但无益反而有害；
- 除非必要，不应在代码或表达中间插入注释，否则容易使代码可理解性变差；
- 避免在注释中使用缩写，特别是非常用缩写，在使用缩写时或之前，应对缩写进行必要的说明；
- 注释格式尽量统一，建议使用“/\* ..... \*/”
- 建议多使用中文，除非能用非常流利准确的英文表达。

### 3.1.2 说明性文件头部的注释

说明性文件（如头文件.h 文件、.c/c++文件、.def 文件、编译说明文件.cfg 等）头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

```
/******
```

```
Copyright (C), 2008-2009, DNC BUAA
```

```
File name:    // 文件名
```

```
Author:      Version:      Date:      // 作者、版本及完成日期
```

```
Description:  // 用于详细说明此程序文件完成的主要功能，与其他模块  
              // 或函数的接口，输出值、取值范围、含义及参数间的控  
              // 制、顺序、独立或依赖等关系
```

```
Others:      // 其它内容的说明
```



Function List: // 主要函数列表，每条记录应包括函数名及功能简要说明

1. ....

History: // 修改历史记录列表，每条修改记录应包括修改日期、修改者及修改内容简述

1. Date:

Author:

Modification:

2.

\*\*\*\*\*/

### 3.1.3 源文件头部的注释

源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块目的/功能、主要函数及其功能、修改日志等。

示例：下面这段源文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

/\*\*\*\*\*\*

Copyright (C), 2008-2009, DNC BUAA

FileName: test.cpp

Author:      Version :      Date: // 作者、版本及完成日期

Description:    // 模块描述

Version:        // 版本信息

Function List: // 主要函数及其功能

1. -----

History:        // 历史修改记录

David    96/10/12    1.0    build this moudle

\*\*\*\*\*/

说明：Description 一项描述本文件的内容、功能、内部各部分之间的关系及本文件与其它文件关系等。History 是修改历史记录列表，每条修改记录应包括修改日期、修改者及修改内容简述。

### 3.1.4 函数头部的注释

函数头部应进行注释，列出：函数的目的/功能、输入参数、输出参数、返回值、调用关系（函数、表）等。

示例：下面这段函数的注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

```

/*****
Function:    // 函数名称
Description: // 函数功能、性能等的描述
Calls:      // 被本函数调用的函数清单
Called By:   // 调用本函数的函数清单
Table Accessed: // 被访问的表（此项仅对于牵扯到数据库操作的程序）
Table Updated: // 被修改的表（此项仅对于牵扯到数据库操作的程序）
Input:       // 输入参数说明，包括每个参数的作
              // 用、取值说明及参数间关系。
Output:      // 对输出参数的说明。
Return:      // 函数返回值的说明
Others:      // 其它说明
*****/

```

### 3.1.5 对于 if、while、do 等其大括号内嵌代码块的注释

比较长(需拖动滚动条来看)或者多层嵌套时，在结尾的“}”后要加上其所对应的语句的注释说明。

例1:

```

if (...)
{
    // program code
    while (index < MAX_INDEX)
    {
        // program code
    } /* end of while (index < MAX_INDEX) */ // 指明该条while语句结束

```

```
    } /* end of if (...)*/ // 指明是哪条if语句结束
```

例2:

```
BOOL DNC_SaveToFile(  
    const char cszFileName[],  
    BOOL bCanReplace /* = TRUE */);
```

### 3.1.6 某个代码段的注释

描述某个代码段的注释，可以给注释描述的代码段外围加上{}，帮助阅读。

例:

```
    //代码段实现功能或意图描述  
    {  
        代码段  
    }
```

### 3.1.7 注释的位置

注释应与其描述的代码相近，对代码的注释应放在其上方或右方相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

示例：如下例子不符合规范。

例 1:

```
/* get replicate sub system index and net indicator */
```

```
repssn_ind = ssn_data[index].repssn_index;
```

```
repssn_ni = ssn_data[index].ni;
```

例 2:

```
repssn_ind = ssn_data[index].repssn_index;
```

```
repssn_ni = ssn_data[index].ni;
```

```
/* get replicate sub system index and net indicator */
```

应如下书写

```
/* get replicate sub system index and net indicator */
```

```
repssn_ind = ssn_data[index].repssn_index;
repssn_ni = ssn_data[index].ni;
```

### 3.1.8 对于所有物理含义的变量、常量及宏的注释

对于所有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方。

```
/* active statistic task number */
```

```
#define MAX_ACT_TASK_NUMBER 1000
```

```
#define MAX_ACT_TASK_NUMBER 1000 /* active statistic task number */
```

### 3.1.9 数据结构声明(包括数组、结构、类、枚举等)的注释

数据结构声明(包括数组、结构、类、枚举等)，如果其命名不是充分自注释的，必须加以注释。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释放在此域的右方。

示例：可按如下形式说明枚举/数据/联合结构。

```
/* sccp interface with sccp user primitive message name */
```

```
enum DNC_USER_PRIMITIVE
```

```
{
```

```
    N_UNITDATA_IND; /* dnc notify dnc user unit data come */
```

```
    N_NOTICE_IND; /* dnc notify user the No.7 network can not */
```

```
                /* transmission this message */
```

```
    N_UNITDATA_REQ; /* dnc user's unit data transmission request*/
```

```
};
```

### 3.1.10 全局变量的注释

全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明。

示例：

```
/* The ErrorCode when DNC translate */
```

```
/* Global Title failure, as follows */ // 变量作用、含义
```

```

/* 0 — SUCCESS 1 — GT Table error */
/* 2 — GT error Others — no use */ // 变量取值范围
/* only function DNC_Translate() in */
/* this modual can modify it, and other */
/* module can visit it through call */
/* the function DNC_GetGTTransErrorCode() */ // 使用方法
BYTE g_GTTranErrorCode;

```

### 3.1.11 注释与所描述内容缩进的要求

注释与所描述内容进行同样的缩排，做到程序排版整齐，并方便注释的阅读与理解。

例如

```

void DNC_ExampleFun( void )
{
    /* code one comments */
    CodeBlock One
    /* code two comments */
    CodeBlock Two
}

```

## 3.2 C/C++源代码注释书写与帮助文档的自动生成

本节规定的 C/C++源代码注释书写规范主要是利用 Doxygen 工具自动生成帮助文档。对于那些不希望提取为帮助文档的注释，请参照本规范的 3.1 节。

### 3.2.1 块注释

[格式]

开头: /\*\*

结尾: \*/

例:

```
/**
```

```
* ... This is a block comment ...  
*/
```

### 3.2.2 行注释

[格式]

开头: ///

例:

```
/// This is a line comment...
```

### 3.2.3 函数的摘要

[格式]

在函数声明主体开头的注释块中加上控制符: @brief

例:

```
/**  
 * @brief 用于从输入流读取一个字符，不回显  
 */  
int getch(void);
```

### 3.2.4 函数的形参

[格式]

在函数声明主体开头的注释块中加上控制符: @param

例:

```
/**  
 * @param buf1 缓冲区1  
 * @param buf2 缓冲区2  
 * @param count 字符的个数  
 */  
int memcmp(const void *buf1, const void *buf2, size_t count);
```

### 3.2.5 函数的返回值

[格式]

在函数声明主体开头的注释块中加上控制符：@return

例1:

```
/**
 * @return 读取的字符
 */

int getchar(void);
```

例2:

```
/**
 * @return None
 */

void rewind(FILE *stream);
```

### 3.2.6 函数的注意事项（Remark）

[格式]

在函数声明主体开头的注释块中加上控制符：@remark

例:

```
/**
 * @remark 拷贝strSource 到strDestination（包含strSource 中的'\0'），
 * 要注意在拷贝的过程中并没有溢出检查
 */

char *strcpy(char *strDestination, const char *strSource);
```

### 3.2.7 补充说明

更多的格式请参考 Doxygen 文档，因为有些高级的选项比较繁琐。编写注释的目的是为了方便自动生成文档，而在此基础上又不能使程序员过于分散精力。Doxygen 的文档自动生成功能非常强大，很多事情都不需要手工干预，我们只要用到最基本的几项功能也可以满足需求了。

## 3.3 C/C++代码书写规范

### 3.3.1 每行代码长度

每行代码的长度推荐为 80 个 ASCII 字符，最长不得超过 120；折行以对齐为准。

例：

```
HANDLE DNC_OpenFile(const char cszFileName[],
                    int nMode);
```

或者：

```
BOOL DNC_ReadFile(
    HANDLE hFile,
    void *pvBuffer,
    int nReadSize,
    int *pnReadSize
);
```

### 3.3.2 合并行的问题

循环、分支代码，判断条件与执行代码不得在同一行上。

例：

正确：

```
if (n == -2)
    n = 1;
else
    n = 2;
```

不得写做：

```
If (n == -2) n = 1;
else n = 2;
```

### 3.3.3 指针中\*号的位置

指针的定义，\* 号既可以紧接类型，也可以在变量名之前。

例：

可写做： `int* pnsiz;` //统一使用这种格式

也可写做： `int *pnsiz;`

但不得写做： `int * pnsiz;`



### 3.3.4 全局函数的调用

在类的成员函数内调用全局函数时，在全局函数名前必须加上“::”。

### 3.3.5 关于 if...else if

else if 必须写在一行。

### 3.3.6 与“{”、“}”有关的规定

#### 3.3.6.1 与“{”、“}”有关代码的缩进格式

在“{”之前不允许有除空格与 Tab 之外的其他任何字符；在它之后可有注释，但不允许有代码。与“{”对应的“}”必须在同一列上。

例：

正确：

```
for (i = 0; i < cbLine; i++)
{ // .....
    printf("Line %d:", i);
    printf("%s\n", pFileLines[i]);
}
```

不得写做：

```
for (i = 0; i < cb; i++)
{ printf("Line %d:", i);
  printf("%s\n", pFileLines[i]);
}
```

也不得写做：

```
for (i = 0; i < cb; i++){
    printf("Line %d:", i);
    printf("%s\n", pFileLines[i]);
}
```

#### 3.3.6.2 “{”、“}”的省略

在循环、分支之后若只有一行代码，在无歧义的情况下可省略“{”、“}”。

例：

```
if (n == -2)
    n = 1;
else
    n = 2;
```

### 3.3.7 与空格有关的各项规定

#### 3.3.7.1 对两目、三目运算符两边的空格的规定

在所有两目、三目运算符的两边都必须有空格。在单目运算符两端不必空格。但在“>”、“::”、“.”、“[”、“]”等运算符前后，及“&”（取地址）、“\*”（取值）等运算符之后不得有空格。

例：

正确：

```
int n = 0, nTemp;
for (int i = nMinLine; i <= nMaxLine; i++)
```

不得写做：

```
int n=0, nTemp;
for ( int i=nMinLine; i<=nMaxLine; i++ )
```

#### 3.3.7.2 对 for、while、if 等关键字之后空格的规定

for、while、if 等关键字之后应有 1 个空格，再接“(”。

例：

正确：

```
if (-2 == n)
```

不得写做：

```
if(-2 == n)
```

#### 3.3.7.3 对调用函数、宏时的空格规定

调用函数、宏时，“(”前不得有空格。

例：

正确：

```
printf("%d\n", nIndex);
```

不得写做：

```
printf ("%d\n", nIndex);
```

#### 3.3.7.4 对类型强制转换时的空格规定

类型强制转换时，“(”“)”前后不得有空格

例：

可写做：

```
(DNC_File*)pFile;
```

也可写做：

```
(DNC_File *)pFile
```

不得写做：

```
( DNC_File* )pFile
```

```
( DNC_File * ) pFile
```

#### 3.3.8 代码缩进/对齐使用空格的规定

为避免用不同的编辑器阅读程序时，因 TAB 键所设置的空格数目不同而造成程序布局不整齐，不要使用 BC 作为编辑器合版本，因为 BC 会自动将 8 个空格变为一个 TAB 键，因此使用 BC 合入的版本大多会将缩进变乱。

##### 3.3.8.1 代码缩进一个 4 个空格

###### (1) 函数体相对函数名及“{”、“}”

例：

```
int power(int x)
{
    return (x * x);
}
```

###### (2) if、else、for、while、do 等之后的代码

一行之内写不下，折行之后的代码，应在合理的位置进行折行。若有+ - \* / 等运算符，则运算符应在上一行末尾，而不应在下一行的行首。

##### 3.3.8.2 代码不缩进情况

switch 之后的 case、default 代码不缩进。

例：

```
switch (nID)
```

```
{  
    case ID_PLAY:  
        .....  
        break;  
    case ID_STOP:  
        .....  
        break;  
    default:  
        .....  
        break;  
}
```

### 3.3.9 与类相关的.h 文件与.cpp 文件

#### 3.3.9.1 类的定义与实现

原则上，应该在一个单独的.h 文件中定义一个类，在一个单独的.cpp 文件中实现这个类。 .h 与.cpp 文件的文件名必须与类名相同。

#### 3.3.9.2 类的合并

若几个类的规模都不大，关系又很密切，则可在一个.h 文件中定义这些类，在一个.cpp 文件中实现。对于附属于较大规模类的一个很小规模的类， 可以写在那个大规模类的.h 和.cpp 里。

### 3.3.10 对变量的要求

#### 3.3.10.1 对变量的要求

- 公共变量是增大模块间耦合的原因之一，故应减少没必要的公共变量以降低模块间的耦合度；
- 在对变量声明的同时，应对其含义、作用及取值范围进行注释说明，同时若有必要还应说明与其它变量的关系；
- 对公共变量赋值时，若有必要应进行合法性检查，以提高代码的可靠性、稳定性；
- 防止局部变量与公共变量同名；

- 构造仅有一个模块或函数可以修改、创建，而其余有关模块或函数只访问的公共变量，防止多个不同模块或函数都可以修改、创建同一公共变量的现象；
- 严禁使用未经初始化的变量作为右值；
- 使用严格形式定义的、可移植的数据类型，尽量不要使用与具体硬件或软件环境关系密切的变量；
- 明确过程操作变量的关系后，将有利于程序的进一步优化、单元测试、系统联调以及代码维护等。这种关系的说明可在注释或文档中描述。

示例：在源文件中，可按如下注释形式说明。

RELATION System\_Init Input\_Rec Print\_Rec Stat\_Score

Student Create Modify Access Access

Score Create Modify Access Access, Modify

注：RELATION 为操作关系；System\_Init、Input\_Rec、Print\_Rec、Stat\_Score 为四个不同的函数；Student、Score 为两个全局变量；Create 表示创建，Modify 表示修改，Access 表示访问。其中，函数 Input\_Rec、Stat\_Score 都可修改变量 Score，故此变量将引起函数间较大的耦合，并可能增加代码测试、维护的难度。

### 3.3.10.2 对变量类型的要求

- 留心具体语言及编译器处理不同数据类型的原则及有关细节，如在 C 语言中，static 局部变量将在内存“数据区”中生成，而非 static 局部变量将在“堆栈”中生成。这些细节对程序质量的保证非常重要；
- 要注意数据类型的强制转换，当进行数据类型强制转换时，其数据的意义、转换后的取值等都有可能发生变化，而这些细节若考虑不周，就很有可能留下隐患；
- 尽量减少没有必要的数据类型默认转换与强制转换；
- 合理地设计数据并使用自定义数据类型，避免数据间进行不必要的类型转换；
- 对编译系统默认的数据类型转换，也要有充分的认识。

### 3.3.11 对结构的要求

#### 3.3.11.1 结构的功能要单一

设计结构时应力争使结构代表一种现实事务的抽象，而不是同时代表多种。结构中的各元素应代表同一事务的不同侧面，而不应把描述没有关系或关系很弱的不同事

务的元素放到同一结构中。

示例：如下结构不太清晰、合理。

```
typedef struct STUDENT_STRU
{
    unsigned char name[8]; /* student's name */
    unsigned char age; /* student's age */
    unsigned char sex; /* student's sex, as follows */
                        /* 0 - FEMALE; 1 - MALE */
    unsigned char
    teacher_name[8]; /* the student teacher's name */
    unsigned char
    teacher_sex; /* his teacher sex */
} STUDENT;
```

若改为如下，可能更合理些。

```
typedef struct TEACHER_STRU
{
    unsigned char name[8]; /* teacher name */
    unsigned char sex; /* teacher sex, as follows */
                        /* 0 - FEMALE; 1 - MALE */
} TEACHER;
```

```
typedef struct STUDENT_STRU
{
    unsigned char name[8]; /* student's name */
    unsigned char age; /* student's age */
    unsigned char sex; /* student's sex, as follows */
                        /* 0 - FEMALE; 1 - MALE */
    unsigned int teacher_ind; /* his teacher index */
} STUDENT;
```

### 3.3.11.2 不同结构间的关系要简单

若两个结构间关系较复杂、密切，那么应合为一个结构。

示例：如下两个结构的构造不合理。

```
typedef struct PERSON_ONE_STRU
```

```
{  
    unsigned char name[8];  
    unsigned char addr[40];  
    unsigned char sex;  
    unsigned char city[15];
```

```
} PERSON_ONE;
```

```
typedef struct PERSON_TWO_STRU
```

```
{  
    unsigned char name[8];  
    unsigned char age;  
    unsigned char tel;
```

```
} PERSON_TWO;
```

由于两个结构都是描述同一事物的，那么不如合成一个结构。

```
typedef struct PERSON_STRU
```

```
{  
    unsigned char name[8];  
    unsigned char age;  
    unsigned char sex;  
    unsigned char addr[40];  
    unsigned char city[15];  
    unsigned char tel;
```

```
} PERSON;
```

### 3.3.12 函数的编写要求

#### 3.3.12.1 函数编写的总体要求

- 明确函数功能，精确（而不是近似）地实现函数设计；
- 函数的规模尽量限制在 **200** 行以内(不包括注释和空格行)；
- 函数的返回值要清楚、明了，让使用者不容易忽视错误情况；



- 除非必要，最好不要把与函数返回值类型不同的变量，以编译系统默认的方式或强制的转换方式作为返回值返回；
- 让函数在调用点显得易懂、容易理解，在调用函数填写参数时，应尽量减少没有必要的默认数据类型转换或强制数据类型转换。

### 3.3.12.2 函数名的要求

- 函数名应准确描述函数的功能。
- 避免使用无意义或含义不清的动词为函数命名，如 `process`、`handle` 等为函数命名，因为这些动词并没有说明要具体做什么。
- 使用动宾词组为执行某操作的函数命名。如果是 **OOP** 方法，可以只有动词（名词是对象本身）。

示例：参照如下方式命名函数。

```
void DNC_PrintRecord( unsigned int rec_ind );
```

```
int DNC_InputRecord( void );
```

```
unsigned char DNC_GetCurrentColor( void );
```

### 3.3.12.3 函数参数的要求

- 在同一项目组应明确规定对接接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责，缺省是由函数调用者负责。对于模块间接口函数的参数的合法性检查这一问题，往往有两个极端现象，即：要么是调用者和被调用者对参数均不作合法性检查，结果就遗漏了合法性检查这一必要的处理过程，造成问题隐患；要么就是调用者和被调用者均对参数进行合法性检查，这种情况虽不会造成问题，但产生了冗余代码，降低了效率。
- 防止将函数的参数作为工作变量。将函数的参数作为工作变量，有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。
- 避免设计多参数函数，不使用的参数从接口中去掉。

示例：下函数的实现不太好。

```
void DNC_SumData( unsigned int num, int *data, int *sum )
```

```
{
```

```
    unsigned int count;
```

```
    *sum = 0;
```



```
for (count = 0; count < num; count++)
{
    *sum += data[count]; // sum 成了工作变量，不太好。
}
}
```

若改为如下，则更好些。

```
void DNC_SumData( unsigned int num, int *data, int *sum )
{
    unsigned int count ;
    int sum_temp;
    sum_temp = 0;
    for (count = 0; count < num; count ++ )
    {
        sum_temp += data[count];
    }
    *sum = sum_temp;
}
```

#### 3.3.12.4 函数功能的要求

- 一个函数仅完成一件功能。
- 为简单功能编写函数。 虽然为仅用一两行就可完成的功能去编函数好像没有必要，但用函数可使功能明确化，增加程序可读性，亦可方便维护、测试。

示例：如下语句的功能不很明显。

```
value = ( a > b ) ? a : b ;
```

改为如下就很清晰了。

```
int max (int a, int b)
{
    return ((a > b) ? a : b);
}

value = max (a, b);
```

- 不要设计多用途面面俱到的函数。多功能集于一身的函数，很可能使函数的理解、测试、维护等变得困难。
- 函数的功能应该是可以预测的，也就是只要输入数据相同就应产生同样的输出。

### 3.3.12.5 可重入函数编写

- 编写可重入函数时，应注意局部变量的使用（如编写 C/C++语言的可重入函数时，应使用 **auto** 即缺省态局部变量或寄存器变量）；编写 C/C++语言的可重入函数时，不应使用 **static** 局部变量，否则必须经过特殊处理，才能使函数具有可重入性。
- 编写可重入函数时，若使用全局变量，则应通过关中断、信号量（即 **P**、**V** 操作）等手段对其加以保护；若对所使用的全局变量不加以保护，则此函数就不具有可重入性，即当多个进程调用此函数时，很有可能使有关全局变量变为不可知状态。

示例：假设 Exam 是 int 型全局变量，函数 Squire\_Exam 返回 Exam 平方值。那么如下函数不具有可重入性。

```
unsigned int example( int para )
{
    unsigned int temp;
    Exam = para; //  (**)
    temp = Square_Exam( );
    return temp;
}
```

此函数若被多个进程调用的话，其结果可能是未知的，因为当 (\*\*) 语句刚执行完后，另外一个使用本函数的进程可能正好被激活，那么当新激活的进程执行到此函数时，将使 Exam 赋与另一个不同的 para 值，所以当控制重新回到“temp = Square\_Exam( )”后，计算出的 temp 很可能不是预想中的结果。此函数应如下改进。

```
unsigned int example( int para )
{
    unsigned int temp;
```

```
[申请信号量操作] // 若申请不到“信号量”，说明另外的进程正处于  
Exam = para; // 给 Exam 赋值并计算其平方过程中（即正在使用此  
temp = Square_Exam(); // 信号），本进程必须等待其释放信号后，才可继  
[释放信号量操作] // 续执行。若申请到信号，则可继续执行，但其  
// 它进程必须等待本进程释放信号量后，才能再使用本信  
号。
```

```
    return temp;  
}
```

### 3.3.13 C/C++程序效率的要求

- 编程时要经常注意代码的效率。代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。
- 在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。
- 局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。
- 通过对系统数据结构的划分与组织的改进，以及对程序算法的优化来提高空间效率。这种方式是解决软件空间效率的根本办法。
- 仔细分析有关算法，并进行优化。
- 仔细考查、分析系统及模块处理输入（如事务、消息等）的方式，并加以改进。
- 对模块中函数的划分及组织方式进行分析、优化，改进模块中函数的组织结构，提高程序效率。
- 避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中。
- 尽量用乘法或其它方法代替除法，特别是浮点运算中的除法。
- 不要一味追求紧凑的代码。说明：因为紧凑的代码并不代表高效的机器码。

### 3.3.14 C/C++代码质量的要求

#### 3.3.14.1 代码质量保证优先原则

- 正确性，指程序要实现设计要求的功能。
- 稳定性、安全性，指程序稳定、可靠、安全。
- 可测试性，指程序要具有良好的可测试性。
- 规范/可读性，指程序书写风格、命名规则等要符合规范。
- 全局效率，指软件系统的整体效率。
- 局部效率，指某个模块/子模块/函数的本身效率。
- 个人表达方式/个人方便性，指个人编程习惯。

#### 3.3.14.2 对内存及句柄操作的要求

- 只引用属于自己的存贮空间。若模块封装的较好，那么一般不会发生非法引用他人的空间。
- 防止引用已经释放的内存空间。在实际编程过程中，稍不留心就会出现在一个模块中释放了某个内存块（如C语言指针），而另一模块在随后的某个时刻又使用了它。要防止这种情况发生。
- 过程/函数中分配的内存，在过程/函数退出之前要释放。
- 过程/函数中申请的（为打开文件而使用的）文件句柄，在过程/函数退出之前要关闭。分配的内存不释放以及文件句柄不关闭，是较常见的错误，而且稍不注意就有可能发生。这类错误往往会引起很严重后果，且难以定位。

示例：下函数在退出之前，没有把分配的内存释放。

```
typedef unsigned char BYTE;

int DNC_ExampleFun( BYTE gt_len, BYTE *gt_code )
{
    BYTE *gt_buf;

    gt_buf = (BYTE *) malloc (MAX_GT_LENGTH);

    //program code, include check gt_buf if or not NULL.

    /* global title length error */

    if (gt_len > MAX_GT_LENGTH)
    {
```

```
        return GT_LENGTH_ERROR; // 忘了释放 gt_buf
    }
    // other program code
}
```

应改为如下。

```
int DNC_ExampleFun( BYTE gt_len, BYTE *gt_code )
{
    BYTE *gt_buf;
    gt_buf = (BYTE *) malloc ( MAX_GT_LENGTH );
    // program code, include check gt_buf if or not NULL.
    /* global title length error */
    if (gt_len > MAX_GT_LENGTH)
    {
        free( gt_buf ); // 退出之前释放 gt_buf
        return GT_LENGTH_ERROR;
    }
    // other program code
}
```

- 防止内存操作越界。内存操作主要是指对数组、指针、内存地址等的操作。内存操作越界是软件系统主要错误之一，后果往往非常严重，所以当我们进行这些操作时一定要仔细小心。

示例：假设某软件系统最多可由 10 个用户同时使用，用户号为 1-10，那么如下程序存在问题。

```
#define MAX_USR_NUM 10
unsigned char usr_login_flg[MAX_USR_NUM]= "";
void DNC_SetUsrLoginFlg( unsigned char usr_no )
{
    if (!usr_login_flg[usr_no])
    {
        usr_login_flg[usr_no]= TRUE;
    }
}
```

```
    }  
}  
当 usr_no 为 10 时, 将使用 usr_login_flg 越界。可采用如下方式解决。  
void DNC_SetUsrLoginFlg( unsigned char usr_no )  
{  
    if (!usr_login_flg[usr_no - 1])  
    {  
        usr_login_flg[usr_no - 1]= TRUE;  
    }  
}
```

### 3.3.14.3 其他要求

- 系统运行之初, 要初始化有关变量及运行环境, 防止未经初始化的变量被引用。
- 系统运行之初, 要对加载到系统中的数据进行一致性检查。使用不一致的数据, 容易使系统进入混乱状态和不可知状态。
- 严禁随意更改其它模块或系统的有关设置和配置。 编程时, 不能随心所欲地更改不属于自己模块的有关设置如常量、数组的大小等。
- 不能随意改变与其它模块的接口。
- 充分了解系统的接口之后, 再使用系统提供的功能。
- 编程时, 要防止差 1 错误。 此类错误一般是由于把“<=”误写成“<”或“>=”误写成“>”等造成的, 由此引起的后果, 很多情况下是很严重的, 所以编程时, 一定要在这些地方小心。当编完程序后, 应对这些操作符进行彻底检查。
- 要时刻注意易混淆的操作符。当编完程序后, 应从头至尾检查一遍这些操作符, 以防止拼写错误。形式相近的操作符最容易引起误用, 如 C/C++中的“=”与“==”、“|”与“||”、“&”与“&&”等, 若拼写错了, 编译器不一定能够检查出来。
- 有可能的话, **if** 语句尽量加上 **else** 分支, 对没有 **else** 分支的语句要小心对待; **switch** 语句必须有 **default** 分支。
- 不要滥用 **goto** 语句。 **goto** 语句会破坏程序的结构性, 所以除非确实需要, 最好不使用 **goto** 语句。
- 不使用与硬件或操作系统关系很大的语句, 而使用建议的标准语句, 以提高软



件的可移植性和可重用性。

- 除非为了满足特殊需求，避免使用嵌入式汇编。程序中嵌入式汇编，一般都对可移植性有较大的影响。
- 精心地构造、划分子模块，并按“接口”部分及“内核”部分合理地组织子模块，以提高“内核”部分的可移植性和可重用性。对不同产品中的某个功能相同的模块，若能做到其内核部分完全或基本一致，那么无论对产品的测试、维护，还是对以后产品的升级都会有很大帮助。
- 精心构造算法，并对其性能、效率进行测试。
- 对较关键的算法最好使用其它算法来确认。
- 时刻注意表达式是否会上溢、下溢。
- 使用变量时要注意其边界值的情况。
- 留心程序机器码大小（如指令空间大小、数据空间大小、堆栈空间大小等）是否超出系统有关限制。
- 为用户提供良好的接口界面，使用户能较充分地了解系统内部运行状态及有关系统出错情况。
- 系统应具有一定的容错能力，对一些错误事件（如用户误操作等）能进行自动补救。
- 对一些具有危险性的操作代码（如写硬盘、删数据等）要仔细考虑，防止对数据、硬件等的安全构成危害，以提高系统的安全性。
- 使用第三方提供的软件开发工具包或控件时，要注意以下几点：（1）充分了解应用接口、使用环境及使用时注意事项；（2）不能过分相信其正确性；（3）除非必要，不要使用不熟悉的第三方工具包与控件。使用工具包与控件，可加快程序开发速度，节省时间，但使用之前一定对它有较充分的了解，同时第三方工具包与控件也有可能存在问题。
- 资源文件（多语言版本支持），如果资源是对语言敏感的，应让该资源与源代码文件脱离，具体方法有下面几种：使用单独的资源文件、**DLL** 文件或其它单独的描述文件（如数据库格式）

### 3.3.15 C/C++代码测试与维护

- 单元测试要求至少达到语句覆盖。

- 单元测试开始要跟踪每一条语句，并观察数据流及变量的变化。
- 清理、整理或优化后的代码要经过审查及测试。
- 代码版本升级要经过严格测试。
- 使用工具软件对代码版本进行维护。
- 正式版本上软件的任何修改都应有详细的文档记录。
- 发现错误立即修改，并且要记录下来。
- 关键的代码在汇编级跟踪。
- 仔细设计并分析测试用例，使测试用例覆盖尽可能多的情况，以提高测试用例的效率。
- 尽可能模拟出程序的各种出错情况，对出错处理代码进行充分的测试。
- 仔细测试代码处理数据、变量的边界情况。
- 保留测试信息，以便分析、总结经验及进行更充分的测试。
- 不应通过“试”来解决问题，应寻找问题的根本原因。
- 对自动消失的错误进行分析，搞清楚错误是如何消失的。
- 修改错误不仅要治表，更要治本。
- 测试时应设法使很少发生的事件经常发生。
- 明确模块或函数处理哪些事件，并使它们经常发生。
- 坚持在编码阶段就对代码进行彻底的单元测试，不要等以后的测试工作来发现问题。
- 去除代码运行的随机性（如去掉无用的数据、代码及尽可能防止并注意函数中的“内部寄存器”等），让函数运行的结果可预测，并使出现的错误可再现。

## 4 C/C++源代码程序命名规范

本规范给出了工程、文件、函数、变量、类/结构、宏、常量、枚举、联合等的命名规范。

### 4.1 总体命名要求

- 所有命名都应使用标准的英文单词或缩写，最好不使用拼音或拼音缩写，除非该名字描述的是中文特有的内容，如半角、全角，声母、韵母等。
- 所有命名都应遵循达意原则，即名称应含义清晰、明确。



- 所有命名都不易过长，应控制在规定的最大长度以内。
- 所有命名都应尽量使用全称。
- 如果命名使用缩写，则应该使用《通用缩写表》(见最后一节)中的缩写；原则上不推荐使用《通用缩写表》以外的缩写，如果使用，则必须对其进行注释和说明。

## 4.2 具体命名规范

### 4.2.1 工程名

工程名不强制统一。

### 4.2.2 文件名

- 只能由英文字母、数字和下划线组成文件名。
- 区分大小写,基于跨平台的考虑,建议全小写。
- 长度建议不多于 30 个字符。
- 如果文件用于定义或实现类，则文件名应与类名必须保持一致。

### 4.2.3 函数名

- 参照 Windows API 的命名规范。
- 函数名最长不得超过 64 个字符。
- 推荐使用动宾结构。函数名应清晰反映函数的功能、用途。
- 推荐函数名第一个字母大写。

### 4.2.4 变量名

原则上，变量的命名遵从匈牙利记法。即：前缀+ 类型+ 名称。最终的变量名总长不得超过 32 个英文字符。

格式：

[m\_|s\_|g\_] type [class name|struct name] variable name

解释：

- m\_ ： 类的成员变量
- ms\_： 类的静态成员变量
- s\_ ： 静态全局变量

·g\_ : 普通全局变量

·类型缩写 (type)

·char, TCHAR: c/ch

·字符串: s/sz/str

·bool, BOOL: b

·int, \_\_int16, \_\_int32, \_\_int64: n

·unsigned: u

·long: l

·unsigned long: ul

·double, float: f

·BYTE: by

·WORD: w

·DWORD: dw

·function: fn

·pointer: p

小范围局部变量与结构成员可省略类型缩写。

#### 4.2.5 类名

·必须以大写"C"开头, 后面字母反映具体含义, 以清晰表达类的用途和功能为原则。

·接口必须以大写"I"开头, 代表 Interface 。

·当名称由多个单词构成时, 每一个单词的第一个字母必须大写。

#### 4.2.6 结构、宏、枚举及联合的名字

结构、宏、枚举以及联合的名字必须全部大写。

#### 4.2.7 具有互斥意义的变量或相反动作的函数的命名

用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。说明: 下面是一些在软件中常用的反义词组。

add/remove	begin/end	create/destroy
insert/delete	first/last	get/release

increment/decrement	put/get	add/delete
lock/unlock open/close	min/max	old/new
start/stop	next/previous	source/target
show/hide	send/receive	cut/paste
source/destination	up/down	

示例:

```
int min_sum;

int max_sum;

int add_user( BYTE *user_name );

int delete_user( BYTE *user_name );
```

#### 4.2.8 对命名中下划线的使用规定

除了编译开关/头文件等特殊应用,应避免使用\_**EXAMPLE\_TEST**\_之类以下划线开始和结尾的定义。

### 5 程序开发中相关文档撰写规定

程序开发中相关文档的撰写是对程序设计、使用、测试和问题记录的一个总结,是他人熟悉、掌握和运用(使用、升级和修改)程序的基础,也是一个程序软件的生命力所在之处。

#### 5.1 程序设计与开发文档

程序设计与开发文档是整个软件系统的总体构成和开发的内容,应体现系统的完整性和易读性。

##### 5.1.1 软件的开发背景

简要介绍软件程序的开发背景及其要实现的功能,并给出文档中出现的符号的详细说明。

##### 5.1.2 软件的结构设计

软件的结构设计主要体现整个系统软件的总体架构,按功能模块进行划分,给出每个模块的功能及模块间交换的数据项,并附图加以说明。在此基础之上给出整个软件的流程图。

### 5.1.3 数据及资源文件说明

#### 5.1.3.1 数据用途

本节中给出所编写运行时需要的数据文件和资源文件的格式及来源。数据文件的后缀格式（如\*.dat，\*.txt 等等）和数据文件的用途，可通过列表给出。

#### 5.1.3.2 数据格式

本节中需要给出软件中所用到的数据的存储格式，数据的大小，并给出每一个（列或行）元素所代表的内容及量纲。

### 5.1.4 程序结构设计与开发

本节按照软件需求和软件的结构设计进行开发，给出每个模块对应的类或结构的功能，并对类中重要的函数和变量进行详细说明，并给出每个模块的具体流程图。

## 5.2 程序使用文档

程序使用文档是保证软件程序在不同的机器上运行或者是非开发人员运行软件程序的基础，是软件程序的使用手册，应尽量详细。

### 5.2.1 软件系统的程序组成

详细给出所开发软件的程序组成及功能，可列表给出。要求将程序文件夹中的每一个文件都要列出，并给出相应的功能。在此基础上给出各个程序间的调用关系和配置关系，即如何操作才能使软件程序正常运行。

### 5.2.2 开发环境要求、软件安装及环境配置

根据软件的任务需求及目的，确定软件系统所采用的开发环境及应用软件，在此基础上配置系统工作的整体环境，要求图文并茂，注重细节。以在 Windows XP 环境下，采用 VC++6.0 和 OpenCV 应用软件进行程序开发为例，给出开发环境要求、软件安装和环境配置。

对于软件开发中使用的开发包的情况，要说明使用的版本号、获取方法和安装方式等。另外，还要说明该开发包有没有替代品。

#### （1）软件运行最低配置要求

运行环境：中文 Windows NT 操作系统

CPU：2.0GHz 以上

内存要求：1G 内存以上

显示卡最小设置：16 色

显示器：17 寸以上彩显

数据接口：键盘鼠标输入，文件输入输出，显示器输出

## (2) VC++6.0 应用软件的安装及配置

VC++6.0 应用软件的安装及配置过程（略）。

## (3) OpenCV 的安装及配置

### • Stept1: 从 <http://www.opencv.org.cn/> 下载 OpenCV 安装程序

假如要将 OpenCV 安装到 C:\Program Files\OpenCV。（下面附图为 OpenCV 1.0rc1 的安装界面，OpenCV 1.0 安装界面与此基本一致。）在安装时选择"将\OpenCV\bin 加入系统变量"（Add\OpenCV\bin to the system PATH），如图 1 所示。

### • Stept2: 配置 Windows 环境变量

检查 C:\Program Files\OpenCV\bin 是否已经被加入到环境变量 PATH，如果没有，请加入，如图 2 所示。加入后需要注销当前 Windows 用户（或重启）后重新登陆才生效。（可以在任务管理器里重启 explorer.exe）

### • Stept3: 对 Visual C++ 6.0 环境的全局配置

在 Visual C++ 6.0 菜单 Tools->Options->Directories: 先设置 lib 路径，选择 Library files，在下方填入路径，如图 3（a）所示。

C:\Program Files\OpenCV\lib

然后选择 include files，在下方填入路径：

C:\Program Files\OpenCV\cxcore\include

C:\Program Files\OpenCV\cv\include

C:\Program Files\OpenCV\cvaux\include

C:\Program Files\OpenCV\ml\include

C:\Program Files\OpenCV\otherlibs\highgui

C:\Program Files\OpenCV\otherlibs\cvcam\include

### • Stept4: 对 Visual C++ 6.0 环境的项目配置

每创建一个将要使用 OpenCV 的 VC Project，都需要给它指定需要的 lib，同时要给出如何获取 lib 库。菜单: Project->Settings 然后将 Setting for 选为 All Configurations

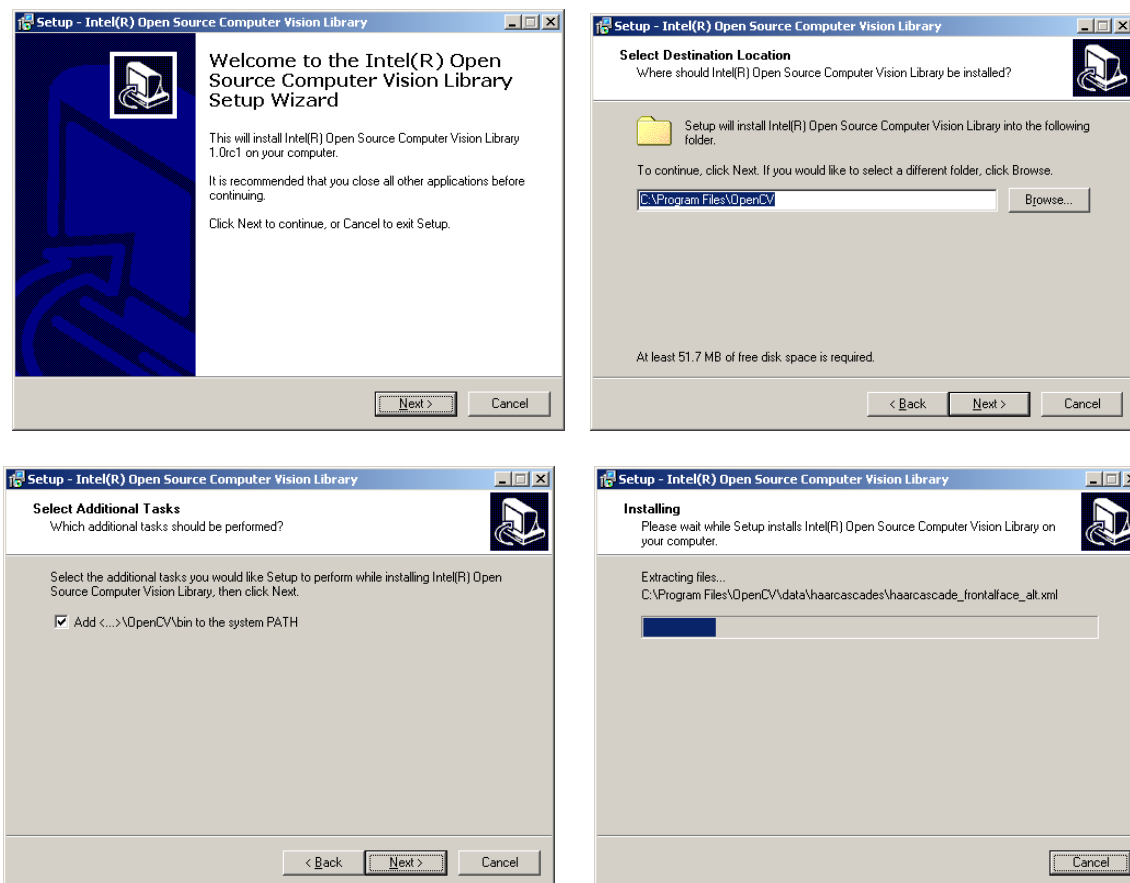


图 1 OpenCV 的安装界面及主要步骤

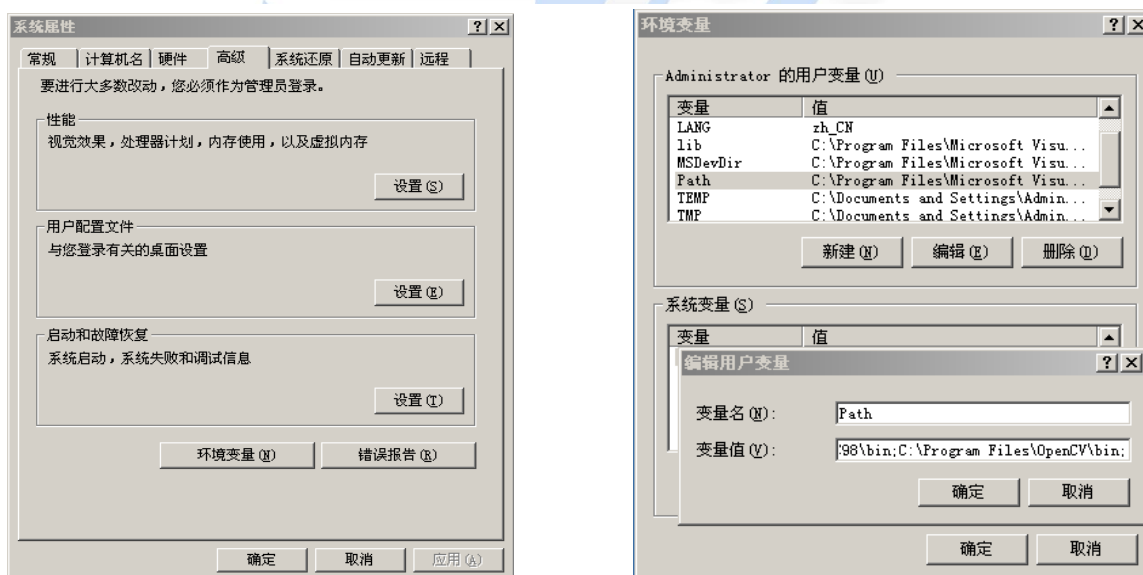


图 2 OpenCV 环境变量的设置

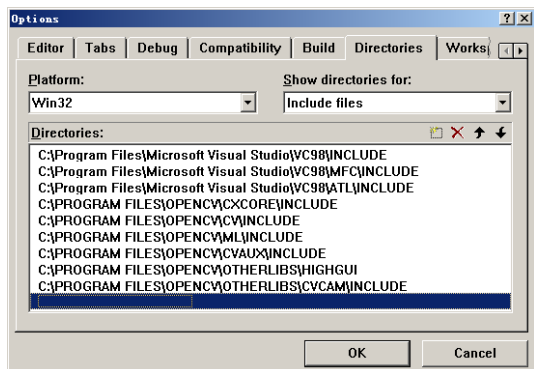
然后选择右边的 link 标签，在 Object/library modules 附加上

cxcore.lib cv.lib ml.lib cvaux.lib highgui.lib cvcam.lib

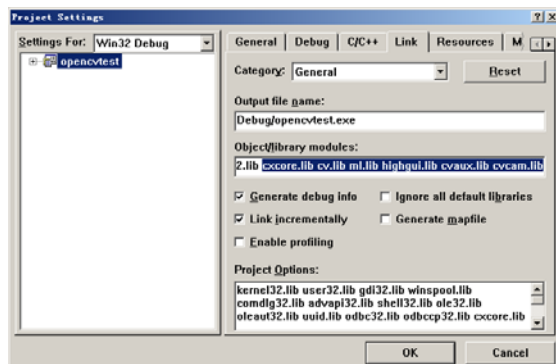
如果你不需要这么多 lib，你可以只添加你需要的 lib，如图 3（b）所示。

## • Step5: 安装 DirectX SDK 及其配置

安装 DirectX SDK 及其配置过程（略）。



(a)



(b)

图 3 VC++6.0 全局和项目环境设置

## 5.2.3 软件系统的使用方法

根据所开发的软件系统的功能给出具体操作的过程，所有的功能都需要给出操作的过程，从数据输入到结果输出。以实验室开发的四组合导航仿真软件的数据加载及其显示为例给出软件具体操作方法。

软件运行结果如图 4 所示。

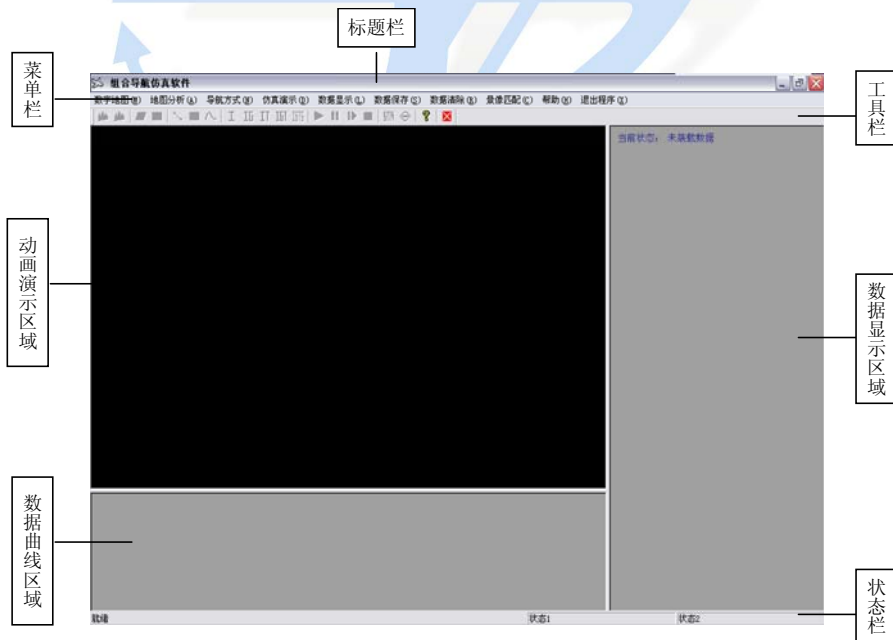


图 4 组合导航仿真软件实验平台主界面

在菜单中选择“数字地图(M)”→“地图生成(G)”→“真实地图(R)...”，如图 5 所示；

或者点击工具栏的“”。弹出加载真实地图对话框，选择地图文件和地形纹理文件，



如图 6 所示。加载了真实地图后，数据显示区域会显示当前状态及加载的地形数据的参数，如图 7 所示。



图 5 加载真实数字地图菜单



图 6 加载真实数字地图对话框

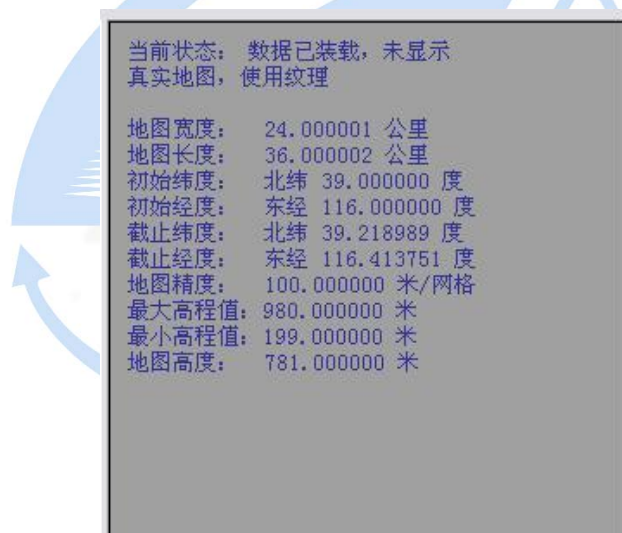


图 7 加载真实数字地图后的数据显示区域


加载数字地图后，在菜单中选择“数字地图(M)”→“地图显示(S)”→“三维地形图(T)...”，如图 8 所示；或者在工具栏上点击“”。弹出地形三维显示对话框，选择显示模式和显示参数，如图 9 所示。显示方式包括“鸟瞰”和“漫游”两种，分别对地图进行全貌显示和三维漫游显示；可以设定显示的步长，以设定动画演示的精细程度；还可以设置网格模式，查看地形网格的显示。确定后，动画演示区域会对加载地图进行三维显示，如图 10 所示。数据显示区域会显示当前演示状态，状态栏会显示当前动画演示帧率。





图 8 数字地图三维显示菜单



图 9 数字地图三维显示对话框

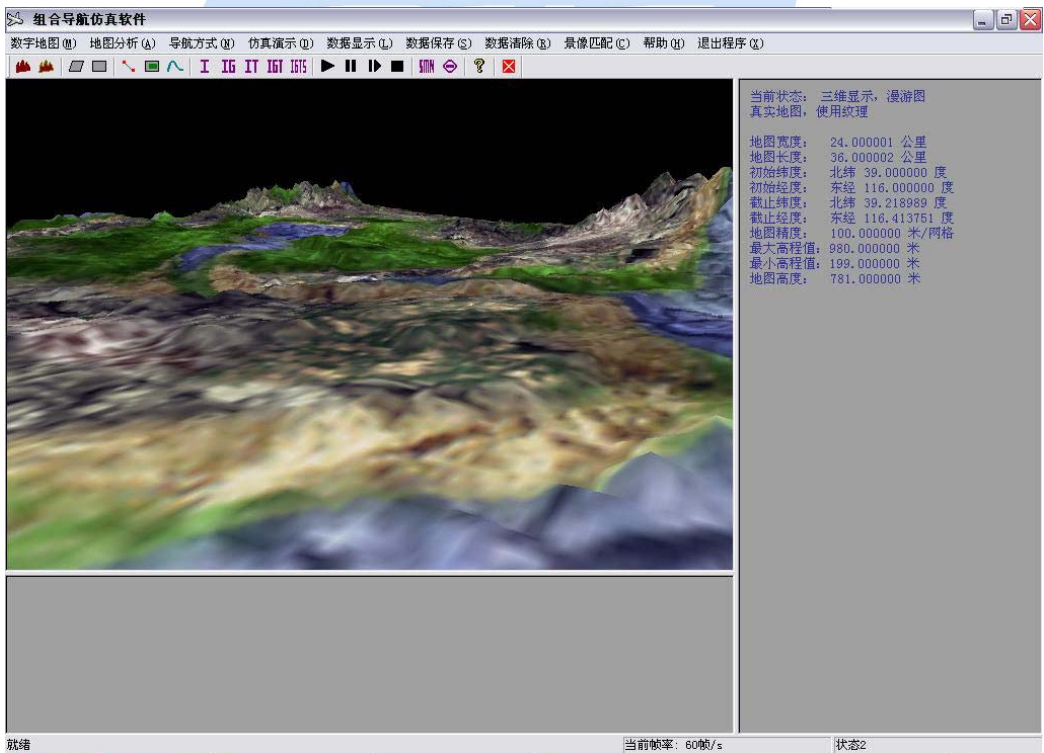


图 10 数字地图三维显示界面

5.3 程序测试文档

软件程序开发完后需要对软件进行测试，测试软件的各项功能，为软件的修开提

供参考意见。需要包括以下几部分内容。

### 5.3.1 测试环境

给出软件测试程序的测试环境，形式可参照 5.1 节中软件系统的开发环境。

### 5.3.2 软件测试项目和功能

给出软件需要测试哪些项目和功能。

### 5.3.3 软件测试数据及格式

给出软件测试时用到的数据及其格式。

### 5.3.4 软件测试过程

给出软件测试的详细过程及每一步得到的结果，具体可参照 5.2 节的软件使用方法。

## 5.4 软件程序调试文档

软件程序开发与调试文档是在整个软件开发过程中记录遇到的问题及解决的方法和调试过程，便于他人学习时方便地解决类似问题。

具体形式可参照如下形式。

问题一：虚拟机下创造可连接外网的环境问题：

问题产生：apache 服务器对机器的网络设置要求和连接外网对机器的网络设置要求相同，故可通过使机器连接外网的方式设置 apache 服务器使其正常运行。

系统环境：虚拟机：宿主机系统：Windows Vista + 虚拟机系统：Red Hat Linux 9.0 + 虚拟机软件：Vmware Workstation 5.0

双系统：Windows xp sp3 + fedora 8

解决方式：在虚拟机下，由于有学校客户端限制，只能采用 NAT 方式与宿主机相连以连接外网。可通过以下步骤进行：

- 1.关闭宿主机防火墙，然后在宿主机中设置允许虚拟机虚拟的“Vmware Network Adapter Vmnet8”通过“本地连接”共享连接网络。
- 2.在虚拟机软件中设置为 NAT 连接方式，并设置 subnet: 192.168.0.0; DHCP start IP address: 192.168.0.128; DHCP end IP address: 192.168.0.254; NAT Gateway IP address: 192.168.0.2。

- 3.在虚拟机中通过“主菜单 - 系统设置 - 网络”设置主 DNS：192.168.0.2；第二 DNS：255.255.255.0；第三 DNS：192.168.0.1，并设置主机 IP 为宿主机系统的 IP 地址，具体过程如图 11 所示。
- 4.激活网卡，可连接外网的网络设置完成。
- 5.在虚拟机系统终端中输入“/etc/init.d/httpd start”可打开 apache 服务器，服务器地址为 192.168.0.2，登陆页面如图 12 所示。
- 6.在双系统下设置较为简单，只要安装一个适用的 linux 华为客户端并连网即可。

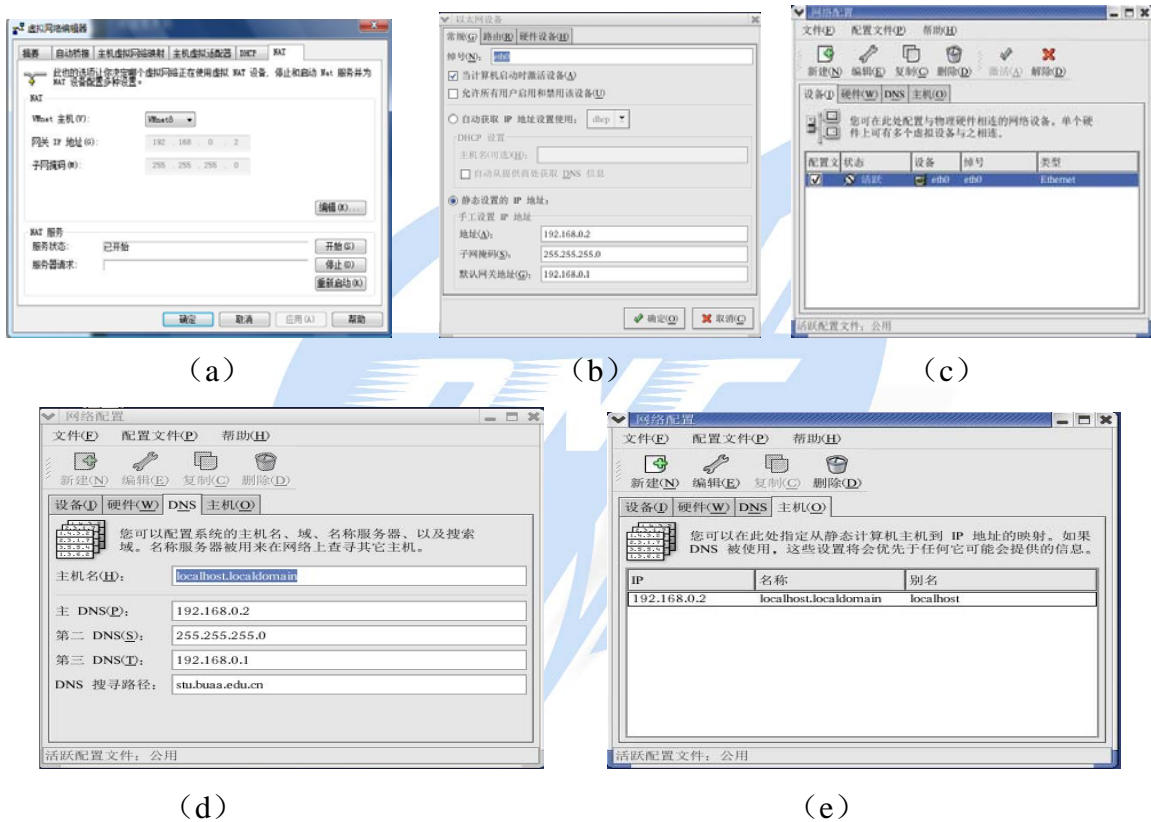


图 11 虚拟机网络环境配置过程



图 12 在虚拟机环境下登陆外网

解决人：××××

解决时间：×年×月×日

对于软件程序开发与调试文档可参照《北航数字导航中心\_项目(程序)调试报告》，即问题记录单。

## 6 程序及文档备份

### 6.1 源程序及其文档备份

软件程序及其文档备份是保证软件程序正常运行的一种手段，需要在项目主管、项目经理和数字导航中心三处同时备份。备份时要将源程序代码、设计与开发文档、使用文档、测试文档和调试文档以及测试数据一同备份，将程序和文档放到一个文件夹下，并以项目名称+年月日的形式命名，同时文件夹内配有一个 `readme.txt` 进行简要说明，`readme.txt` 的内容包括文件夹下包括的具体内容，备份时间，备份人等等。

### 6.2 第三方软件（开发包或源码）及其文档备份

在进行源程序及其文档备份的同时，还要备份源程序中所用到的第三方软件（开发包的安装文件或者源码），同时要给出第三方软件（开发包或源码）的使用说明。

## 7 词语缩写表

词语所写是软件开发和程序文档撰写过程中所必要的，本节给出通用词和专业词的所写。当出现新的词语所写时，直接填写在相应的表格空白（通用词和专业词）处，在本软件修订时统一补充。

### 7.1 词语缩写的总体要求

- 本缩写表中列出的都是通用性缩写或者是专业缩写，不提供标准缩写，如：`Win9x`、`COM` 等。
- 使用本缩写表里的缩写时，请对其进行必要的注释说明。
- 除少数情况以外，大部分缩写与大小写无关。
- 通用词和专业词缩写表中的词量要定期补充和更新。

### 7.2 通用词缩写表

通用缩写表如表 1 所示。

表 1 通用缩写

#	缩写	全称
1	addr	address
2	adm	Administrator
3	app	Application
4	arg	Argument
5	asm	assemble
6	asyn	asynchronization
7	avg	average
8	DB	Database
9	bk	Back
10	bmp	Bitmap
11	btn	Button
12	buf	Buffer
13	calc	Calculate
14	char	Character
15	chg	Change
16	clk	Click
17	clr	Color
18	cmd	Command
19	cmp	Compare
20	col	Column
21	coord	coordinates
22	cpy	Copy
23	ctl	Control
24	cur	Current
25	cyl	Cylinder
26	dbg	Debug
27	dbl	Double

28	dec	Decrease
29	def	Default
30	del	Delete
31	dst	Destination
32	dev	Device
33	dict	dictionary
34	diff	different
35	dir	directory
36	disp	Display
37	div	Divide
38	dlg	Dialog
39	doc	Document
40	drv	Driver
41	dyna	Dynamic
42	env	Environment
43	err	Error
44	ext	Extend
45	exec	Execute
46	flg	Flag
47	frm	Frame
48	func	Function
49	grp	Group
50	horz	Horizontal
51	idx	Index
52	img	Image
53	impl	Implement
54	inc	Increase
55	info	Information
56	init	Initial/Initialize/Initialization

57	ins	Insert
58	inst	Instance
59	intr	Interrupt
60	len	Length
61	lib	Library
62	lnk	Link
63	log	Logical
64	lst	List
65	max	Maximum
66	mem	Memory
67	mgr/man	Manage/Manager
68	mid	Middle
69	min	Minimum
70	msg	Message
71	mul	Multiply
72	num	Number
73	obj	Object
74	ofs	Offset
75	org	Origin/Original
76	param	Parameter
77	pic	Picture
78	pkg	Package
79	pkt	Packect
80	pnt/pt	Point
81	pos	Position
82	pre/prev	Previous
83	prg	Program
84	prn	Print
85	proc	Process/Procedure



86	prop	Properties
87	psw	Password
88	ptr	Pointer
89	pub	Public
90	rc	Rect
91	ref	Reference
92	reg	Register
93	req	Request
94	res	Resource
95	ret	Return
96	rgn	Region
97	scr	Screen
98	sec	Second
99	seg	Segment
100	sel	Select
101	src	Source
102	std	Standard
103	stg	Storage
104	stm	Stream
105	str	String
106	sub	Subtract
107	sum	summation
108	svr	Server
109	sync	Synchronization
110	sys	System
111	tbl	Table
112	temp/tmp	Temporary
113	tran/trans	translate/transation/transparent
114	tst	Test



4	imam	Image matching
5	smns	Scene matching navigation system
6	tm	terrain matching
7	tmns	terrain matching navigation system
8	tans	Terrain-aided navigation system
9	trns	Terrain-referenced navigation system
10	gps	Global positioning system
11	bdps	Beidou positioning system
12	mif	Multi-sensor information fusion
13	atr	Automatic target recognition
14	vns	Visual navigation system
15	vans	Visual-aided navigation system
16	cv	Computer vision
17	imaf	Image fusion
18	cns	Combined navigation system
19	kf	Kalman filter
20	ekf	Extended Kalman Filter
21	ukf	Unscented Kalman Filter
22	pf	Particle Filter
23	ls	Least square

## 8 中心代码实例

### 8.1 \*.h 的编写实例

```

/*
*Copyright (C), 2008-2009, DNC BUAA
*File name dncplaymodel.h
Author:
    liuweifeng
Version: 0.5
Date:
    2009.03.15

```

#### Description:

视频播放模块接口声明。本模块实现 3D Tracker 程序和第三方的视频播放 SDK 的对接。通过本模块

就可以使实现播放多种视频格式的功能，而不必重新编译 3D Tracker 程序。

3D Tracker 程序、本模

块和第三方的视频播放模块之间的关系如下：



#### Function List:

DNC\_PlayInit 播放模块初始化

DNC\_GetMaxChl 获得播放模块支持的最大播放路数，

DNC\_PlayDestroy 销毁播放器模块

DNC\_Play 启动播放

DNC\_Pause 播放暂停

DNC\_Stop 播放停止

DNC\_Fast 快速播放

```
DNC_Slow      慢速播放

*/

#ifndef __DNC_PLAY_MODEL_H__
#define __DNC_PLAY_MODEL_H__

#if defined DNC_DLL_EXPORT
#define DNCAPI __declspec(dllexport)
#else
#define DNCAPI __declspec(dllimport)
#endif

extern "C"
{
    /**
     * @brief 初始化播放模块
     * @param NONE
     * @return 0 模块初始化正常, -1 模块初始化失败
     */
    DNCAPI int DNC_PlayInit();

    /**
     * @brief 销毁播放器模块
     * @param NONE
     * @return void
     */
    DNCAPI void DNC_PlayDestroy();

    /**
```

```
* @brief 获得播放模块的描述
* @param description 存储描述信息的缓冲区，不能小于 256
* @return void
*/
DNCAPI void DNC_GetPlayDescription(char description[256]);

/**
* @brief 获得播放模块支持的最大播放路数，
* @param NONE
* @return 播放模块支持的最大通道数
*/
DNCAPI void DNC_GetMaxChl();

/**
* @brief 启动播放
* @param nPort 播放通道号
* @return void
*/
DNCAPI int DNC_Play(unsigned long nPort);

/**
* @brief 暂停播放
* @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
* @return void
*/
DNCAPI int DNC_Pause(unsigned long nPort);

/**
* @brief 停止播放
```

```
* @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
* @return void
*/

DNC_API int DNC_Stop(unsigned long    nPort);

/**
* @brief 快速播放
* @param nPort 播放通道号
* @param iSpeed 快放的速度
* @return void
*/

DNC_API int DNC_Fast(unsigned long    nPort, int iSpeed);

/**
* @brief 慢速播放
* @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
* @param iSpeed 慢放的速度 iSpeed 1, 2, 3
* @return int 当前的播放速度
*/

DNC_API int DNC_Slow(unsigned long    nPort, int iSpeed);

}

#endif//__DNC_PLAY_MODEL_H__
```

## 8.2 \*.CPP 的编写实例

```
/*
*Copyright (C), 2008-2009, DNC BUAA
*File name dncplaymodel.cpp
Author:
```



liuweifeng

Version: 0.5

Date:

2009.03.15

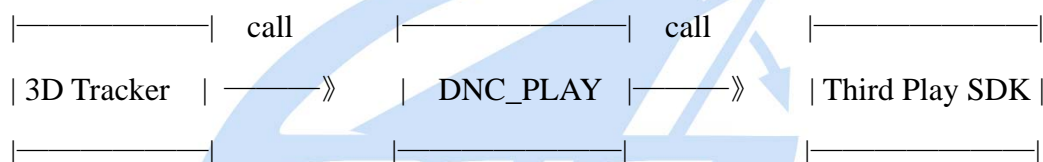
Description:

视频播放模块接口声明。本模块实现 3D Tracker 程序和第三方的视频播放 SDK 的对接。通过本模块

就可以使实现播放多种视频格式的功能，而不必重新编译 3D Tracker 程序。

3D Tracker 程序、本模

块和第三方的视频播放模块之间的关系如下：



Function List:

DNC\_PlayInit 播放模块初始化

DNC\_GetMaxChl 获得播放模块支持的最大播放路数，

DNC\_PlayDestroy 销毁播放器模块

DNC\_Play 启动播放

DNC\_Pause 播放暂停

DNC\_Stop 播放停止

DNC\_Fast 快速播放

DNC\_Slow 慢速播放

\*/

/\*定义函数导出标示\*/

#define DNC\_DLL\_EXPORT

#include "dncplaymodel.h"

```
extern "C"
{
    /**
     * @brief 初始化播放模块
     * @param NONE
     * @return 0 模块初始化正常，-1 模块初始化失败
     */
    DNCAPI int DNC_PlayInit()
    {
        return 0;
    }

    /**
     * @brief 获得播放模块的描述
     * @param description 存储描述信息的缓冲区，不能小于 256
     * @return void
     */
    DNCAPI void DNC_GetPlayDescription(char description[256])
    {

    }

    /**
     * @brief 获得播放模块支持的最大播放路数，
     * @param NONE
     * @return 播放模块支持的
     */
    DNCAPI void DNC_GetMaxChl()
```

```
{

}

/**
 * @brief 销毁播放器模块
 * @param NONE
 * @return void
 */
DNC_API void DNC_PlayDestroy()
{

}

/**
 * @brief 启动播放
 * @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
 * @return int 返回当前操作的通道, -1 表示错误
 */
DNC_API int DNC_Play(unsigned long nPort)
{
    return 0;
}

/**
 * @brief 暂停播放
 * @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
 * @return int 返回当前操作的通道, -1 表示错误
 */
DNC_API int DNC_Pause(unsigned long nPort)
```

```
{
    return 0;
}

/**
 * @brief 停止播放
 * @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
 * @return int 返回当前操作的通道, -1 表示错误
 */
DNC_API int DNC_Stop(unsigned long nPort)
{
    return 0;
}

/**
 * @brief 快速播放
 * @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
 * @param iSpeed 慢放的速度 iSpeed 1, 2, 3
 * @return int 当前的播放速度
 */
DNC_API int DNC_Fast(unsigned long nPort, int iSpeed)
{
    return 0;
}

/**
 * @brief 慢速播放
 * @param nPort 操作的通道号 nPort >= 0, nPort < maxchl
 * @param iSpeed 慢放的速度 iSpeed 1, 2, 3
```

```
* @return int 当前的播放速度
*/
DNCAPI int DNC_Slow(unsigned long    nPort, int iSpeed)
{
    return 0;
}
}
```

