



# WalkLan 开发手册

---

(对象化版本)

浙大万维科技有限公司

2011 年 5 月

# 目录

目录.....	1
第一章 概述.....	3
1、引言.....	3
2、WalkLan 涵义.....	3
3、编写目的.....	4
4、术语解释.....	4
5、脚本应用举例.....	5
第二章 结构和语法.....	6
1、基本语法说明.....	6
2、变量定义.....	8
3、基本运算符.....	10
4、流程控制.....	12
5、宏.....	22
第三章 标准函数类.....	27
1、main 函数.....	27
2、用户定义函数.....	28
3、系统标准函数.....	28
4、string：字符串.....	34
5、array：数组.....	36
6、wkPoint：三维坐标点.....	39
第四章 实体类对象.....	40
1、wkView：图形主窗口.....	40
2、wkGeoset：工作空间.....	44
3、wkDb：数据库.....	45
4、wkLayer：图层.....	48
5、wkFeature：地物.....	52
6、wkAnnotation：文字和影像.....	53
7、wkStyle：式样.....	54
8、wkGeometry：几何体.....	55
9、wkPolygon：多边形.....	59
10、wkParts：线流.....	60
11、wkPoints：线串.....	62
第五章 工具类对象.....	64
1、wkBox：矩形盒.....	64
2、wkGis：GIS 工具.....	65
3、wkClean：线素整理和构面.....	66
4、wkFile：读写文件.....	68
5、wkRgn：区域操作.....	70
6、wkDialog：自定义对话框.....	72

---

7、wkTrans：坐标变换.....	75
附录 1：常量定义.....	79
1、CMD 命令.....	79
2、3x3 定位常量表-wc3x3Constant.h.....	82
3、wkView 常量表-wcViewConstant.h.....	83
4、坐标系统和尺度单位常量表-wcSRSConstant.h.....	85
5、wkLayer 常量表- wcLayerConstant.h.....	87
6、wkAnnotation 常量表-wcAnnoConstant.h.....	88
7、wkStyle 常量表-wcStyleConstant.h.....	90
8、式样颜色常量表-wcStyleColors.h.....	92
9、wkGeometry 常量表-wcGeomConstant.h.....	93
10、wkFile 等工具类常量表-wcToolConstant.h.....	94
11、wkDialog 常量表-wcDialogConstant.h.....	98
附录 2：脚本错误代码.....	102
1、编译错误.....	102
2、运行错误：.....	105
3、系统限制.....	106
附录 3：开发工具.....	107
附录 4：脚本语言的特点.....	108
1、WalkLan 和 c 语言代码对比.....	108
2、Console 控制台.....	110
3、脚本对话框.....	111
4、使用脚本响应层编辑和数据存取.....	114
附录 5 样例及说明.....	120

## 第一章 概述

### 1、引言

地理信息系统技术日趋成熟,工具类产品被广泛使用,如 ArcGIS 和 WalkGIS 等系统平台在空间数据库管理和空间数据采集、加工等领域发挥着越来越大的作用。从本世纪初以来面向对象地理信息系统技术深入人心,并随着城市空间基础设施建设的需求“对象-关系”数据模型技术得到长足的发展,已成为市场主流。

以 WalkGIS 系统为例,完全采用商用关系型数据库作为数据载体,承载空间数据和属性数据为一体,实现了空间数据与业务系统无缝衔接,为巨型业务系统的空间数据挖掘、采集和更新提供了坚实基础,包括实用工具和便易的二次开发手段。

等同于 Oracle 和 Sql Server2008, WalkGIS 在数据库中增加了符合国际开放空间信息协会(OGC)标准的 Geometry 数据类型(OGC 的 WKB--Well Known Binary 结构),为此 WalkGIS 提供了标准查询语言扩展 WalkSQL,提供了二次开发组件 WalkXb 和 C++二次开发包 WalkLt,并在 2008 年首次提供了脚本语言 WalkScriptor。

基于特定系统的脚本语言是一种边解释边执行的语言,为各层次用户提供了简炼的专业领域技术和清晰的应用接口。在许多 GIS 计算和图形处理上,因可借助系统内部函数、数据结构和数据存储等,WalkScriptor 的代码量少于编程语言的代码量,运行效率相当高。

WalkGIS 是一个拥有自主产权的国产 GIS 平台,经近十年的开发和应用,技术优势凸显,承担了国家 863 计划、星火计划、南极考察、国务院二次土地调查、科技部国产地理信息系统平台等重大项目,是国内多所重点大学的 GIS 教学软件平台,是众多中国甲级测绘院的生产平台,广泛应用于国土、房地产和规划等地理信息相关行业。

多行业多层次的应用需求促使 WalkGIS 平台提供更为丰富更系统性的二次开发接口和工具。WalkLan 显现了 Walk 的一些优势技术,如图形符号化、几何计算和拓扑关系、图属一体化和 WalkSQL 等。

与 ArcView 的 Avenue、MapInfo 的 MapBasic 语言不同,WalkLan 类似于 c/c++/java 语言,更为简炼,结构清晰,是基于 GIS 数据特点的开发语言。

本开发手册包括 WalkLan 脚本语言的结构和语法、变量类型、标准函数、GIS 实体类对象、工具类对象等说明和简单样例,以及常量定义、脚本开发环境、脚本错误代码等附录。对象化开发更安全,效率更高,功能涵盖更加全面,因此本开发手册完全集中在对象化开发方法的阐述,不再介绍脚本的 Ws 函数集。

本手册的第二章和第三章主要介绍 WalkLan 脚本语言的基本结构和语法,以及标准函数,若阅读者已具有 c/c++或 java 的基本知识,该两章简单浏览即可。第四章为 GIS (OGC 推荐)对象在 Walk 平台中的实例化应用,讲述了空间数据库、层表、地物和几何体等实例构造和操作的方法。第五章为 GIS 工具箱通过对象化实例进行应用的方法,包括坐标变换、地图投影、线素整理和构面、空间分析和网络分析等各种应用。本手册包含了四个附录供阅读者参考。

### 2、WalkLan 涵义

WalkLan 是在 Walk 平台基础上构建起的一个 Walk 语言解释器。该解释器依附于 Walk

软件产品，可以在 Walk 产品环境中运行，并且能脱离主体软件单独使用。

### 3、编写目的

通过 Lan 技术，可以为用户二次开发（特别是一些短期的计算和数据处理任务）提供接口；它不需要为安装复杂系统消耗大量的时间，免去了大炮打蚊子的尴尬，更加方便用户利用 WalkLan 语言进行二次开发，也能更深层次地理解 Walk 产品的基本功能与实现过程。

假定本手册的读者对 c 或 c++ 已有初步了解。

### 4、术语解释

下列术语和定义适用于本手册。

#### 1) 工作空间 (Wks)

所有对象的组织者，任何 Walk 数据对象都由此获得入口并进行操作。

#### 2) 数据库(WalkDb)

所有对象与数据进行有效组织管理的一种数据组织形式与管理工具。

#### 3) SQL(WalkSQL)

实现数据库表中数据的查询与存储功能。

#### 4) 层(Layer)

按照专题要求组织起来的数据集合，一般具有相同的属性结构。

#### 5) 地物(Feature)

现实世界中地理事物的抽象与概括。

#### 6) 标注(Annotation)

用于对地物等对象进行信息传达与特征描述。

#### 7) 样式(Style)

通常包含地物的符号化信息，如线型、线宽、颜色、面填充色等。

#### 8) 文字(Text)

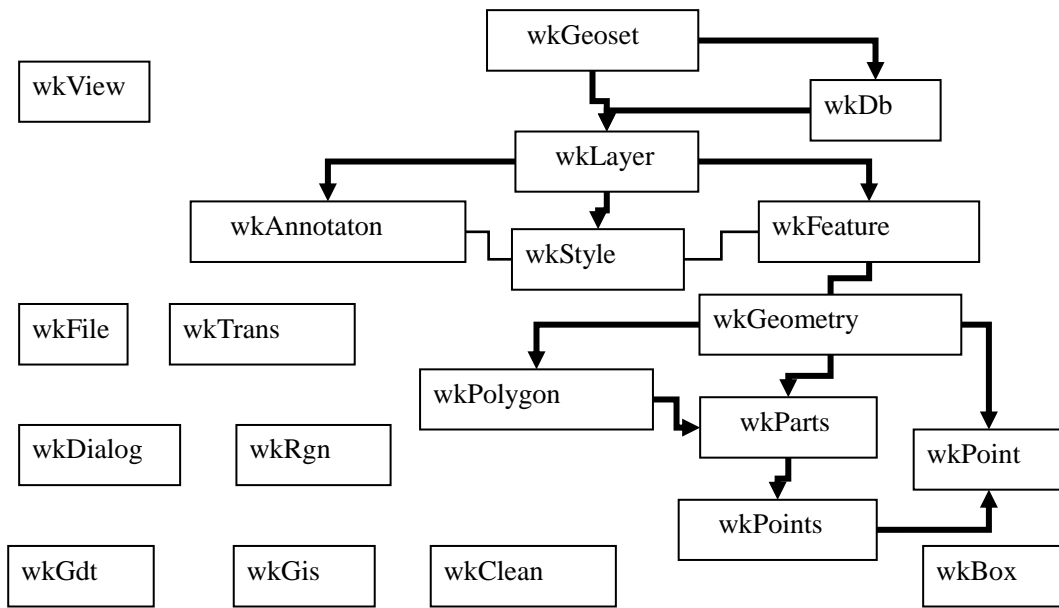
用于表述对象性质的文本信息。

#### 9) 地物几何(Geometry)

用于区分描述层中地物的几何特征，包括地物中的点、地物几何中的多边形及地物几何中的部分等内容。

10) WalkGis 平台的基本对象关系如下：

```
<wkView 视图窗口> ::= <wkGeoset 工作空间 (work space -- wks)>
<wkGeoset> ::= {<wkLayer 图层>} {<wkDb 数据库>}
<wkDb> ::= {库表} {属性字段} {SQL} {WalkProcedure}
<wkLayer> ::= {<wkFeature 地物>} {<wkStyle 式样>} {<wkAnnotation 文字>}
<wkFeature> ::= {属性值} {<wkGeometry 地物几何体>}
<wkGeometry> ::= {<wkPoint 点>} {<wkParts 线>} {<wkPolygon 多边形>}
<wkPolygon> ::= {<wkParts 环 Ring>}
<wkParts> ::= {<wkPoints 线串>}
<wkPoints> ::= {<wkPoint 点>}
```



## 5、脚本应用举例

【example】:

//计算 1+2+3+。。。 +100=?

void main()

{

int sum=0;

for(int i=1;i<=100;i++)

{

sum+=i;

}

message(toString(sum));//输出运算结果;

}

结果:



message(toString(sum));

## 第二章 结构和语法

### 1、基本语法说明

#### 1) 保留字

define, null, bool, int, double, string, array, wkPoint, if, else if, else, switch, case, default, for, while, break, continue, void, function, return

#### 2) 大小写识别

该解释器同 c 一样，有着严格的大小写区分。

#### 3) 程序结构控制语句

分支语句:

二分支 if() ... else if() ... else ...

多分支 switch() ... case

else if 部分或 else 部分也可以省略

case 中的 default 语句也可以省略

循环语句:

for();

while();

函数返回语句:

return;

其他语句:

break continue

#### 4) 字符串常量

字符串常量用“...”表示。

#### 5) 转义字符

常见的转义符号及含义如下:

\t 横向跳格

\n 换行

\\ 反斜杠

\b 退格

<code>\r</code>	回车
<code>\"</code>	双引号
<code>\'</code>	单引号

例如，要在屏幕上显示`\\`，则在代码中需表示为`\\\\`。

## 6) 数据类型格式控制说明

<code>%d</code>	以十进制输入输出一个整数
<code>%o</code>	以八进制输入输出一个整数
<code>%x</code>	以十六进制输入输出一个整数
<code>%f</code>	以小数形式输入输出一个单精度浮点数
<code>%lf</code>	以小数形式输入输出一个双精度浮点数
<code>%s</code>	以字符串形式输入输出字符串变量

输出格式控制说明中：

`%md` 表示指定了整型数据的输出宽度为 `m`（包括符号位），若数据的实际位数（含符号位）小于 `m`，则左端补空格；若大于 `m`，则按实际位数输出。如`%5d` 输出数据的值为 17，左端补了 3 个空格。

`%m.nf` 表示指定输出浮点型数据时，保留 `n` 位小数，且输出宽度是 `m`（包括符号位和小数点）。若数据的实际位数小于 `m`，左端补空格；若大于 `m`，按实际位数输出。输出变量的值为 3.1415926，`%5.3f` 输出 3.142（保留三位小数）。

同 c 语言中 `printf` 的格式规定。

## 7) 注释

WalkLan 的注释使用形式有两种：

<code>/* */</code>	段落注释，用以注释包含在注释符对内的所有程序代码
<code>//</code>	单行注释，注释符在所注释代码行的同行前面

## 8) 递归函数

该解释器能够执行递归函数，参见附录 4 的“典型的递归用法——枚举目录下文件”

## 9) 命名规则

`for,if,while` 等循环和条件语句，`cos,sin,pow` 等内部函数以及`+, -, ==, >=`等都采用了 c 的语法规则和命名，底层提供的两个类型 `string` 和 `array` 的成员函数采用的是 stl 的 camel 命名规则，即命名中第一字母小写。

WalkScript 中支持全对象化编程，从 walk 平台引出的功能归于 19 个类。`wk` 类在脚本中类似于 `string` 和 `array`，也是变量类型，因此 `wk` 成员函数的命名亦同样采用 camel 命名规

则。

如，获取当前主窗口中心点的实际坐标：

```
wkView view;  
wkPoint center = view.getCenter();
```

## 10) 与 c/c++的异同

WalkLan 支持所有的 c 操作符。

支持的 c 数据类型有：void, bool, int, double

不支持的 c 数据类型有：char, long, short, float, unsigned, signed, const 等，不支持所有 MS 扩展的数据类型（如:BYTE, DWORD 等）

支持扩展数据类型有：string, array, wkPoint。

支持的 walk 对象有：wkBox, wkView, wkGeoset, wkDb, wkLayer, wkFeature, wkAnnotation, wkStyle, wkGeometry, wkPolygon, wkParts, wkPoints, wkFile, wkRgn, wkGis, wkTrans, wkClean, wkDialog, wkGdt

支持函数中引用参数，如: void func(int i, array &ar, wkDb &db)

支持宏：#define, #undef

支持宏：#ifdef, #ifndef, #else, #endif

支持宏：#include

不支持指针，不支持 new, 不支持 delete

不支持自定义结构 struct

不支持自定义类 class

支持条件语句 if ... else ... 和 switch

不支持 goto（因此也不支持标号）

支持循环语句 for(;;) 和 while()

不支持 do ... while

变量和对象实例作用域定义同 c 和 c++语言

支持变量定义时的初始化，如: int a=0;

脚本被限制在 walk 环境中运行，因此不支持 console 标准输入与输出。

不支持的 c 和 c++保留字（部分）：

const, static, goto, do, enum, extern, new, delete, struct, this, class, public, private, protected, virtual, ...

## 2、变量定义

### 1) 数据类型

系统的基本数据类型有五种：整型、浮点型、字符型、布尔型和空值型，其中字符型由字符串数据类型表达，以及一个数组数据类型，具体数据类型定义及说明如下表所示：

数据类型名	定义类型名	类型编号	说明
空	void	-1	同 C 中的 void
布尔型	bool	0	值为: true 或 false
整型	int	1	同 C 中的 int
实型	double	2	同 C 中的 double
字符串	string	3	类似 MFC 中的 CString
数组	array	4	类似 MFC 中的 CArray
三维点坐标	wkPoint	5	同 walk 中的 Point

说明：系统中无单字符类型（无 char），所有单字符按字符串表达。

系统内置对象数据类型：

数据类型名	定义类型名	类型编号	说明
工作空间	wkGeoset	7	多库多层的用户工程组织
数据库	wkDb	8	普通关系型数据库
层	wkLayer	9	地理信息系统概念上的图层
地物	wkFeature	10	客观世界上的实体
文字和影像	wkAnnotation	11	标注和遥感影像及其定位
式样	wkStyle	12	地物和文字等的符号表现形式
几何体	wkGeometry	13	点线面地物的几何形状和位置等
多边形	wkPolygon	14	可含洞的面
线	wkParts	15	由多线串构成的点流
线串	wkPoints	16	同线型的点流，如 3 点弧、折线、曲线等
矩形盒	wkBox	17	实数矩形的左下角和右上角
自定义对话框	wkDialog	18	无模态对话框，可与系统交互
区域	wkRgn	19	由实数构造的 GDI+ 的 Region
文件	wkFile	20	等价于 c 标准库中 FILE
线素整理	wkClean	21	提供 GIS 中的 clean 和 build 功能
GIS 分析	wkGis	22	空间分析和网络分析
坐标变换	wkTrans	23	高斯与经纬度坐标变换、同名点集坐标系变换、地图投影

## 2) 变量的定义

变量定义的具体格式：

<变量数据类型> <变量名称>;

例如：

int i, j;

wkGeoset geoset;

定义在函数体内或由 ‘{’ 和 ‘}’ 构成的程序块内的变量称为自由变量或叫局部变量，这些变量出了函数体或程序块后定义失效，变量便不得再使用。

定义在函数体外的变量称为全局变量，在其定义后可有效使用。若全局变量与局部变量

重名，则函数体内或程序块内的局部变量有效。

### 3) 变量初始化

在变量声明中可赋值，如：

```
string strLayerName="JZX";
for (int i=0; i<n; i++) ...
```

### 4) 混合类型声明

一些函数，如 `toString`, `getDataType` 等，以及 `array` 类的成员函数和返回值为多类型之一，即可以是 `bool`, `int`, ..., `wkPoint`, ..., `wkGeometrey`, ..., `wkTrans` 等所有 WalkLan 支持的变量类型和对象类型，因此在 "wkClassDefine.h"（参见附录“开发工具”——使用 Visual Studio 编写脚本和语法检查）声明为 `wkMultiType`。

混合类型仅在头文件中声明，主要为 c/c++ 环境编译和编辑脚本代码查错服务，在实际运行时系统将自动为脚本配置合法的数据类型。

如，`toString` 声明为：

```
string toString(wkMultiType v);
```

使用时，脚本写为：

```
message(toString(10));
```

```
wkPoint point;
```

```
message(toString(point));
```

这两条语句在 c/c++ 环境中可无错编译，在 walk 环境下可正确运行。

## 3、基本运算符

WalkLan 二次开发的基本运算主要包括比较运算、逻辑运算及四则运算，运算过程也遵循一定的优先级，按照优先级的增减由高到低依次进行。

### 1) 比较运算

如果两个数据类型不同，则比较类型编号；如果两个数据类型相同，与 c 语言中的比较规则相同；比较时，`bool`, `int`, `double` 三种类型可以相互转换；比较返回值是 `bool`。

比较操作符包括：

操作符	比较操作
<code>a&gt;b</code>	大于
<code>a&lt;b</code>	小于
<code>a&gt;=b</code>	大于或等于
<code>a&lt;=b</code>	小于或等于
<code>a==b</code>	等于
<code>a!=b</code>	不等于

说明 1：当两个 `array` 比较时：依次比较两个 `array` 对应的元素，直到找到两个不相同的元素为止。如果一个 `array` 是另一个 `array` 的前缀，则短的 `array` 小。

说明 2: 当两个 point 比较时, 先比较 x, 如果 x 相同则比较 y, 如果 x 和 y 都相同再比较 z 的大小。

## 2) 四则运算

计算时, 两数的类型应一致, 如必须都是整数或者实数, 两数只要有一个数字是实数, 结果就返回实数, 否则返回整数。

四则运算符包括:

操作符	说明
<code>a=b</code>	赋值
<code>? a : b</code>	条件取值, 如: <code>int ab_min = a&lt;b? a:b;</code>
<code>a+b, a+=b</code>	两数相加
<code>a-b, a-=b</code>	两数相减
<code>-a</code>	取相反数
<code>a*b, a*=b</code>	两数相乘
<code>a/b, a/=b</code>	两数相除
<code>++, --</code>	加 1 和减 1
<code>a%b, a%=b</code>	整数取余

## 3) 位运算对整数的位操作。

操作符	说明
<code>a b, a =b</code>	两值或
<code>a&amp;b, a&amp;=b</code>	两值与
<code>a&lt;&lt;b, a&lt;=&lt;b</code>	对 a 左移 b 位
<code>a&gt;&gt;b, a&gt;=&gt;b</code>	对 a 右移 b 位
<code>a^b, a^=b</code>	对 a 按 b 异或
<code>~a</code>	对 a 取反 (按位取反)

## 4) 逻辑运算

两个参数可以同时为 bool 或者同时为 int 类型。

逻辑运算符包括:

操作符	比较操作
<code>a    b</code>	或
<code>a &amp;&amp; b</code>	与
<code>!a</code>	非

## 5) 优先级

运算的优先顺序符合标准 (同 c/c++/java) 的运算优先级, 优先级分类如下:

(a) `<= > >= < == !=` (比较函数)

- (b) + - (加减) || (或)
- (c) \* /(乘除) ^(异或) & &&(与)
- (d) ~ !(反)

说明：由 a)到 d)，优先级依次增加。

当然，若编程者对优先级不肯定时可使用 ‘(’ 和 ‘)’ 保证自己的预期优先级。

## 4、流程控制

不论哪一种编程语言，都会提供两种基本的流程控制结构：分支结构和循环结构。其中分支结构用于实现根据条件来选择性地执行某段代码，循环结构则用于根据循环条件重复执行某段代码。WalkLan 同样提供了这两种流程控制结构的语法，WalkLan 提供了 if 和 switch 两种分支语句，并提供了 while 和 for 两种循环语句。

### 1) 顺序结构

任何编程语言中最常见的程序结构就是顺序结构。顺序结构就是程序从上到下一行一行地执行，中间没有任何判断和跳转。

如果 main 方法多行代码之间没有任何流程控制，则程序总是从上向下依次执行，排在前面的代码先执行，排在后面的代码后执行。这意味着：如果没有流程控制，WalkLan 方法里的语句是一个顺序执行流，从上向下依次执行每条语句。

### 2) if 条件语句

if 语句使用布尔表达式或布尔值作为分支条件来进行分支控制，其中 if 语句有如下三种形式：

第一种形式：

```
【example】:  
if ( logic expression )  
{  
    statements...  
}
```

第二种形式：

```
if (logic expression)  
{  
    statements...  
}  
else  
{  
    statements...  
}
```

第三种形式：

```
【example】:  
if (logic expression)
```

```

{
    statements...
}
else if(logic expression)
{
    statements...
} //可以有零个或多个 else if 语句
else
{
    statement..
} //最后的 else 语句也可以省略

```

在上面 if 语言的三种形式中，放在 if 之后的括号里的只能是一个逻辑表达式，即这个表达式的返回值只能是 true 或 false。第二种情形和第三种情形是相通的，如果第三种形式中 else if 块不出现，则变成了第二种形式。

上面的条件语句中，if(logic expression)、else if(logic expression)以及 else 后花括号括起来多行代码被称为代码块，一个代码块通常被当成一个整体来执行（除非运行过程中遇到 return、break、continue 等关键字，或者遇到了异常），因此这个代码块也被称为条件执行体。例如如下程序：

**【example】:**

```

void main()
{
    int age = 30;
    if (age > 20) //只有当 age > 20 时，下面花括号括起来的语句块才会执行
    {
        message("年龄已经大于 20 岁了");
        message("20 岁以上的人应该学会承担责任");
    } //花括号括起来的语句是一个整体，要么一起执行，要么一起不会执行
}

```

因此，如果 if(logic expression)、else if(logic expression)和 else 后的语句块只有一行语句时，则可以省略花括号，因为单行语句本身就是一个整体，无须花括号来把它们定义成一个整体。下面代码完全可以正常执行：

**【example】:**

```

int a = 5; //定义变量 a，并为其赋值
if (a > 4) //如果 a > 4，执行下面的执行体，只有一行代码作为代码块
    message("a 大于 4");
else //否则，执行下面的执行体，只有一行代码作为代码块
    message("a 不大于 4");

```

通常，我们建议不要省略 if、else、else if 后执行块的花括号，即使条件执行体只有一行代码，因为保留花括号会有更好的可读性，而且保留花括号会减少发生错误的可能，例如如下代码，则不可正常执行：

**【example】:**

```

int b = 5; //定义变量 b，并为其赋值
if (b > 4) //如果 b > 4，执行下面的执行体，只有一行代码作为代码块

```

```

message("b 大于 4");
else//否则，执行下面的执行体
    b--;//对于下面代码而言，它已经不再是条件执行体的一部分，因此总会执行
message("b 不大于 4");

```

上面代码中以粗体字标识的代码行：`message("b 不大于 4");`，将总是会执行，因为这行代码并不属于 `else` 后的条件执行体，`else` 后的条件执行体就是 `b--;`这行代码。

`if`、`else`、`else if` 后条件执行体要么是一个花括号扩起来的语句块，则这个语句块整体作为条件执行体；要么是以分号为结束符的一行语句，甚至可能是一个空语句（空语句是一个分号）。

如果 `if` 块后有多条语句作为条件执行体，如果省略了这个条件执行体的花括号，则会引起编译错误，看下面代码：

**【example】:**

```

int c = 5; //定义变量 c ， 并为其赋值
if (c > 4) //如果 c>4，执行下面的执行体，将只有 c--;一行代码为条件执行体
    c--;//下面是一行普通代码，不属于条件执行体
message("c 大于 4");//此处的 else 将没有 if 语句，因此编译出错
else //否则，执行下面的执行体，只有一行代码作为代码块
    message("c 不大于 4");

```

在上面代码中，因为 `if` 后的条件执行体省略了花括号，则系统只把 `c--;`一行代码作为条件执行体，当 `c--;`语句结束后，`if` 语句也就结束了。后面的 `message("c 大于 4");`代码已经是一行普通代码了，不再属于条件执行体，从而导致 `else` 语句没有 `if` 语句，从而引起编译错误。

对于 `if` 语句，还有一个很容易出现的逻辑错误，这个逻辑错误并不属于语法问题，但引起错误的可能性更大。看下面程序：

**【example】:**

```

void main()
{
    int age = 45;
    if (age > 20)
    {
        message("青年人");
    }
    else if (age > 40)
    {
        message("中年人");
    }
    else if (age > 60)
    {
        message("老年人");
    }
}

```

表面上看起来，上面的程序没有任何问题：人的年龄大于 20 岁时是青年人，年龄大于 40 岁是中年人，年龄大于 60 岁是老年人。但运行上面程序，发现打印结果是：青年人，而实际上我们希望 45 岁应判断为中年人——这显然出现了一个问题。

对于任何的 if else 语句，表面上看起来 else 后没有任何条件，或者 else if 后只有一个条件——但这不是真相：因为 else 的含义是“否则”——else 本身就是一个条件！这也是把 if、else 后代码块统称为条件执行体的原因，else 的隐含条件对前面条件取反。因此上面代码实际上可改写为：

【example】：

```
void main()
{
    int age = 45;
    if (age > 20)
    {
        message("青年人");
    } //在原本的 if 条件中增加了 else 的隐含条件
    else if (age > 40 && !(age > 20))
    {
        message("中年人");
    } //在原本的 if 条件中增加了 else 的隐含条件
    else if (age > 60 && !(age > 20) && !(age > 40 && !(age > 20)))
    {
        message("老年人");
    }
}
```

此时就比较容易看出为什么发生上面的错误了，对于 `age > 40 && !(age > 20)` 这个条件，又可改写成 `age > 40 && age <= 20`，这样情况永远也不会发生了。对于 `age > 60 && !(age > 20) && !(age > 40 && !(age > 20))` 这个条件，则更不可能发生了。因此，无论如何，程序永远都不会判断中年人和老年人的情形。

为了达到正确的目的，我们把程序改写成如下形式：

【example】：

```
void main()
{
    int age = 45;
    if (age > 60)
    {
        message("老年人");
    }
    else if (age > 40)
    {
        message("中年人");
    }
    else if (age > 20)
    {
        message("青年人");
    }
}
```

运行程序，得到了正确结果。实际上，上面程序等同于下面代码：

```

【example】:
void main()
{
    int age = 45;
    if (age > 60)
    {
        message("老年人");
    } //在原本的 if 条件中增加了 else 的隐含条件
    else if (age > 40 && !(age > 60))
    {
        message("中年人");
    } //在原本的 if 条件中增加了 else 的隐含条件
    else if (age > 20 && !(age > 60) && !(age > 40 && !(age > 60)))
    {
        message("青年人");
    }
}

```

上面程序的判断逻辑即转为如下三种情形：

age 大于 60 岁，判断为“老年人”。

age 大于 40 岁，且 age 小于等于 60 岁，判断为“中年人”。

age 大于 20 岁，且 age 小于等于 40 岁，判断为“青年人”。

上面的判断逻辑才是实际希望的判断逻辑。因此，当我们使用 if...else 语句进行流程控制时，一定不要忽略了 else 所带的隐含条件。

如果每次都去计算 if 条件和 else 条件的交集也是一件非常烦琐的事情，为了避免出现上面的错误，在使用 if...else 语句有一条基本规则：总是优先把包含范围小的条件放在前面处理。如 age>60 和 age>20 两个条件，明显 age>60 的范围更小，所以应该先处理 age>60 的情况。

使用 if...else 语句时，一定要先处理包括范围更小的情况。

### 3) switch 分支语句

switch 语句由一个控制表达式和多个 case 标签组成，和 if 语句不同的是，switch 语句后面的控制表达式的数据类型只能是整型或字符串，不能是 boolean 型。case 标签后紧跟一个代码块，case 标签作为这个代码块的标识。switch 语句的语法格式如下：

```

【example】：
switch (expression)
{
    case condition1:
    {
        statement(s)
        break;
    }
    case condition2:

```

```
{
    statement(s)
    break;
}
...
case conditionN:
{
    statement(s)
    break;
}
default:
{
    statement(s)
}
}
```

这种分支语句的执行是先对 `expression` 求值，然后依次匹配 `condition1`，`condition2...conditionN` 等值，遇到匹配的值即执行对应的执行体；如果所有 `case` 标签后的值都不与 `expression` 表达式的值相等，则执行 `default` 标前后的代码块。

和 `if` 语句不同的是，`switch` 语句中各 `case` 标签前后代码块的开始点和结束点非常清晰，因此完全可以省略 `case` 后代码块的花括号。与 `if` 语句中 `else` 类似，`switch` 语句中 `default` 标签看似没有条件，其实是有条件的：条件就是 `expression` 表达式的值不能与前面任何一个 `case` 标签后的值相等。

下面程序示范了 `switch` 语句的用法：

**【example】：**

```
void main()
{
    string score = "c" ; //声明变量 score，并为其赋值为"C"
    switch (score) //执行 swicth 分支语句
    {
        case "A":
            message("优秀.");
            break;
        case "B":
            message("良好.");
            break;
        case "C":
            message("中");
            break;
        case "D":
            message("及格");
            break;
        case "F":
            message("不及格");
            break;
        default:
```

```

        message("成绩输入错误");
    }
}

```

运行上面程序，看到输出“中”，这个结果完全正常，字符表达式 `score` 的值为 'C'，对应结果为“中”。

值得指出的是，`switch` 语句中控制表达式的类型可以是 `int` 和 `string` 与 C# 中相同。

在 `case` 标签后的每个代码块后都有一条 `break;` 语句，这个 `break;` 语句有极其重要的意义，WalkLan 的 `switch` 语句允许省略 `case` 后代码块的 `break;` 语句，但这种省略可能引入一个陷阱。如果我们把上面程序中的 `break;` 语句都注释掉，将看到如下运行结果：

```

中
及格
不及格
成绩输入错误

```

这个运行结果看起来比较奇怪，但这正是由 `switch` 语句的运行流程决定的：`switch` 语句会先求出 `expression` 表达式的值，然后拿这个表达式和 `case` 标签后的值进行比较，一旦遇到相等的值，程序开始执行这个 `case` 标签后代码，不再判断与后面 `case`、`default` 标签的条件是否匹配，除非遇到 `break;` 才会结束。

使用 `switch` 语句时，有两个值得注意的地方：第一个地方是 `switch` 语句后的 `expression` 表达式的数据类型可以是 `int` 和 `string` 类型；第二个地方是如果省略了 `case` 后代码块的 `break;` 时所引入的陷阱。

## 4) while 循环语句

循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体。当反复执行这段循环体时，需要在合适的时候把循环条件改为假，从而结束循环，否则循环将一直执行下去，形成死循环。循环语句可能包含如下四个部分：

**初始化语句（init\_statements）：**一条或多条语句，这些代码用于完成一些初始化工作，初始化语句在循环开始之前执行。

**循环条件（test\_expression）：**这是一个 `boolean` 表达式，这个表达式能决定是否执行循环体。

**循环体（body\_statements）：**这个部分是循环的主体，如果循环条件允许，这个代码块将被重复执行。如果这个代码块只有一行语句，则这个代码块的花括号是可以省略的。

**迭代语句（iteration\_statements）：**这个部分在一次循环体执行结束后，对循环条件求值之前执行，通常用于控制循环条件中的变量，使得循环在合适时候结束。

上面四个部分只是一般分类，并不是每个循环中都非常清晰地分出了上面四个成分。

`while` 循环的语法格式如下：

```

[init_statements]
while(test_expression)
{
    statements;
    [iteration_statements]
}

```

}

while 循环每次执行循环体之前，先对 `test_expression` 循环条件求值，如果循环条件为 `true`，则运行循环体部分。从上面语法格式中来看，迭代语句 `iteration_statements` 总是位于循环体的最后，因此只有当循环体能成功执行完成时，while 循环才会执行 `iteration_statements` 迭代语句。

从这个意义上来看，while 循环也可被当成条件语句——如果 `test_expression` 条件一开始就为 `false`，则循环体部分将永远不会获得执行。

下面语句示范了一个简单的 while 循环：

```
【example】：
void main()
{
    int count = 0; //循环的初始化条件
    while (count < 10) //当 count 小于 10 时，执行循环体
    {
        message(toString(count));
        count++; //迭代语句
    }
    message("循环结束!");
}
```

如果 while 循环的循环体部分和迭代语句合并在一起，且只有一行代码，则可以省略 while 循环后的花括号。但这种省略花括号的作法，可能降低程序的可读性。

使用 while 循环时，一定要保证循环条件有变成 `false` 的时候，否则这个循环将成为一个死循环，永远无法结束这个循环。例如如下代码（程序清单同上）：

```
【example】：
int count = 0;
while (count < 10)
{
    message("不停执行的死循环 " + toString(count));
    count--;
}
message("永远无法跳出的循环体");
```

在上面代码中，count 的值越来越小，这将导致 count 值永远小于 10，count < 10 循环条件一直为 `true`，从而导致这个循环永远无法结束。

除此之外，对于许多初学者而言，使用 while 循环时还有一个陷阱：while 循环的循环条件后紧跟一个分号。如果有如下程序片段：

```
【example】：
int count = 0;
while (count < 10); //while 后紧跟一个分号，表明循环体是一个分号（空语句）
//下面的代码块与 while 循环已经没有任何关系
{
    message("-----" + toString(count));
    count++;
}
```

乍一看上，这段代码片段没有任何问题，但仔细看一下这个程序，不难发现 `while` 循环的循环条件表达式后紧跟了一个分号。在 Java 程序中，一个单独的分号表示一个空语句，不作任何事情的空语句，这意味着这个 `while` 循环的循环体是空语句。空语句作为循环体也不是最大的问题，问题是当 Java 反复执行这个循环体时，循环条件的返回值没有任何改变，这就成了一个死循环。分号后面的代码块则与 `while` 循环没有任何关系。

## 5) for 循环

`for` 循环是更加简洁的循环语句，大部分情况下，`for` 循环可以代替 `while` 循环。`for` 循环的基本语法格式如下：

```
for ([init_statements]; [test_expression]; [iteration_statement])
{
    statements
}
```

程序执行 `for` 循环时，先执行循环的初始化语句 `init_statements`，初始化语句只在循环开始前执行一次。每次执行循环体之前，先计算 `test_expression` 循环条件的值，如果循环条件返回 `true`，则执行循环体部分，循环体执行结束后执行循环迭代语句。因此，对于 `for` 循环而言，循环条件总比循环体要多执行一次，因为最后一次执行循环条件返回 `false`，将不再执行循环体。

值得指出的是，`for` 循环的循环迭代语句并没有与循环体放在一起，因此即使在执行循环体时遇到 `continue` 语句结束本次循环，循环迭代语句一样会得到执行。

`for` 循环和 `while` 循环不一样：由于 `while` 循环的循环迭代语句紧跟着循环体，因此如果循环体不能完全执行，如使用 `continue` 来结束本次循环，则循环迭代语句不会被执行。但 `for` 循环的循环迭代语句并没有与循环体放在一起，因此不管是否使用 `continue` 来结束本次循环，循环迭代语句一样会获得执行。

与前面循环类似的是，如果循环体只有一行语句，循环体的花括号可以省略。下面使用 `for` 循环代替前面的 `while` 循环，代码如下：

```
【example】：
void main()
{
    //循环的初始化条件,循环条件，循环迭代语句都在下面一行
    for (int count = 0 ; count < 10 ; count++)
    {
        message(toString(count));
    }
    message("循环结束!");
}
```

在上面的循环语句中，`for` 循环的初始化语句只有一个，循环条件也只是一个简单的 `boolean` 表达式。实际上，`for` 循环允许同时指定多个初始化语句，循环条件也可以是一个包含逻辑运算符的表达式，例如如下程序：

```
【example】：
void main()
```

```
{
//同时定义了三个初始化变量，使用&&来组合多个 boolean 表达式
for (int b = 0, s = 0, p = 0; b < 10 && s < 4 && p < 10; p++)
{
    message(b++);
    message(++s + p);
}
}
```

上面代码中初始化变量有三个，但是只能有一个声明语句，因此如果需要在初始化表达式中声明多个变量，那么这些变量应该有相同的数据类型。

初学者使用 for 循环时也容易犯一个错误，他们以为只要在 for 后的括号内控制了循环迭代语句就万无一失，但实际情况则不是这样的，例如下面的程序：

```
【example】：
void main()
{
//循环的初始化条件,循环条件，循环迭代语句都在下面一行
for (int count = 0; count < 10; count++)
{
    message(count);
    count *= 0.1; //再次修改了循环变量
}
message("循环结束!");
}
```

在上面 for 循环中，表面上看起来控制了 count 变量的自加，count < 10 有变成 false 的时候。但实际上程序中粗体字标识的代码行在循环体内修改了 count 变量的值，并且把这个变量的值乘以了 0.1，这也会导致 count 的值永远都不能超过 10，因此上面程序也是一个死循环。

for 循环圆括号中只有两个分号是必须的，初始化语句、循环条件、迭代语句部分都是可以省略的，如果省略了循环条件，则这个循环条件默认是 true，将会产生一个死循环。例如下面程序：

```
【example】：
void main()
{
    for (;;) //省略了 for 循环三个部分，循环条件将一直为 true
    {
        message("=====");
    }
}
```

运行上面程序，将看到程序一直输出=====字符串，这表明上面程序是一个死循环。

使用 for 循环时，还可以把初始化条件定义在循环体之外，把循环迭代语句放在循环体内，这种做法就非常类似于前面的 while 循环了，下面的程序再次使用 for 循环来代替前面的 while 循环，程序如下：

```

【example】：
void main()
{
    int count = 0; //把 for 循环的初始化条件提出来独立定义
    for( ; count < 10 ; ) //for 循环里只放循环条件
    {
        message(count);
        count++; //把循环迭代部分放在循环体之后定义
    }
    message("循环结束!"); //此处将还可以访问 count 变量
}

```

上面程序的执行流程和前面的 TestWhile.java 程序的执行过程完全相同。因为把 for 循环的循环迭代部分放在循环体之后，则会出现与 while 循环类似的情形，如果循环体部分使用 continue 来结束本次循环，将会导致循环迭代语句得不到执行。

把 for 循环的初始化语句放在循环之前定义还有一个作用：可以扩大初始化语句中所定义的变量的作用域。在 for 循环里定义的变量，其作用域仅在该循环内有效，for 循环终止以后，这些变量将不可被访问。如果需要在 for 循环以外的地方使用这些变量的值，就可以采用上面的做法。除此之外，还有一种做法也可以满足这种要求：额外定义一个变量来保存这个循环变量的值。例如下面代码片段：

```

【example】：
int tmp = 0;
//循环的初始化条件,循环条件，循环迭代语句都在下面一行
for (int i = 0 ; i < 10 ; i++)
{
    message(count); 。
    tmp = i; //使用 tmp 来保存循环变量 i 的值
}
message("循环结束!"); //此处还可通过 tmp 变量来访问 i 变量的值。

```

相比之前，笔者更喜欢这种解决方案。使用一个变量 tmp 来保存循环变量 i 的值，使得程序更加清晰，变量 i 和变量 tmp 的责任更加清晰。

反之，如果采用前一种方式，则变量 i 的作用域被扩大了，功能也被扩大了。作用域扩大的后果是：如果该方法还有另一个循环也需要定义循环变量，则不能再次使用 i 作为循环变量。

选择循环变量时，习惯选择 i、j、k 来作为循环变量。

## 5、宏

### 1) #include

文件包含命令的一般形式为：

#include"文件名"

文件包含命令的功能是把指定的文件插入该命令行位置取代该命令行，从而把指定的文

件和当前的源程序文件连成一个源文件。

在程序设计中，文件包含是很有用的。一个大的程序可以分为多个模块，由多个程序员分别编程。有些公用的符号常量等可单独组成一个文件，在其它文件的开头用包含命令包含该文件即可使用。这样，可避免在每个文件开头都去书写那些公用量，从而节省时间，并减少出错。

对文件包含命令还要说明以下几点：

1. 包含命令中的文件名要用双引号括起来，文件名可以是全路径，也可以仅仅是文件名，如：

```
#include "c:\\walkgis\\WalkLan\\wcViewConstant.h"
#include "wcViewConstant.h"
```

但是这两种形式是有区别的：使用全路径表示在指定目录下去查找包含文件，若只给出文件名则从执行程序下的 WalkLan 目录下去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。

2. 一个 include 命令只能指定一个被包含文件，若有多个文件要包含，则需用多个 include 命令。

3. 文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

## 2) #ifdef 等宏

这几个宏是为了进行条件编译。一般情况下，程序中所有的行都参加运行。但是有时希望对其中一部分内容只在满足一定条件才解释运行，也就是对一部分内容指定运行的条件，这就是“条件解释”。有时，希望当满足某条件时对一组语句进行运行，而当条件不满足时则运行另一组语句。

条件解释命令最常见的形式为：

```
#ifdef 标识符
程序段 1
#else
程序段 2
#endif
```

它的作用是：当标识符已经被定义过(用#define 命令定义)，则对程序段 1 进行编译，否则编译程序段 2。

其中#else 部分也可以没有，即：

```
#ifdef
程序段 1
#endif
```

这里的“程序段”可以是语句组，也可以是命令行。这种条件编译可以提高脚本程序的通用性。如果一个脚本程序在不同计算机系统上运行，而不同的计算机又有一定的差异。例如，在调试程序时，常常希望输出一些所需的信息，而在调试完成后不再输出这些信息。可以在源程序中插入以下的条件解释段：

```
#ifdef DEBUG
message ("device_open(%s)\n", file);
#endif
```

如果在它的前面有以下命令行：

**#define DEBUG**

则在程序运行时输出 file 的值，以便调试分析。调试完成后只需将这个 define 命令行删除即可。有人可能觉得不用条件编译也可达此目的，即在调试时加一批 message 语句，调试后一一将 message 语句删除去。的确，这是可以的。但是，当调试时加的 message 语句比较多时，修改的工作量是很大的。用条件解释，则不必一一删改 message 语句，只需删除前面的一条“#define DEBUG”命令即可，这时所有的用 DEBUG 作标识符的条件编译段都使其中的 message 语句不起作用，即起统一控制的作用，如同一个“开关”一样。

有时也采用下面的形式：

**#ifndef 标识符**

程序段 1

**#else**

程序段 2

**#endif**

只是第一行与第一种形式不同：将“ifdef”改为“ifndef”。它的作用是：若标识符未被定义则编译程序段 1，否则编译程序段 2。这种形式与第一种形式的作用相反。

以上两种形式用法差不多，根据需要任选一种，视方便而定。

有人会问：不用条件解释命令而直接用 if 语句也能达到要求，用条件解释命令有什么好处呢？的确，此问题完全可以不用条件解释处理，但那样做目标程序长（因为所有语句都解释），而采用条件解释，可以减少被解释的语句，从而减少实际运行代码的长度。当条件解释段比较多时，实际运行程序长度可以大大减少。

### 3) # ifdef 在脚本开发中的用途

Walk 用户分布于全国各地，各用户需要的基本功能都是一样的，但在某些功能上要随着需求变化，不断加以升级，要想实现全国各地用户的升级工作是很困难的，若利用 E-mail 发送补丁程序给用户，这些补丁程序都是在一套软件的基础上不断地修改与扩充而编写的，并由不同的标志文件转入到不同的模块，虽然程序体积在不断扩大，但丝毫不影响老用户的功能，这主要是得益于脚本程序的#ifdef/#else/#endif 的作用。

我们主要使用以下几种方法,假设我们已在程序首部定义 # ifdef DEBUG 与 # ifdef TEST:

1.利用#ifdef/#endif 将某程序功能模块包括进去，以向某用户提供该功能。

在程序首部定义#ifdef HNLD:

**#ifdef HNLD**

include"n166\_hn.wsp"

**#endif**

如果不许向别的用户提供该功能，则在解释之前将首部的 HNLD 加一下划线即可。

2.在每一个子程序前加上标记，以便追踪程序的运行。

**#ifdef DEBUG**

message(" Now is in hunan !");

**#endif**

头文件的中的#ifndef, 这是一个很关键的东西。比如你有两个脚本文件，这两个脚本文件都 include 了同一个头文件。而解释时，这两个脚本文件要一同解释，于是问题来了，大量的声明冲突。

还是把头文件的内容都放在`#ifndef`和`#endif`中吧。不管你的头文件会不会被多个文件引用，你都要加上这个。一般格式是这样的：

```
#ifndef <标识>
```

```
#define <标识>
```

```
.....
```

```
.....
```

```
#endif
```

<标识>在理论上来说可以是自由命名的，但每个头文件的这个“标识”都应该是唯一的。标识的命名规则一般是头文件名全大写，前后加下划线，并把文件名中的“.”也变成下划线，如：`wcViewConstant.h`

```
#ifndef __WCVIEWCONSTANT_H_
```

```
#define __WCVIEWCONSTANT_H_
```

```
.....
```

```
#endif //__WCVIEWCONSTANT_H_
```

## 4) #define 用法

### 1.简单的 define 定义

```
#define MAXTIME 1000
```

一个简单的 `MAXTIME` 就定义好了，它代表 1000，如果在程序里面写

```
if(i<MAXTIME){.....}
```

编译器在处理这个代码之前会对 `MAXTIME` 进行处理替换为 1000。

这样的定义看起来类似于普通的常量定义 `CONST`，但也有着不同，因为 `define` 的定义更像是简单的文本替换，而不是作为一个量来使用，这个问题在下面反映的尤为突出。

### 2.define 的“函数定义”

`define` 可以像函数那样接受一些参数，如下

```
#define max(x,y) (x)>(y)?(x):(y);
```

这个定义就将返回两个数中较大的那个，看到了吗？因为这个“函数”没有类型检查，就好像一个函数模板似的，当然，它绝对没有模板那么安全就是了。可以作为一个简单的模板来使用而已。

但是这样做的话存在隐患，例子如下：

```
#define Add(a,b) a+b;
```

在一般使用的时候是没有问题的，但是如果遇到如：`c * Add(a,b) * d` 的时候就会出现问題，代数式的本意是 `a+b` 然后去和 `c`，`d` 相乘，但是因为使用了 `define`（它只是一个简单的替换），所以式子实际上变成了

```
c*a + b*d
```

因此我们在定义的时候，养成一个良好的习惯，建议所有的层次都要加括号。

### 3.define 的多行定义

`define` 可以替代多行的代码，例如下面的宏定义（非常的经典，虽然让人看了恶心）

```
#define MACRO(arg1, arg2) while (1){ \
```

```
/* declarations */\
```

```
stmt1;\
```

```
stmt2;\
break; /* ... */\
} /* (no trailing ; ) */
```

关键是要在每一个换行的时候加上一个“\”

#### 4.如何定义宏、取消宏

定义宏

```
#define [MacroName] [MacroValue]
```

取消宏

```
#undef [MacroName]
```

#### 5.条件解释

```
#ifdef XXX...(#else) ...#endif
```

例如

```
#ifdef DV22_AUX_INPUT
```

```
#define AUX_MODE 3
```

```
#else
```

```
#define AUY_MODE 3
```

```
#endif
```

```
#ifndef XXX ... (#else) ... #endif
```

#### 6.头文件(.h)可以被头文件或脚本文件包含；

重复包含（重复定义）

由于头文件包含可以嵌套，那么脚本文件就有可能包含多次同一个头文件，就可能出现重复定义的问题。

通过条件编译开关来避免重复包含（重复定义）

例如

```
#ifndef __headerfileXXX__
```

```
# define __headerfileXXX__
```

```
...
```

文件内容

```
...
```

```
#endif
```

## 第三章 标准函数类

WalkLan 与 c/c++语言类似，包含三种基本程序体：顺序执行体、循环执行体和条件执行体。以函数为基本编程单元，程序从 main 函数开始执行，其中可调用系统内置函数和用户自定义函数。

### 1、main 函数

程序从 main()函数开始执行。

如果程序中没有定义任何函数，则认为所有代码在 main()函数内。也就是说这时解释器自动添加开头和结尾。

不允许用户定义了一些函数，但是其中没有 main()函数。

main 函数有两种类型。一种是无参数，另一种为带参数。

无参数形式，如：

```
void main()
{
    string s="hellow!";
    message(s);
}
```

main()函数可以带一个 string 数组参数，形式为：

```
void main(array args)
{
    for (int i=0; i<args.getSize(); i++)
        message(args[i]);
}
```

该形式的 main 函数一般发生在被另一个脚本按 wkView 的 WalkLan 成员调用时，参见后面的例子。

在脚本中可以包含另一个脚本，或执行另一个脚本。

脚本文件应保持在<Walk 执行程序目录>\WalkLan\目录下，这样在包含或执行一个脚本时只要书写脚本文件名就可以了，不需要给出全路径。（参见 wkView 的 getVailFilePath 函数说明）如：

```
#include "wcGeomConstant.h"
wkView view;
...
array args;
args.add("first arg"); args.add("second arg");
view.WalkLan("myInclude.wsp", args);
```

其中 wcGeomConstant.h 一般存放在 <Walk 执行程序目录>\WalkLan\include\目录下。

而 myInclude.wsp 则可存在 WalkLan 目录下，或其子目录下。

## 2、用户定义函数

1) 定义方法:

返回类型 函数名(参数列表)

```
{
    函数体
}
```

2) 函数的参数都是值参或引用。

3) 函数不支持重载。

4) 函数支持递归。

【example】:

```
double distance(wkPoint a, wkPoint b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z));
}

void main()
{
    wkPoint p1, p2;
    p1.x=1; p1.y=1; p2.x=100; p2.y=100;
    message(toString(distance(p1,p2)));
}
```

## 3、系统标准函数

强制终止脚本运行

```
void exit ( );
```

### 1) 输入输出及控制台函数

控制台 console

```
int consoleOpen(int act=0);
```

参数 act 可取值:

0-- 打开 console (控制台)

1-- 等待任意键 (返回 key\_value)

2-- 清屏

4-- 根据当前颜色刷新屏幕

5-- 获取字符颜色

6-- 获取背景颜色

7-- 获取输入环境的 CodePage

9-- 获取输出环境的 CodePage

```
int consoleOpen(int act, int codePageID);
```

8-- 设置输入环境的 CodePage

10-- 设置输出环境的 CodePage

参数 `CodePage` 的作用，是决定输入输出环境以何种编码方式显示动态内容。`CodePage`：可读/可写。整型。代码页是字符集的数字值，不同的语言使用不同的代码页。例如，ANSI 代码页为 1252，日文代码页为 932，简体中文代码页为 936。

```
int consoleOpen(int act, int textColor, int backColor);
```

3-- 设置当前颜色

参数 `textColor`: 字符颜色

参数 `backColor`: 背景颜色

颜色：1--blue,2--green,4--red,8--增强, 15--white ,0--black。

关闭控制台

```
void consoleClose(bool bKeyHit);
```

参数 `bKeyHit`: true-- press any key continue

脚本结束时系统会自动执行 `consoleClose(true)`

输入函数

从控制台读入整数

```
Int readInteger();
```

从控制台读入字符串

```
string readString();
```

从控制台读入实数

```
double readDouble();
```

输出函数

格式化输出到 `console`，输出格式同 `c` 的 `printf`

```
void printf(string fmt, ...);
```

非格式化输出到 `console`

```
void print(...);
```

输出并换行。同 `print`

```
void println(...);
```

弹出系统消息对话框并显示 `msg`

```
void message( string msg );
```

格式化输出到脚本编辑器输出栏，输出格式同 `c` 的 `printf`

```
void trace ( ... );
```

获取数据类型

```
int getDataType( wkMultiType d );
```

参数 `wkMultiType` 指可以包含 `int`、`bool`、`double`、`LPCTSTR`、`wkPoint`、`wkDb`、`wkLayer`、`wkFeature`、`wkAnnotation`、`wkStyle`、`wkGeometry`、`wkPolygon`、`wkParts`、`wkPoints`、`wkBox`;

【example】:

```

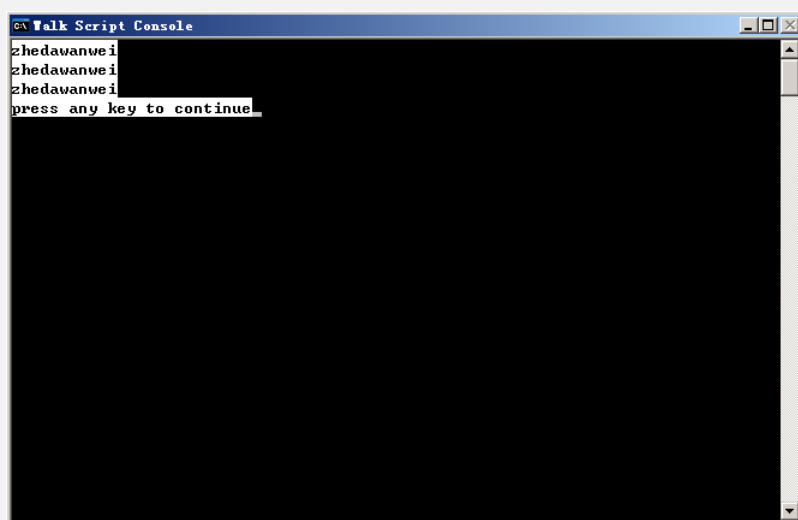
void main()
{
    string str;
    consoleOpen(3,0,15);//设置控制台的字体和背景;
    str=readString();//向控制台读入字符串;
    message(str);//在消息对话框中输出字符串;
    printf("%s\n",str);//格式化输出到控制台;
    print(str);//非格式化输出到控制台;
    consoleClose(true);//关闭控制台;
}

```

结果:



message(str);



```

printf("%s\n",str);
print(str);

```

## 2) 数学函数

计算弧度 x 的正弦值

```
double sin( double x );
```

计算弧度 x 的余弦值

```
double cos( double x );
```

计算弧度 x 的正切值

```
double tan( double x );
```

计算实数  $x$  的反正弦值

`double asin( double x );`

计算实数  $x$  的反余弦值

`double acos( double x );`

计算实数  $x$  的反正切值

`double atan( double x );`

计算点  $(x,y)$  的角度

`double atan2( double y, double x );`

计算  $x$  的开方

`double sqrt( double x );`

计算  $x$  的  $y$  次方

`double pow( double x, double y )`

取绝对值

`int abs( int );`

取以  $e$  为底数、 $x$  为指数的幂函数值

`double exp(double x);`

取  $x$  的自然对数值

`double log(double x);`

取以 10 为底数、 $x$  为指数的对数值

`double log10(double x);`

返回大于或者等于指定表达式的最小整数

`double ceil(double x);`

取不大于  $x$  的最大整数

`double floor(double x);`

取绝对值

`double abs( double );`

产生随机数

`int random( );`

设置随机种子

`void randomize( );`

### 3) 转化函数

整数转为字符

```
string chr( int );
```

字符转为整数

```
int ord( string );
```

把字符串转换成 int，与 atoi 相同

```
int parseInt( string s );
```

```
int atoi ( string s );
```

把字符串转换成 double，与 atof 相同

```
double parseDouble( string s );
```

```
double atof ( string s );
```

把 value 转换成 string，类似于 Java 中的 value.toString()

```
string toString( wkMultiType value );
```

### 4) 时间函数

时钟数

```
int clock();
```

这个函数返回从“开启这个程序进程”到“程序中调用 clock()函数”时之间的 CPU 时钟计时单元（clock tick）数，在 MSDN 中称之为挂钟时间（wal-clock）；若挂钟时间不可取，则返回-1。

```
int clock(int &clocksPerSec);
```

参数 clocksPerSec：每秒时钟数

clocksPerSec 是个常量，它用来表示一秒钟会有多少个时钟计时单元，其定义如下：

```
#define clocksPerSec((clock_t)1000)
```

可以看到每过千分之一秒（1 毫秒），调用 clock（）函数返回的值就加 1。

当前机器时间

```
double time();
```

按年月日和时分秒生成时间

```
double time(int year, int mon, int mday, int hour, int min, int sec);
```

定义	说明	取值范围
Year	Year	100-9999
Mon	Month	0-12
Mday	Day	0-31
Hour	Hour	0-23
Min	Minute	0-59
Sec	Second	0-59
Wday	Day of week	0-6 Sunday = 1
Yday	Day of year	0-365 January= 1

详细时间

```
void timeTm(double tm, int& year, int& mon, int& mday, int& hour, int& min, int& sec);
```

取时间的年日和周日

```
void timeTm(double tm, int& yday, int& wday);
```

时间字符串

```
void timeTm(double tm, string& ascTime);
```

获取当前电脑机器时间转换为时间字符串并赋给 ascTime;

参数 tm: 当前电脑机器时间

时间差

```
void timeSpan(double tm1, double tm2, int& day, int& hour, int& min, int& sec);
```

得到两个时刻电脑机器时间 tm1、tm2 的时间差，并赋给 day、hour、min、sec。

【example】:

```
void main()
{
    int clocksPerSec,clocknumber,tm1,tm2;
    int year,mon,day,mday,hour,min,sec,yday,wday;
    string strtm;
    trace("%d\n",clocksPerSec);//没有调用前系统默认值为 0（整形）
    //从程序开始到执行这个 clock（）函数时所用的时钟数
    clocknumber=clock(clocksPerSec);
    trace("%d\n",clocknumber);
    trace("%d\n",clocksPerSec);//调用函数后 clocksPerSec 的值发生改变
    tm1=time(year,mon,mday,hour,min,sec);//按年月日和时分秒生成时间
    trace("%d\n",tm1);
    timeTm(time(),year,mon,mday,hour,min,sec);//详细时间
    trace("%d,%d,%d,%d,%d,%d\n",year,mon,mday,hour,min,sec);
    timeTm(time(),yday,wday);//取时间的年日和周日
    trace("%s\n",strtm);//调用函数前的 strtm 的值
    timeTm(time(),strtm);
    trace("%s\n",strtm);//调用函数后的 strtm 的值
    tm2=time();
    timeSpan(tm2,time(),day,hour,min,sec);//时间差
```

```

    trace("%d,%d,%d,%d\n",day,hour,min,sec);
}
    结果:
0
15237840
1000
36494
2011,6,20,15,3,1
15:03:01 Monday, June 20, 2011
0,15,3,1

```

## 4、string：字符串

注意：两个字符串常量不得相加！如：string s="s1"+"s2";

字符串赋值（string 的用法同 mfc 的 CString，以及 stl 中的 string）

string& operator =( string lpsz );

置取字符串指定位置的字符

void setAt( int index, string chr );  
string getAt(int index);

在 index 处插入字符串 str

void insertAt(int index, string str);  
从 index 处开始删除 count 个字符  
void removeAt(int index, int count);

从字符串的指定位置开始取指定长度的子串

string mid( int nFirst, int nCount );  
从字符串的指定位置开始子串一直到字符串末尾  
string mid( int nFirst );

从左开始取字符串中指定长度的子串

string left( int nCount );

取字符串指定长度的尾串

string right( int nCount );

字符串是否为空

bool isEmpty ( );

获取字符串长度（字符个数）

int getLength();

对字符串格式化赋值（格式规则参见 c 语言的 printf 函数）

void format( string lpszFormat, ... );

例如：s.format("输出的值是%d ", d1); d1 是一个整型变量

将字符串字符全部转为大写

```
void makeUpper();
```

将字符串字符全部转为小写

```
void makeLower();
```

将字符串反转

```
void makeReverse();
```

与另一个字符串 `str` 比较，相等时返回 0

```
int compare( string str);
```

与另一个字符串 `str` 比较，比较中忽略大小写，相等时返回 0

```
int compareNoCase( string str );
```

在字符串中查找一个子串 `str`，若未找到则返回-1

```
int find( string str);
```

从字符串尾部向前查找一个子串 `str`，若未找到则返回-1

```
int reverseFind( string str);
```

将字符串中子串 `str1` 替换为另一个子串 `str2`

```
void replace( string str1, string str2 );
```

剔除字符串左边的白字符，或指定的字符 `str`

```
void trimLeft();
```

```
void trimLeft( string str);
```

剔除字符串右边的白字符，或指定的字符

```
void trimRight();
```

```
void trimRight( string );
```

从指定位置 `iStart` 开始截取以指定分隔符 `tokens`,分隔的左串

```
string tokenize( string tokens, int &iStart );
```

**【example】:**

```
void main()
{
    string str="%First Second#Third";
    int curPos = 0;
    for (string res= str.tokenize("% #",curPos);
        !res.isEmpty();
        res= str.tokenize("% #", curPos))
        message("Resulting token: "+ res);
}
```

上例的输出结果如下：

Resulting Token: First  
Resulting Token: Second  
Resulting Token: Third

## 5、array：数组

返回数组中元素的个数

`int getSize();`

设置数组大小

`void setSize( int );`

数组完全拷贝

`array& operator =( array& );`

取特定位置的元素

`wkMultiType getAt( int );`

数组清空

`void removeAll( );`

删除特定位置上的元素

`void removeAt( int );`

在特定位置插入给定元素

`void insertAt( int, wkMultiType );`

在最后插入给定元素

`void add( wkMultiType );`

设置特定位置的元素

`void setAt( int, wkMultiType );`

把 b 的所有元素加在数组的后面

`void append( array& b );`

例如：a.append(b); 把 b 的所有元素加在数组 a 的后面

对数组元素排序(true-升序, 或 false-降序)

`bool sort( bool bAscend=true );`

排序约束：

数组元素的数据类型必须一致（单一数据类型）

数组元素的数据类型只能是整型、浮点和字符串

若不满足约束，则 sort 返回 false

矩阵相加。

`array mtAdd(array b);`

结果矩阵为 array 所有元素为 double 类型，若为二维数组，则存储和表达为 A[i][j]。

输入数组的元素可以是 int 类型或 double 类型

下同

矩阵相减

`array mtSubtract(array b);`

矩阵相乘

`array mtMultiply(array b);`

矩阵转置

`array mtTranspose(array b);`

高斯消元： $AX+L=0$

为保证求解精度，高斯消元采用的是主元素消元法

若要求 n 满秩矩阵 N 的逆阵  $N(-1)$ ，可组成  $N*N(-1)-E=0$ ，E 为单位矩阵

使用 n 次高斯消元，得到逆阵

`int mtGauss(array A, array L);`

```

【example】:
void traceMT(array& mt, int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            if (j == 0)
                trace ("\t\t");
            trace ("%0f\t", r==1 ? mt[j] : mt[i][j]);
            if (j == c-1)
                trace ("\t ");
        }
        trace ("\n");
    }
    trace("\n");
}

int main()
{
    array a = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {3, 6, 9}};
    array b = {{3, 2, 1}, {6, 5, 4}, {9, 8, 7}, {6, 1, 7}};

    array c = a.mtAdd(b);
    trace("a + b = \n");
    traceMT(c, 4, 3);

    array d = a.mtSubtract(b);
    trace("a - b = \n");
    traceMT(d, 4, 3);

    array e = a.mtMultiply(b.mtTranspose());
    trace("a * transpose(b) = \n");
    traceMT(e, 4, 4);

    array p1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 10}};
    array x1;
    int o1 = x1.mtGauss(p1, {3, 9, 10});
    trace("p1 = \n");
    traceMT(p1, 3, 3);
    if (o1 == 1)
    {
        trace("唯一解    x1 = \n");
        TraceMT(x1, 1, 3);
    }
    else if (o1 == -1)
        trace("无解\n");
    else if (o1 == 0)
        trace("无数组解\n");
}

    结果:
a + b =
|   4   4   4   |
|  10  10  10   |
|  16  16  16   |
|   9   7  16   |

```

```

a - b =
|  -2  0  2  |
|  -2  0  2  |
|  -2  0  2  |
|  -3  5  2  |

a * transpose(b) =
|  10  28  46  29  |
|  28  73  118  71  |
|  46  118  190  113  |
|  30  84  138  87  |

p1 =
|  87  87  87  |
|  87  87  87  |
|  87  87  87  |

唯一解    x1 =
|  -4  11  -5  |

```

## 6、wkPoint：三维坐标点

wkPoint 有 3 个数据成员：

double x, y, z;

Walk 系统平台处理的空间对象为点、线、面和文字影像要素。面由边界包围而成，边界和线由序列点构成，因此 WalkLan 脚本语言中将实数三维点坐标作为一种基本数据类型，以彰显 WalkLan 的 GIS 专业特征。

**【example】:**

```

void main()
{
    array arr1,arr2;
    wkPoint pt;
    pt.x=100.0; pt.y=200.0; pt.z=0;
    arr1.add("12");//向数组 arr1 中添加元素"12";
    arr1.add("13");//向数组 arr1 中添加元素"13";
    arr1.add(pt);//向数组 arr1 中添加元素 pt;
    arr2.append(arr1);//将数组 arr1 中的元素全部加到 arr2 后面;
    for(int i=0;i<arr2.getSize()-1;i++)
        trace("%s\n",toString(arr2[i]));//输出数组 arr2 中的元素;
    trace("{ %d,%d,%d }",arr2[2].x,arr2[2].y,arr2[2].z);
}

结果:
12
13
{0,1079574528,0}

```

## 第四章 实体类对象

对象类成员函数的参数值可以用系统预定义的宏（参见附录 1：常量定义），使得脚本看起来更加易读。若使用预定义宏，则要包含相应的头文件。如：

```
#include "wcGeomConstant.h"
...
if (wcGeom9iContain==zhejiangGeom.get9i(hangzhouGeom))
...
```

### 1、wkView：图形主窗口

获取图形窗口类对象句柄，用于与 **Ws** 函数的接口。

```
int handle();
```

#### 1) 交互命令

执行 windows 命令

```
int shell( string command );
```

类似于 WinExec，执行外部程序。

参数 command 可以是任何 shell 命令,例如:calc, cmd, cleanmgr, odbcad32, logoff, regedit, sysdm.cpl,taskmgr, write, telnet, services.msc, osk, fonts, access.cpl 等等。

```
int shell(string operation, string file, string parameters = "", int nShowCmd =
SW_SHOWNORMAL);
```

参数 operation 可以是" open "," explore "," print "等等；

参数 file 是一个文档文件或者是文件夹路径；

参数 nShowCmd 可以是：

```
SW_HIDE 0,
SW_SHOW+NORMAL 1,
SW_SHOWMINIMIZED 2,
SW_MAXIMIZE SW_SHOWMAXIMIZED 3,
SW_SHOWNOACTIVATE 4,
SW_SHOW 5,
SW_MINIMIZE 6,
SW_SHOWMINNOACTIVE 7,
SW_SHOWNA 8,
SW_RESTORE 9,
SW_SHOWDEFAULT 10
```

根据指定文件的扩展名，查找注册表运行程序，执行该命令

```
int shell(string commandLine, bool waitThisProcess, int nShowCmd =
SW_SHOWNORMAL);
```

参数 commandLine 可以是"www.zju.edu.cn"，"C:\test.doc"等

参数 waitThisProcess 是 true 时，它将会等待进程（例如 "c:\myTest.exe"）直到它被执

行了。

```

【example】:
void main()
{
    wkView v;
    int ret;
    ret = v.shell("osk");
    if (ret > 32) message("osk"); //屏幕键盘
    ret = v.shell("calc");
    if (ret > 32) message("calc");//计算器
    ret = v.shell("cmd /C tree c: >>c:\aa.txt");
    if (ret > 32) message("list c: to aa.txt");//把 c 盘的目录存到 aa.txt
    ret = v.shell("cmd /C copy c:\\aa.txt c:\\bb.doc");
    if (ret > 32) message("copy aa.txt to bb.doc");//把 aa.txt 的内容复制到 bb.doc 中
    ret = v.shell("explorer /select, c:\\aa.txt");
    if (ret > 32) message("select c:\\aa.txt");//寻找 c 盘中的 aa.txt 文件，并选中
    ret = v.shell("open", "c:\\aa.txt");
    if (ret > 32) message("open aa.txt");//打开 aa.txt 文件
    ret = v.shell("explorer", "c:\\aa.txt");
    if (ret > 32) message("explorer,aa.txt");//打开 aa.txt 文件
    ret = v.shell("print", "c:\\aa.txt");
    if (ret > 32) message("print aa.txt");//打印 aa.txt 文件
    ret = v.shell("new", "www.google.com");
    if (ret > 32) message("new google.com");//刷新 google.com
    ret = v.shell("open", "http://www.zju.edu.cn");
    if (ret > 32) message("open zju.edu.cn");//打开浙江大学主页
    ret = v.shell("c:\\bb.doc", false);
    //打开 bb.doc,但是它将不会等待进程 ("c:\\bb.doc")
    if (ret > 32) message("not wait bb.doc");
    ret = v.shell("c:\\aa.txt", true);
    //打开 aa.txt,它将会等待进程 ("c:\\aa.txt") 直到它被执行了。
    if (ret > 32) message("wait aa.txt");
}

```

详细的 command 命令见附录 1

执行一个 WalkLan 脚本

执行一个脚本，这个脚本不需要外部输入参数，参数 wspPath 是脚本文件的路径

**bool walkScript** ( **string** wspPath );

脚本需要外部输入数组型参数（这里的数组必须为字符串型的数组）

**bool walkScript** ( **string** wspPath, **array** params );

脚本需要外部输入字符串型参数

**bool walkScript** (**string** wspPath, **string** args);

## 2) 交互对话框

单行输入对话框

```
string dlgInput( string title, string defText );
```

参数 title 是对话框的标题

参数 defText 是对话框默认的输入内容

寻求确认对话框

```
bool dlgQuestion( string question );
```

参数 question 是指所要询问的问题

文件选择对话框

```
string dlgFile( int bOpen, string defaultExt, string filter );
```

参数 bOpen : true--打开文件, false--保存文件

参数 defaultExt : 默认的文件扩展名

参数 filter : 为过滤条件, 如: "WalkISurvey 数据库 \*.mdb|\*.mdb|所有文件 \*.\*|\*.\*|"

目录选择对话框, 并且定位到 defFolder 所在位置

```
string dlgFolder( string defFolder );
```

参数 defFolder 是默认的文件夹

【example】:

```
void main()
```

```
{
```

```
    wkView view;
```

```
    view.dlgInput("浙大万维是在杭州吗? ", "是"); //单行输入对话框;
```

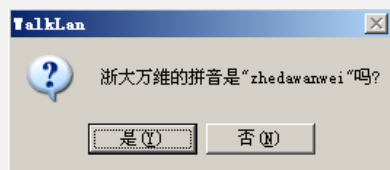
```
    view.dlgQuestion("浙大万维的拼音是\"zhedawanwei\"吗?"); //寻求确认对话框;
```

```
    view.dlgFile( true, "mdb", "*.mdb|*.mdb|所有文件 *.*|*.*|"); //文件选择对话框;
```

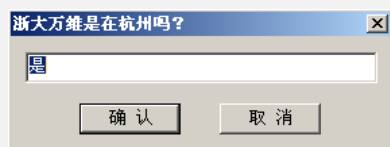
```
    view.dlgFolder("F:\\浙大万维"); //目录选择对话框;
```

```
}
```

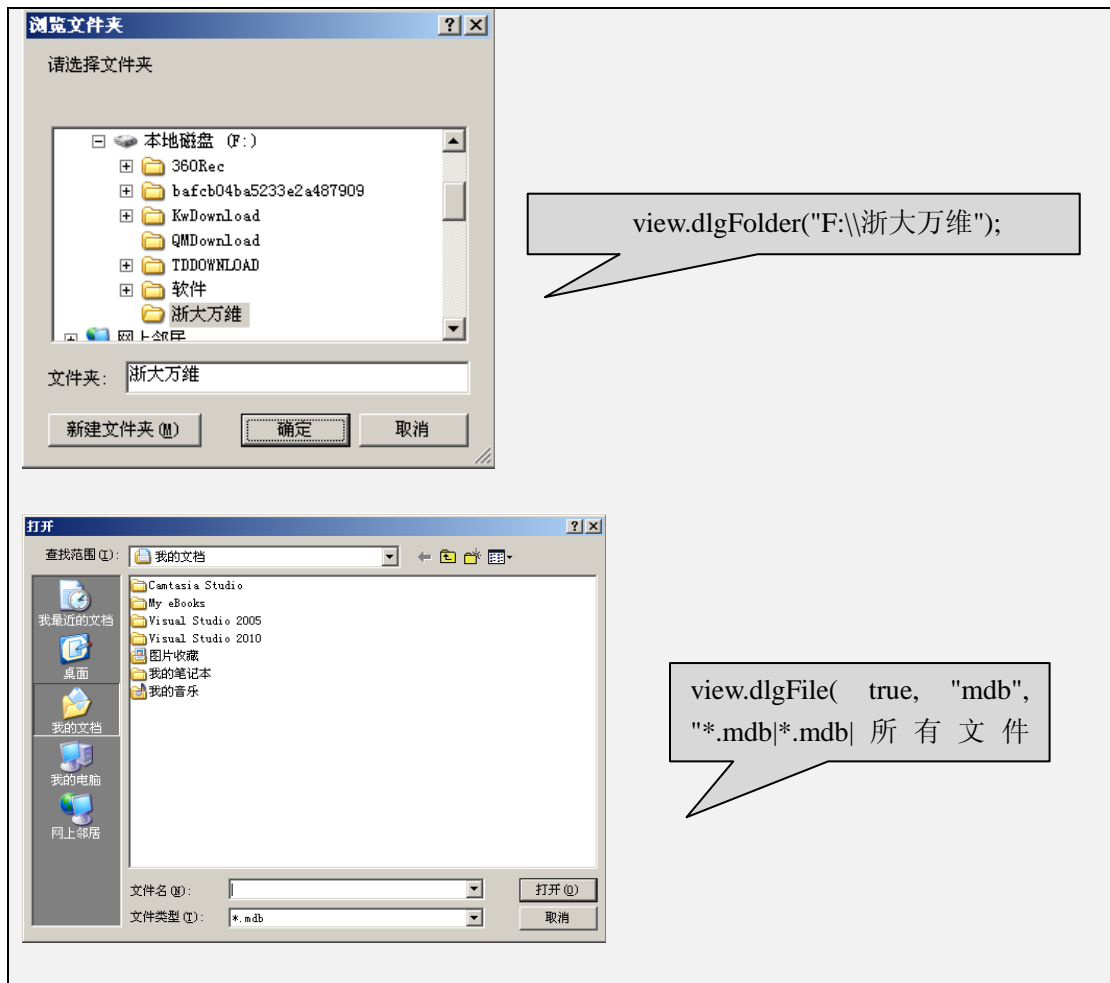
结果:



```
view.dlgQuestion(" 浙大万维的拼音是  
\"zhedawanwei\"吗?");
```



```
view.dlgInput("浙大万维是在杭州吗? ", "  
是");
```



### 3) 进程提示

开始进程提示

```
void waitStart( string msg );
```

进程中提示

```
void waitMessage( string msg, int i );
```

结束进程提示

```
void waitEnd( );
```

### 4) 获取全路径文件名

获取文件全路径并判断文件存在，若成功则返回全路径，否则为空

```
string getVailFilePath(string file);
```

若文件名中含全路径（含'.'），则按全路径查找。否则

若无相对路径，则在 <执行程序目录>\WalkLan\ 目录下查找

若含相对路径，则在 <执行程序目录>\WalkLan\<相对路径>\ 目录下查找

允许相对路径中前缀 "..\"

若非全路径文件名，且文件名中不含扩展名时，系统将按.wsp 和.cpp 后缀扩展名进行查找。

如：

当执行文件目录为 d:\Newwalk\Bin（脚本的默认目录为 d:\Newwalk\Bin\WalkLan）时

1、当指定文件全路径：“c:\myTool\myScript.wsp”，则系统将判断在 c:\myTool\目录下是否有 myScript.wsp 文件存在。

2、若指定文件为 “myScript.wsp”，则系统将判断在 d:\Newwalk\Bin\WalkLan\ 目录下以及其子目录下是否存在 myScript.wsp 文件。

3、若指定文件为 “myScript”，则系统将判断在 d:\Newwalk\Bin\WalkLan\ 目录下以及其子目录下是否存在 myScript.wsp 或 myScript.cpp 文件。

4、若指定文件为 “..\WalkIni\myScript.wsp”，则系统将判断在 d:\Newwalk\Bin\WalkIni\目录下以及其子目录下是否存在 myScript.wsp 文件。

5、若指定文件为 “WalkIni\myScript.wsp”，则系统将判断在 d:\Newwalk\Bin\WalkLan\WalkIni\ 目录下以及其子目录下是否存在 myScript.wsp 文件。

#### 【example】:

```
void main()
{
    wkView view;
    string fl="地物跳转.wst";
    string s = view.getVailFilePath(fl);
    s.format("%s",s.isEmpty()? fl+" 文件未找到":s);
    trace("%s\n",s);
}
```

结果：

E:\ walkLan\WalkScript\template\地物跳转.wst

```
void main()
{
    wkView view;
    string fl=" getFiles.cpp";
    string s = view.getVailFilePath(fl);
    s.format("%s",s.isEmpty()? s+" 文件未找到":s);
    trace("%s\n",s);
}
```

结果：

E:\ walkLan\WalkScript\lib\getFiles.cpp

## 2、wkGeoset：工作空间

获取工作窗口类对象句柄，用于与 Ws 函数的接口。

```
int handle( );
```

## 1) 层管理

获取工作空间中的层数

```
int getLayerCount();
```

按下标获取工作空间中的层对象(下标是工作空间中的层的排列顺序, 从 0 开始)

```
wkLayer getLayerAt( int at );
```

按层名和数据库获取工作空间中的层对象

```
wkLayer getLayer( string layer );
```

```
wkLayer getLayer( string layer, wkDb db );
```

若未指定 wkDb 则从工作空间的第一个 Db 开始查找, 返回层名匹配的层

获取工作空间中的当前可编层

```
wkLayer getEditableLayer();
```

将层从工作空间中移去

```
void removeLayer( wkLayer layer );
```

从库中加入一层到工作空间中

```
wkLayer addLayer( string layer, int dbAt );
```

```
wkLayer addLayer( string layer, wkDb db );
```

## 3、wkDb：数据库

获取数据库类对象句柄, 若为 0, 表示对象不可用。

```
int handle();
```

### 1) 数据库操作

获取数据库名称

```
string getName();
```

取得数据库类型: 0--access 库, 1--dsn 数据库

```
int getType();
```

打开数据库

```
bool open( string dbName, bool bMDB, bool addToGeoset );
```

参数 dbName: 数据库名, 可以是 ODBC 名

参数 bMDB: true--要打开的库是 MDB 库. false--是 sql-server 库

参数 addToGeoset: true--将打开的数据库添加到 geoset, 要求该库必须是 walk 库

参数 false--不添加到 geoset 中, 数据库使用完后必须 close ( ) .

关闭数据库, 且从工作空间中移去该库的所有层

```
void close();
```

判断层是否在库中

```
bool isLayerExisted( string layerName );
```

在库中创建一个简单层( 不加入到 geoset 中 )

```
bool createLayer( string layerName );
```

新层包含 FeatureId, StyleId, Geometry, CreateTime 四个字段

若要在层中添加字段，可使用 execSQL

在库中创建一个与 fromLayer 同结构的普通层

```
bool createLayer(string layerName, string fromLayer, bool bCopyData);
```

参数 bCopyData: true--从 fromLayer 复制数据到新层中

【example】:

```
void main()
```

```
{
```

```
    wkGeoset geoset;
```

```
    wkDb db;
```

```
    db.open("e:\\hff0003.mdb", true, true);
```

```
    wkLayer layer1 = geoset.addLayer("实测点层", db);
```

```
    wkLayer layer2= geoset.addLayer("建筑物", db);
```

```
    wkLayer layer3 = geoset.addLayer("一个新层", db);
```

```
    for(int i=0;i< geoset.getLayerCount();i++)
```

```
        trace("%s\n",geoset.getLayerAt(i).getName());
```

```
    layer1.setEditable(true);//将工作空间中第一个层设置为可编状态;
```

```
    wkPoint pt1;
```

```
    wkFeature fea;
```

```
    pt1.x=100;pt1.y=100;pt1.z=100;
```

```
    fea=layer1.addFeature(pt1);//向层中添加地物：单点;
```

```
    geoset.clearAllTarget();//清除所有层的地物;
```

```
    geoset.removeLayer( layer1 );//将层从工作空间中移去
```

```
    db.close();
```

```
}
```

结果:

实测点层

建筑物

一个新层

## 2) SQL

执行普通 SQL 语句

**bool** **execSQL**( **string** sql, **bool** warning );

执行 WalkSQL 语句( 参见《WalkSql 使用手册》 )

从 select 语句获取数据( **string** 数组 )

**array** **getDataBySQL**( **string** sql, **int** topLimit );

表是否在库中

**bool** **isTableExisted**( **string** tableName );

获取库中的所有表名称

**array** **getTables**();

获取库中的所有层名称

**array** **getLayers**( );

获取库中的所有视图名称

**array** **getViews**();

计算机数据库中的视图是一个虚拟表，其内容由查询定义。

获取一张表的所有字段名称或主键字段名

**array** **getTableFields**( **string** tableName, **bool** bPrimaryOnly=false );

获取自动编号定义符(在 access 中为 AutoIncrement, 在 sql-server 中为 INT IDENTITY)

**string** **getAutonumberDef**();

获取表中自动编号字段名

**string** **getAotunumberField**( **string** tableName );

自动编号是字段的一种类型

获取表中一个字段的 SQL 类型 (参见 wcLayerConstant.h 中的 wcSQL)

**int** **getFieldSQLType**( **string** tableName, **string** fieldName );

获取表中一个字段的长度

**int** **getFieldPrecision**( **string** tableName, **string** fieldName );

获取表中一个字段的小数位

**int** **getFieldScale**( **string** tableName, **string** fieldName );

获取表中一个字段是否允许为空

**bool** **getFieldNullability**( **string** tableName, **string** fieldName );

### 3) 为 sql 语句解析的工具函数

从 src 中查找字 key( 不处理单引号对内的字符部分 )。不区分大小写，返回位置

```
int sqlFind( string src, string key, int nFnd );
```

分解 str 以分隔符 delimiter 分隔( 不处理单引号对内的字符部分 )的字符串组

```
array sqlSubsByDeli( string str, string delimiter );
```

## 4、wkLayer：图层

Walk 中的层由至少三张库表构成：

地物表——<层名>Features

文字表——<层名>Annotations

式样表——<层名>Symbols

如，层名为：myLayer，则三张库表分别为：myLayerFeatures, myLayerAnnotations, myLayerSymbols。

在地物表中，至少有 4 个 Walk 系统保留字段：

FeatureId——整型，主键

StyleId——整型，地物的式样 ID，与式样表关联

Geometry——二进制，地物几何，与 OGC 的 WKB 相同

CreateTime——地物的创建时间

地物表中允许用户添加自己的字段，如 FeatureName 等。

### 1) 层属性

获取图层类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

获取层名

```
string getName();
```

获取层所属的数据库

```
wkDb getDb();
```

获取层已加载地物和文字的外接盒

```
wkBox getBox();
```

置取层在工作空间中的上下次序(上层压盖下层)

```
int getOrder();
```

```
void setOrder( int order );
```

wcLayerOrderTop( 0 )--在最上层，<0--层不在工作空间中

参数 order: 参见 wcLayerConstant.h 的 wcLayerOrder

获取层的属性字段名称(不含 4 个基本字段名)

```
array getFieldnames( );
```

获取字段的序号

```
int getFieldIndex( string fieldName );
```

用于与 wkFeature::getInfo 和 setInfo 配合使用

加载或卸载层地物属性信息

```
void loadInfo(bool bLoad=true);
```

参数 bLoad:

true—加载层地物属性

false—卸载未修改地物的属性

层中地物和文字是否已增删改

```
bool isDirty( );
```

## 2) 地物

获取层中地物数

```
int getFeatureCount( );
```

按下标获取地物 (at 从 0 开始)

```
wkFeature getFeatureAt( int at );
```

按 featureId 获取地物

```
wkFeature getFeature( int featureId );
```

向层中添加地物

```
wkFeature addFeature( wkFeature fea );
```

```
wkFeature addFeature( wkGeometry geom );
```

```
wkFeature addFeature( wkGeometry geom, int attrFromFeaId );
```

```
wkFeature addFeature( wkPoint pt );
```

```
wkFeature addFeature( wkPoint pt, double angle );
```

```
wkFeature addFeature( wkBox box, bool bRing );
```

```
wkFeature addFeature( array pts, int linetype, bool bRing );
```

```
wkFeature addFeature( wkPolygon poly );
```

```
wkFeature addFeature( wkParts parts, bool bRing );
```

```
wkFeature addFeature( wkPoints points, bool bRing );
```

删除层中的地物

```
void deleteFeature( int featureId );
```

update Feature 在 wkFeature 中

### 3) 文字和影像

获取层中文字和影像个数

**int** **getAnnotationCount**();

按下标获取层中文字和影像（at 从 0 开始）

**wkAnnotation** **getAnnotationAt**( **int** at );

按 annotationId 获取层中文字和影像

**wkAnnotation** **getAnnotation**( **int** annotationId );

向层中添加文字和影像

**wkAnnotation** **addAnnotation**( **wkAnnotation** ann );

**wkAnnotation** **addAnnotation**( **string** text, **wkPoint** pt, **int** charH\_01mm );

参数 charH\_01mm 指 text 的高度，以 0.1mm 为单位

**wkAnnotation** **addAnnotation**( **string** imagePath, **array** pts );

参数 charH\_01mm: 字高;

参数 pts: 用于定位图像（4 个点）;

删除层中的文字和影像

**void** **deleteAnnotation**( **int** annotationId );

update Annotation 在 **wkAnnotation** 中

### 4) 式样

获取层中式样个数

**int** **getStyleCount**();

按下标获取层中式样（at 从 0 开始）

**wkStyle** **getStyleAt**( **int** at );

按 styleId 获取层中式样

**wkStyle** **getStyle**( **int** styleId );

向层中添加式样

**wkStyle** **addStyle**( **wkStyle** style );

**wkStyle** **addStyle**( **int** type, **string** name, **string** note );

参数 **string**: 式样名;

参数 **note**: 式样别名;

删除层中的式样( 若该式样被层中的地物和文字使用，则删除无效 )

**void** **deleteStyle**( **int** styleId );

用它层（可不同库）的式样更新本层的式样

```
bool updateStyle(int tarStyleId, int srcStyleId, wkLayer srcLayer);
```

参数 tarStyleId: 本层要被更新的式样

参数 srcStyleId: 它层式样;

参数 srcLayer: 它层

## 5) 层的类型和其他属性

层类型是否为隐式索引

```
bool isIndexLayer();
```

eg: Layer.isIndexLayer();

获取隐式索引层的矩形范围的相关格子值( 整型 )

```
array getCell( wkBox range );
```

层类型是否为属性索引

```
bool isPIndexLayer();
```

获取属性索引字段名

```
string getPIndexField();
```

设置属性索引字段名

```
void setPIndexField( string field );
```

获取属性索引值

```
string getPIndexValue();
```

设置属性索引值

```
void setPIndexValue( string val );
```

## 6) 查找地物

按矩形范围查找地物，结果加入 Selection. 返回查找到的个数

```
int searchFeature( wkBox range );
```

按位置查找地物，结果加入 Selection. 返回查找到的个数

```
int searchFeature( wkPoint pt );
```

按属性条件进行查找

```
int searchFeature( string condition );
```

eg: layer.searchFeature( "featureId>2000 and createTime>'2008-1-1'" );

查找与地物几何满足指定的 9i 关系的地物，参见 `wcGeomConstant.h` 的 `wcGeom9i...`

```
int searchFeature( wkGeometry geom, int wcGeom9i );
```

从内存中查找：查找结果加入 `AnSelection`。返回查找到的个数

```
int searchAnnotation( wkBox range );
```

查找文字

```
int searchAnnotation( string text );
```

如：`layer.searchAnnotation( "砖" );`

按 `ref` 参考，以指定的 `mask` 查找与 `ref` 所有含同值的 `annotation`

```
int searchAnnotation( wkAnnotation ref, int mask );
```

参数 `mask` 参照 `wcAnnoConstant.h` 的 `wcAnnoInsPos` 到 `wcAnnoFeatureID`

查找与指定地物几何满足指定的 9i 关系的 `annotation`

```
int searchAnnotation( wkGeometry geom, int relation9i );
```

## 7) 其他

将选中地物打散到目标层

```
void scatterFeature(int nDispDef, wkLayer tarLayer, bool bUndoNeed);
```

参数 `nDispDef` 符号打散定义（参见 `wcStyleConstant.h` 的 `wcDiscrete`），如 63 打散到 G

参数 `tarLayer` 打散结果存放到目标层

参数 `bUndoNeed` 打散后是否允许 UNDO（建议取 `false`，特别是要打散的地物比较多时）

将选中地物打散并裁剪到目标层

```
void scatterFeature(int nDispDef, wkLayer tarLayer, wkGeometry trim, bool trimInside);
```

参数 `trim` 裁剪区域

参数 `trimInside` 擦除区域内部 `true`，擦除区域外 `false`

将选中文字串逐字符打散到目标层

```
void scatterAnnotation(wkLayer tarLayer, bool bUndoNeed);
```

参数 `tarLayer` 打散结果存放到目标层

参数 `bUndoNeed` 打散后是否允许 UNDO（建议取 `false`，特别是要打散的文字串比较多时）

参见《附录 5：样例及说明》之【向层中添加地物、文字和影像】

## 5、wkFeature：地物

获取地物类对象句柄，若为 0，表示对象不可用。

```
int handle( );
```

取地物 id

```
int getId();
```

置取地物的式样 id

```
int getStyleId();
```

```
void setStyleId( int styleId );
```

置取地物的修改状态

```
bool isModified();
```

```
void setModified( bool bModified );
```

获得地物的 geometry 对象（不得 free）

```
wkGeometry getGeometry();
```

重置地物的 geometry

```
void setGeometry( wkGeometry geom );
```

置取地物中的属性信息

```
string getInfo( wkLayer layer, string fieldName );
```

```
string getInfo(wkLayer layer, int fieldIndex);
```

```
void setInfo( wkLayer layer, string fieldName, string val );
```

```
void setInfo(wkLayer layer, int fieldIndex, string val);
```

参数 fieldIndex 是从 wkLayer.getFieldIndex(fieldName)取得的字段下标值（从 0 开始）

## 6、wkAnnotation：文字和影像

获取文字和影像类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

取文字或影像的 id

```
int getId();
```

置取式样 id

```
int getStyleId();
```

```
void setStyleId( int styleId );
```

置取修改状态

```
bool isModified();
```

```
void setModified( bool bModified );
```

置取文字串或影像路径

```
string getText();
```

```
void setText( string text );
```

text 为路径;

是否为影像

```
bool isDib();
```

置取文字或影像定位点( 文字 1 个定位点，影像为 4 个 )

```
wkPoint getRefPt( int at );
```

```
void setRefPt( wkPoint pt, int at );
```

置取文字块信息( 尺度以 0.1mm 为单位，角度以 0.1 度为单位 )

```
int getInfo( int mask );
```

```
void setInfo( int mask, int val );
```

参数 mask 参见 wcAnnoConstant.h 的 wcAnnoInsPos ... wcAnnoFeatureID

## 7、wkStyle：式样

获取式样类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

获取式样 id

```
int getId();
```

获取式样类型

```
int getType();
```

参见 wcStyleConstant.h 的 wcStyleType...

置取式样名

```
string getName();
```

```
void setName( string name );
```

置取式样说明

```
string getNote();
```

```
void setNote( string note );
```

创建式样

```
void create( wkLayer refLayer, int wcStyleType, string name, string note );
```

参数 wcStyleType 参见 wcStyleConstant.h

释放创建的式样

```
void free();
```

是否为用户自定义式样

```
bool isUserStyle();
```

点符号、线符号和面符号式样号>255 为用户自定义式样

置取点线面符号的式样号，文字字体名，位图符号名

```
string getStyle();
```

```
void setStyle( string style );
```

面边界的式样号要使用 `getInfo` 或 `setInfo` 的 `mask=15`

点线面式样号亦可使用 `getInfo( wcStyleStyle );`

参见 `wcStyleConstant.h` 的 `wcWalkPen`, `wcPattern`

获取式样信息

```
int getInfo( int mask );
```

```
void setInfo( int mask, int val );
```

参数 `maks` 参见 `wcStyleConstant.h` 的 `wcStyleColor ... wcStyleBorderTransparent`

## 8、wkGeometry：几何体

获取几何体类对象句柄，若为 0，表示对象不可用。

```
int handle( );
```

### 1) 创建几何体

创建几何体。凡是 `create` 得 `geometry`, 就必须 `free`

```
void create( wkPoint pt );
```

```
void create( wkPoint pt, double angle );
```

```
void create( wkBox box, bool bRing );
```

```
void create( array pts, int linetype, bool bRing );
```

```
void create( wkGeometry geom );
```

```
void create( wkPolygon poly );
```

```
void create( wkParts parts, bool bRing );
```

```
void create( wkPoints points, bool bRing );
```

释放创建的几何体

```
void free( );
```

几何类型:

```
int type( );
```

参见 `wcGeomConstant.h`: `wcGeomPoint`=点, `wcGeomLine`=线, `wcGeomRegion` 面

### 2) 多边形

获取多边形个数

```
int getPolygonCount( );
```

获取多边形 (at 从 0 开始)

```
wkPolygon getPolygonAt( int at );
```

添加一个多边形

```
void insertPolygonAt( wkPolygon poly );
```

删除一个多边形

```
void deletePolygonAt( int at );
```

替换一个多边形

```
void replacePolygonAt( wkPolygon poly, int at );
```

### 3) 线（及环）

获取线（及环）个数

```
int getPartsCount( );
```

获取线（at 从 0 开始）

```
wkParts getPartsAt( int at );
```

插入线或环

```
void insertPartsAt( wkParts parts, bool bRing, int at );
```

删除线

```
void deletePartsAt( int at );
```

替换线

```
void replacePartsAt( wkParts parts, int at );
```

### 4) 点和有向点

获取独立点数（不含线中的点）

```
int getPointCount( );
```

获取点（at 从 0 开始）

```
wkPoint getPointAt( int at );
```

插入点

```
void insertPointAt( wkPoint point, int at );
```

删除点

```
void deletePointAt( int at );
```

替换点

```
void replacePointAt( wkPoint point, int at );
```

### 5) OGC 计算

求与他几何体的九交关系

```
int get9i( wkGeometry B );
```

```
int get9i( wkGeometry B, int &r8bool, int &lineTouch, int &pointTouch );
```

返回结果参见 `wcGeomConstant.h` 的 `wcGeom9i...`

参数 `r8bool` 两个几何体“内、界、外”的 3x3 九交关系（每个关系按位存放）

设：this--A; 内: iA, iB; 界: dA, dB; 外: oA, oB;

判两要素相交(^)只有 1-true 或 0-false

0x87654321: 1--iA^iB, 2--iA^dB, 3--iA^oB

4--dA^iB, 5--dA^dB, 6--dA^oB

7--oA^iB, 8--oA^dB, 9--oA^oB==true（该位忽略）

注：由于两个几何体的外必为交，因此只返回 8 个关系。

`r8bool` 与 `walk9i` 若无对应（同样返回 `wcGeom9iNull`），调用者可自行推断。

参数 `lineTouch` 是否含线触

参数 `pointTouch` 是否含点触

获取与他几何体的布尔计算结果

`wkGeometry boolOp( wkGeometry B, int wcGeomOp );`

参见 `wcGeomConstant.h` 的 `wcGeomOp...`

`wcGeomOpIntersect` 0 求交

`wcGeomOpUnion` 1 求和

`wcGeomOpDifference` 2 求差

`wkGeometry boolOp(wkGeometry B, int wcGeomOp, double eps);`

求两个地物的共线: `wcGeomOpOverline==10` 多 parts

求两个地物的悬挂点: `wcGeomOpDangle==11` 多点

获得本几何体擦除 B 几何体的结果

`wkGeometry trim( wkGeometry B, bool trim_out );`

参数 `trim_out`: true--将 B 在本几何体的外边部分擦除，false--将 B 落在本几何体内部的部分擦除。若本 geometry 为点，B 为线，则返回 geometry 为被截断的多个 parts

求平面坐标面积

`double area();`

求椭球面积( 平方米 )

`double area( double dn, double de, bool b54, bool bDegree );`

参数 `dn, de`: 几何坐标转为赤道和含 500 公里标准高斯坐标所需平移量

参数 `b54`: 1-计算 54 椭球, 0-计算 80 椭球面积

参数 `bDegree`: 几何体坐标单位: 1-度, 0-米

周长

`double perimetry();`

型心点。面几何体的型心点必在面内最大内接圆的中心

`wkPoint centroid();`

几何体的外接矩形( ogc 称为 envelope )

`wkBox bounds();`

求本几何体的凸包

**wkGeometry convex**();

求 points 的凸包

**wkGeometry convex**( array points );

生成地物缓冲区

**wkGeometry buffer**(double offset, int wcEndcap, int wcJoin, int wcPenAlign);

**wkGeometry buffer**(array geometrys, double offset, int wcEndcap, int wcJoin, int wcPenAlign);

参数 wcEndcap 缓冲区端点类型

参数 wcJoin 缓冲区转折处类型

参数 wcPenAlign 缓冲方向

参见： `wcToolConstant.h`

几何体完整性检查

**bool isNormal**( double eps, int wcGeomCheck, **wkPoint&** errorPos );

参数 eps--重点限差、判定尖刺的限差、相触限差

参见 `wcGeomConstant.h` 的 `wcGeomCheck`--检查项标志

本地物与他地物的距离

**double distance**(**wkGeometry** geom);

本地物与点的距离

**double distance**(**wkPoint** pt);

6 参数线性变换

**void transform**( double a0, double a1, double a2, double b0, double b1, double b2 );

$X = a_0 + a_1 * x + a_2 * y$

$Y = b_0 + b_1 * x + b_2 * y$

平移

**void transform**( double dx, double dy );

等价于：  $a_0 = dx, a_1 = 1, a_2 = 0, b_0 = dy, b_1 = 0, b_2 = 1$

$X = dx + x$

$Y = dy + y$

顺向

**void putInOrder**();

反向

**void putInOtherOrder**();

拓扑重构

**void rebuild**();

按面积调整多边形(只调整第一个多边形)

```
void adjByArea(double newArea, array &fixedPoints);
```

参见《附录 5：样例及说明》之【几何体】

## 9、wkPolygon：多边形

获取多边形类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

获取多边形中的环数

```
int getRingCount();
```

获取多边形中的环

```
wkParts getRingAt( int at );
```

参数 at 从 0 开始

插入一个环

```
void insertRingAt( wkParts ring, bool bHole );
```

参数 bHole: true 为插入洞，否则为外环

删除一个环

```
void deleteRingAt( int at );
```

替换一个环

```
void replaceRingAt( wkParts ring, int at );
```

创建一个多边形

```
void create( wkPolygon poly );
```

```
void create( array pts, int linetype );
```

释放创建的多边形

```
void free();
```

```

【example】：
void main()
{
    wkPolygon poly;
    wkPoint pt1,pt2,pt3;
    pt1.x=10;
    pt1.y=10;
    pt2.x=20;
    pt2.y=10;
    pt3.x=15;
    pt3.y=30;
    array pts1,pts2;
    pts1.add(pt1);
    pts1.add(pt2);
    pts1.add(pt3);
    poly.create(pts1, 1);//创建多边形;
    int n1=poly.handle();
    int n2=poly.getRingCount();
    wkParts parts1=poly.getRingAt(0);
    poly.deleteRingAt(0);
    int n3=poly.getRingCount();
    trace("%d,%d,%d\n",n1,n2,n3);
    poly.free();
}
    结果：
19819016,1,0

```

## 10、wkParts：线流

获取线流类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

### 1) 对线串的操作

获取线串( 同线型 )个数

```
int getPointsCount();
```

获取一个线串 (at 从 0 开始)

```
wkPoints getPointsAt( int at );
```

插入一个线串

```
void insertPointsAt( wkPoints point, int at );
```

删除一个线串

```
void deletePointsAt( int at );
```

替换一个线串

```
void replacePointsAt( wkPoints point, int at );
```

## 2) 对线顶点的操作

获取线的顶点个数

```
int getVertexCount();
```

获取一个顶点 (at 从 0 开始)

```
wkPoint getVertexAt( int at );
```

插入一个顶点

```
void insertVertexAt( wkPoint pt, int at );
```

删除一个顶点

```
void deleteVertexAt( int at );
```

替换一个顶点

```
void replaceVertexAt( wkPoint point, int at );
```

获取距离点 refPt 最近的顶点 at

```
int nearestVertexAt( wkPoint refPt );
```

## 3) 创建及属性和其他操作

创建线

```
void create( wkParts parts );
```

```
void create( array pts, int linetype );
```

```
void create( wkPoint c, double r, double startAng, double endAng, bool bChord );
```

创建圆弧或扇形. startAng,endAng: 以弧度为单位

释放创建的串

```
void free();
```

是否为闭合线

```
bool isClosed();
```

是否为顺时针环 (要求闭合, 得出来的才正确)

```
bool isClockwise();
```

获取线的近角顶点, 参见 wc3x3Constant.h 的 wc3x3 ...

```
wkPoint getCorner( int nCorner_wc3x3 );
```

将线( 必须是环 )首点移到指定的顶点上

```
bool shiftHeadTo( wkPoint vertex );
```

反向

```
void reverse();
```

求线等间隔的  $n$  个切点以及切点处的切线方向( 弧度 )

```
array inters( int  $n$  );
```

切点的  $z$  值存放切线方向

## 11、wkPoints：线串

获取线串类对象句柄，若为 0，表示对象不可用。

```
int handle();
```

置取线串线型，参见 `wcGeomConstant.h` 的 `wcGeomLine`

```
int getLinetype();
```

```
void setLinetype( int linetype );
```

获取线串中的点个数

```
int getPointCount();
```

获取线串中的点 (at 从 0 开始)

```
wkPoint getPointAt( int at );
```

向线串中插点

```
void insertPointAt( wkPoint pt, int at );
```

从线串中删除点

```
void deletePointAt( int at );
```

替换线串中的点

```
void replacePointAt( wkPoint pt, int at );
```

创建线串

```
void create( wkPoints points );
```

```
void create( array pts, int linetype );
```

释放创建的线串

```
void free();
```

```

【example】：
void main()
{
    wkPoints points;
    array pts;
    wkPoint pt1,pt2,pt3,pt4;
    pt1.x=10;
    pt1.y=10;
    pt2.x=20;
    pt2.y=10;
    pt3.x=15;
    pt3.y=15;
    pt4.x=21;
    pt4.y=12;
    pts.add(pt1);
    pts.add(pt2);
    pts.add(pt3);
    points.create(pts,3);//创建线串，但下文必须 free;

    int n1=points.handle();
    points.setLinetype(1);
    int n2=points.getLinetype();//得到线串的类型;
    int n3=points.getPointCount();
    wkPoint pt5=points.getPointAt(0);//得到线串中第一个点;
    points.insertPointAt(pt4,3);
    points.deletePointAt(2);
    points.replacePointAt(pt1, 0);

    trace("%d,%d,%d,%d,%d\n",n1,n2,n3,toString(pt5));
    points.free();
} 结果：
19818584,1,3,
(1.0000000000000000e+001, 1.0000000000000000e+001, 0.0000000000000000e+000)

```

## 第五章 工具类对象

### 1、wkBox：矩形盒

矩形盒是否为空

```
bool isEmpty();
```

获取矩形盒宽和高

```
double w();
```

```
double h();
```

获取矩形盒的中心

```
wkPoint center();
```

获取矩形盒左下角点

```
wkPoint min();
```

获取矩形盒右上角点

```
wkPoint max();
```

点加入到矩形盒中

```
void add( double x, double y );
```

```
void add( wkPoint pt );
```

```
void add( wkBox box );
```

膨胀矩形盒

```
void inflate( double dx, double dy );
```

平移矩形盒

```
void offset( double dx, double dy );
```

与另一个矩形盒是否有相交( 有重叠 )

```
bool intersect( wkBox box );
```

【example】:

```
void main()
{
    wkBox box;
    wkPoint pt;
    pt.x=20;pt.y=20;pt.z=20;
    box.add(10,20);
    box.add(pt);
    trace("(%d,%d,%d),\n(%d,%d,%d),\n(%d,%d,%d),\n%d,%d\n",box.min().x,box.min().y,box.min().z,box.max().x,box.max().y,box.max().z,box.center().x,box.center().y,box.center().z,box.w(),box.h());
}
```

```

box.inflate(40,40);
box.offset(50,50);
trace("%s\n",toString(box.isEmpty()));
}

```

结果:

```

(0, 1076101120, 0),
(1077149696, 0, 0),
(0, 1077149696, 0),
1077149696, 0
0

```

## 2、wkGis : GIS 工具

判断 GIS 类对象是否有效, 若为 0, 表示对象不可用。

`bool valid( );`

### 1) GIS 网络分析

求中心点 pos 的服务区( 边界 rBestLine )

`wkGeometry findServiceArea( wkPoint pos, wkLayer route, string routePowers, string routeDirects, double crossTime, double serviceRange, bool bConvex, double eps );`

这里得到的 geometry 必须 free (下同)

参数 route 路网所在层

参数 routePowers 路网层的路段耗时字段名

参数 routeDirects 路网层的路段单向通行字段名

参数 crossTime 路网交叉口通过平均耗时

求投递员到投递点的最佳路线( wkGeometry 数组 )

`array findBestPostRoute( array posters, array dealLocs, wkLayer route, string routePowers, string routeDirects, double crossTime, bool bReturned, double eps );`

参数 bReturned—投递员投递邮件后是否返回邮局

求点 pos 到各医院之中的最近医院的路径

`wkGeometry findNearestHospital( wkPoint pos, array hospitals, wkLayer route, string routePowers, string routeDirects, double crossTime, double eps );`

求两点间的最短路径

`wkGeometry findBestRoute2Pos( wkPoint fromPt, wkPoint toPt, wkLayer route, string routePowers, string routeDirects, double crossTime, double eps );`

聚类分析 ( 分级或分区):

`array clustering(int dim, array &data, int m);`

参数 dim: 1--1 维, 2--2 维, 3--3 维

参数 data: 待分级的数据, 1 维要求为 double 数组, 2 维和 3 维要求为 wkPoint 数组

参数 m: 初始分区个数

【返回】聚类后的各元素的区间编号(int 类型数组)

求线 ln 的平行侧点的顶点集合

`wkGeometry dlPoints(wkPoints ln, double offset, bool bLeft);`

offset: 偏移

bLeft: true--平行线在线 ln 的左侧, false--在右侧

【返回】平行线

将线分裂为两条轴线

`wkGeometry dlPoints(wkPoints ln);`

【返回】若成功则几何体包含 2 个 parts, 每个 parts 为一轴线

在线 ln1 与线 ln2 中内插(如台阶)

`wkGeometry dlPoints(wkPoints ln1, wkPoints ln2, int insertLineNum);`

参数 insertLineNum: 内插线个数

【返回】: 几何体包含指定个数 parts, 每个 parts 为一内插线

以线 ln1 向线 ln2 做垂线或均分

`wkGeometry dlPoints(wkPoints ln1, wkPoints ln2, double offset, bool bVert);`

参数 offset: 间隔(若 $\leq 0$ , 则使用轴线 1 的顶点)

参数 bVert: true--做轴 1 到轴 2 的垂线, false--做两轴均分

【返回】几何体含 2 个 parts, 每个 parts 为线间隔点列

从 db 的 table 中按 filter 条件获取 Geometry 和 FeatureID

`array getGeometry(wkDb db, string table, string filter);`

`array getGeometry(wkDb db, string table, string filter, array &featureIds);`

【返回】wkGeometry 数组

### 3、wkClean：线素整理和构面

获取线素整理和构面类对象句柄, 若为 0, 表示对象不可用。

`int handle();`

#### 1) 线素处理

创建线素容器

`void create( wkBox range );`

释放创建的线素容器

`void free();`

向线素容器中添加线

`void addLine( wkParts line );`

添加结束

```
void addEnd();
```

线与容器中的线进行 xor，结果加入容器

```
int xorOverLine( wkParts line, double eps );
```

（\*等价于 **Clean**）线与容器中的线进行 or，结果加入容器

```
int orOverLine( wkParts line, double eps );
```

线与容器中的线进行 diff，结果加入容器

```
int diffOverLine( wkParts line, double eps );
```

线与容器中的线进行 and，结果加入容器

```
int andOverLine( wkParts line, double eps );
```

将线素容器中线构成弧段

```
int combineToArc( );
```

从线素容器中取第一条弧段

```
wkParts getFirstArc();
```

这里得到的 parts 必须 free

从线素容器中取下一条弧段

```
wkParts getNextArc();
```

若取得的 parts 为空，则应结束

清理线素容器中的超短线素

```
bool cleanZero( double eps );
```

## 2) 构面

用线素容器中的线素构面（**build** 后，下列函数可用）

```
bool build( double eps );
```

释放构面 **build** 的内存空间

```
void freeBuild();
```

**build** 后，**clean** 的函数不能再用，且必须使用 **freeBuild**()

获取弧段个数

```
int getArcCount();
```

获取多边形个数

```
int getPolygonCount();
```

按弧段号获取弧段（arcNo 从 1 开始）

`wkParts` **getArcAt**( `int` arcNo );

参数 arcNo--弧段号，从 1 开始( 因为多边形的弧度有正和负，所以从 1 开始 )

这里得到的 parts 必须 free

获得组成多边形的弧段号 at 序列 (polygonAt 从 0 开始)

`array` **getPolygonArcs**( `int` polygonAt );

若，弧段号<0，该弧段在本多边形逆时针；弧段号>0，该弧段在本多边形顺时针

获取多边形 (polygonAt 从 0 开始；这里得到的 geometry 必须 free )

`wkGeometry` **getPolygonAt**( `int` polygonAt );

参见《附录 5：样例及说明》之【线素整理和拓扑构面】

## 4、wkFile：读写文件

### 1) 文本文件读写

获取文件对象句柄，若为 0，表示对象不可用。

`int` **handle**( );

按指定方式 `wcFileMode` 打开文件 `fileName`。参见 `wcToolConstant.h`

`bool` **open**( `string` fileName, `string` wcFileMode );

关闭文件

`void` **close**( );

是否到文件尾

`bool` **eof**( );

从文件中获取一行

`string` **gets**( );

向文件中写一行

`void` **puts**( `string` str );

文本文件的读写操作

【example】：

```
void main()
{
    wkFile file;
    if (!file.open("c:\\walk\\readme.txt", "rt"))
        return;
    while(file.eof())
    {
```

```

        string line=file.gets();
        message(line);
    }
    file.close();
}

```

复制文件

```
bool copyFile(string existingFileName, string newFileName, bool bFailIfExists);
```

删除文件

```
bool deleteFile(string fileName);
```

## 2) INI 文件读写

置取 INI 文件的值

```
string iniGet( string section, string key, string defaults, string iniFile );
```

```
void iniWrite( string section, string key, string val, string iniFile );
```

对 INI 文件的读写不需要 open，不需要 close

【example】：

ini 的读写操作

```
wkFile ini;
```

```
ini.iniWrite("mySec", "myKey", "hello", "c:\\walk\\walk.ini");
```

## 3) 查找文件

fileFind 函数用来查找目录下指定的文件

```
bool findFile(string fileName);
```

参数 fileName：要查找的文件名，fileName 为空（""）则通配搜索（\*.\*）

使用该函数，则需要使用 findClose()来关闭句柄。

继续查找下一个文件

```
bool findNextFile();
```

查找结束后，必须调用 findClose 函数

```
void findClose();
```

```
string findFileGet(int getMask);
```

参数 getMask:

wcFileName--得到被找到的文件的名称，包括扩展名

wcFilePath--得到被找到的文件的全路径

wcFileTitle--得到被找到的文件的标题，不包括扩展名

wcFileURL--得到被找到的文件的 URL 网址，包括文件的路径

wcRoot--得到被找到文件的目录

**bool findFileIs**(**int** isMask);

参数 isMask:

IsArchived--是否存档

IsCompressed--是否压缩

IsDirectory--是否目录

IsDots-- 是否为"."或".."

IsHidden--是否隐藏

IsNormal--是否正常（没有别的属性）

IsReadOnly--是否为只读

IsSystem-- 是否为系统文件

IsTemporary-- 是否为临时文件

可以用 timeTm(...)返回的值去得到更多的详细时间信息

**double findFileTime**(**int** timeMask);

参数 timeMask:

wcCreationTime-- 获取文件创建的时间

wcLastAccessTime--获得文件最后一次被访问的时间

wcLastWriteTime-- 获得文件最后一次被修改和保存的时间

得到找到的文件的文件长度

**int findFileLength**(**bool** bM=false);

参数 bM:

false-- 以字节为单位

true-- 以兆为单位，如果长度小于一兆，则返回 1.

**【example】：**

```
//获取文件信息
wkFile find;
bool suc=find.findFile("c:\\walk\\*.mdb");
while(suc)
{
    suc=find.findNextFile();
    message(find.findFileGet(wcFileName));
}
find.findClose();
```

## 5、wkRgn：区域操作

**int handle**( );

由几何体 geom 创建一个区域

**void create**( **wkGeometry** geom );

释放创建的区域

**void free**( );

将几何体按指定方式 `wcRgnOp` 加入区域

```
void op( wkGeometry geom, int wcRgnOp );
```

参数 `wcRgnOp` 参见 `wcToolConstant.h`

判断点是否可见( 是否在区域内 )

```
bool isPtVisible( wkPoint pt );
```

判断矩形与区域是否有交或重叠

```
bool isRectIntersect( wkBox rect );
```

【example】：

```
void main()
{
    wkRgn  rgn;
    wkBox  box1;
    box1.add(0,0);
    box1.add(100,100);
    wkPoint pt;
    pt.x=1;
    pt.y=1;

    wkBox box2;
    box2.add(80,80);
    box2.add(200,200);

    wkGeometry geom;
    geom.create(box1,true);
    rgn.create(geom);
    geom.free();
    rgn.op(geom, 3);
    bool b1=rgn.isPtVisible(pt);
    bool b2=rgn.isRectIntersect(box2);
    int n1=rgn.handle();
    rgn.free();

    trace("%d,%s,%s\n",n1,toString(b1),toString(b2));
}
```

结果：

```
19186680,true,true
```

## 6、wkDialog：自定义对话框

对自定义无模态对话框的操作( 参见 `wcDialogConstant.h` )

获取对话框类对象句柄，若为 0，表示对象不可用。

```
int handle( );
```

创建无模式交互对话框( 目前仅支持系统内置的无模式交互对话框 )

```
void create( string caption );
```

```
void create( string caption, string wspFile );
```

参数 `wspFile`: 响应对话框消息处理脚本文件名. 若未指定, 则使用当前脚本响应消息

设置要响应的消息

```
void setAction( int ctrlId, int ctrlMsg );
```

```
void setAction( int mapMsg );
```

以循环等待模式响应控件消息

```
int peekAction(int &ctrlId);
```

【example】：

```
while(1)
{
    int act=dlg.peekAction(ctrlId);
    if (act == -1)//用户直接终止对话框
        break;
    ...//响应消息
}
```

添加控件

添加具有文字标题的控件，如静态文字、复选、多选、按钮控件

```
int addCtrl( int ctrlId, int wdCtrl_Type, int x, int y, int w, int h, int wdSty, string title );
```

添加无标题的控件，如文字编辑、单列表、下拉列表控件

```
int addCtrl( int ctrlId, int wdCtrl_Type, int x, int y, int w, int h, int wdSty );
```

添加多列列表控件

```
int addCtrl( int ctrlId, int wdCtrl_Type, int x, int y, int w, int h, int wdSty, int lvs_ex );
```

添加一个静态文字串

```
int addCtrl( string text, int x, int y );
```

参数 `wdSty`, `lvs_ex` 参见 `wcDialogConstant.h` 的“添加控件时的扩展式样”

获取 Walk 对话框内部句柄构造 `wkDialog` 实例

```
bool from( int hWnd);
```

由此构造的 `wkDialog` 对象不允许调用以上的 `create`, `addCtrl`, `setAction`, `getAction` 函数。

在一些 Walk 对话框内有脚本调用命令，向 `void main(array args)` 传递了窗口句柄，因此在相应的脚本中可按如下形式进行对话框的管理：

```

【example】:
void main(array args)
{
    if (args.getSize()>0)
    {
        wkDialog dlg;
        if (dlg.from(parseInt(args[0])))
        ...
    }
}

```

显示对话框

```
void show( bool bestFit=true );
```

显示或隐藏控件:

```
void show( int ctrlId, int wdSw );
```

若 ctrlId=0（对话框）且 wdSw 非 wdSwShow 则关闭对话框

设置对话框或控件的位置和大小

```
void move( int ctrlId, int x, int y, int width, int height );
```

不改变大小，只改变位置

```
void move( int ctrlId, int x, int y );
```

在控件上显示图片

```
bool setPic(int ctrlId, string picPath);
```

例如: dlg.setPic(id, "c:\\test.bmp");

置取对话框或控件标题

```
string getText( int ctrlId );
```

```
void setText( int ctrlId, string text );
```

置取控件的 check 状态 for checkButton, radioButton

```
int getCheck( int ctrlId );
```

```
int setCheck( int ctrlId, int check );
```

获取控件信息

```
int getInfo( int ctrlId, int mask );
```

mask 参见 wcDialogConstant.h 的 wdMask

置取指定行的文字串（多行控件: listBox, comboBox, listCtrl）

```
string getText( int ctrlId, int item );
```

```
void setText( int ctrlId, int item, string text );
```

清除所有行

```
void clear( int ctrlId );
```

获取行数

```
int count( int ctrlId );
```

置取当前选中行

```
int getCurSel( int ctrlId );
```

```
void setCurSel( int ctrlId, int item );
```

置取指定行是否选中

```
bool getSel( int ctrlId, int item );
```

```
void setSel( int ctrlId, int item, bool bSelected );
```

置取指定行的数据

```
int getData( int ctrlId, int item );
```

```
int setData( int ctrlId, int item, int data );
```

增删指定行

```
int insertItem( int ctrlId, int item, string text );
```

```
int deleteItem( int ctrlId, int item );
```

置取指定行的 check 状态（表控件：listCtrl）

```
bool getCheck( int ctrlId, int item );
```

```
void setCheck( int ctrlId, int item, bool check );
```

增删列( LVCFMT 参见 wcDialogConstant.h )

```
int insertColumn( int ctrlId, int column, string colName, int lvcfmt, int width );
```

```
bool deleteColumn( int ctrlId, int column );
```

置取指定行指定列的文字串

```
string getItemText( int ctrlId, int item, int subItem );
```

```
bool setItemText( int ctrlId, int item, int subItem, string text );
```

若 item=-1, 则取相应列的标题文字串

#### 【example】:

```
void main()
{
    wkDialog dlg;
    dlg.create("新的对话框 2");
    int i=5,j=5,k=400;
    int n7=dlg.addCtrl(170, 8,i,j, k,200,0x0004,"listctrl1");
    dlg.setCheck(170, 0,true);
    int n1=dlg.insertColumn(170,0,"列 1",2, 50);
    int n2=dlg.insertColumn(170,1,"列 2",2, 50);
    bool b1=dlg.setItemText(170,-1, 0,"第一列: 女生人数");
    bool b3=dlg.setItemText(170,-1, 1,"第二列: 男生人数");
    string str=dlg.getItemText( 170,0,2);
    bool b2=dlg.deleteColumn(170, 1);
}
```

```

dlg.show();
while(1)
{
    int act= dlg.peekAction(170);
    if (act == -1)
        break;
}
trace("%d,%d,%s,%s,%s,%s\n",n1,n2,toString(b1),toString(b2),toString(b3),str);
}

结果：
0,1,true,true,true

```

## 7、wkTrans：坐标变换

获取坐标变换类对象句柄，若为 0，表示对象不可用。

**int handle( );**

仅对 trans 和 project 有效

### 1) 经纬度与高斯平面直角坐标变换

高斯坐标转经纬度坐标

**wkPoint gaosi2Jwd( wkPoint gaosiPt, bool b54, double oriCL );**

参数 b54: true 为北京 54 坐标系, false 为西安 80 坐标系

参数 oriCL: 高斯投影带中央子午线, 以度为单位

经纬度坐标以度为单位 (下同)

经纬度坐标转高斯坐标

**wkPoint jwd2Gaosi( wkPoint jwdPt, bool b54, double oriCL );**

### 2) 同名点坐标系变换

若同名点有两个坐标系的坐标, 一个为 x 坐标系, 一个为 u 坐标系, 可先建立二者的转换模型, 然后将任意 x 坐标系的点坐标变换为 u 坐标系的点坐标。

根据两坐标系同名点集 xPoints 和 uPoints 创建转换模型

**bool transCreate( array xPoints, array uPoints, int wcTrans );**

**bool transCreate( string fromWtsFile );**

参数 wcTrans 配准模型号, 参见 wcToolConstant.h

释放创建的转换模型

**void transFree( );**

按转换模型进行坐标转换（输入 x 坐标系对象得到 u 坐标系对象结果）

```
void trans( wkPoint &aPoint );
void trans( wkGeometry &geom );
void trans( wkLayer layer, bool bSelectOnly );
```

获取转换模型信息

```
array transInfo( int mask );
```

参数 mask:（参见 Walk 产品手册）

```
0-配准模型号 mo      array[0]--int
1-配准模型中的 a 参数集  array[i]--double
2-配准模型中的 b 参数集  array[i]--double
3-配准模型中的 4 个加常数(x0, y0,u0,v0)  array[0-3]--double
4-配准模型中误差  array[0]--double
5-配准模型中的同名点数  array[0]--int
6-配准模型所需点数（与模型号相关） array[0]--int
7-配准模型中的 A 坐标系的同名点集 array [i]--wkPoint
8-配准模型中的 B 坐标系的同名点集 array [i]--wkPoint
9-配准模型的结果（字符串） array[0]--string
10-配准模型公式文字形式 array[0]--string
11-配准模型保存到打开对话框选择的文件中 array 空
```

保存转换模型

```
bool transSave( string toWtsFile );
```

### 3) 地图投影

创建地图投影

```
bool projectCreate( string prjParam );
```

参数 prjParam -- 地图投影，遵守 <http://trac.osgeo.org/proj/> 中的投影约定。

释放创建的地图投影

```
void projectFree( );
```

对点进行地图投影

```
void project( wkPoint &pt, bool bForward );
```

对层进行地图投影

```
int projectLayer( wkLayer srcLayer, wkLayer tarLayer );
```

### 4) 日照和 3D

创建 3D 实例

**bool view3DCreate();**

释放已创建 3D 实例

**void view3DFree();**

向 3D 实例添加一个几何底面，以及高度

**void view3DAdd(wkGeometry& geom, double height=0);**

设置太阳高和太阳方位角（以度为单位）

**void view3DSetSun(double height,double azimuth=0);**

获取一个几何体的所有 3D 面

**bool view3DGet3D(int buildId, array geomArray);**

获取所有 3D 面

**bool view3DGet3D(array arrayOfGeomArray);**

获取一个几何体的阴影面

**bool view3DGetShade(int buildId, array geomArray);**

获取所有阴影面

**bool view3DGetShade(array arrayOfGeomArray);**

获取一个几何体的阳面

**bool view3DGetLight(int buildId, array geomArray);**

获取所有阳面

**bool view3DGetLight(array &arr);**

设置透视参数

**void view3DSetPerspectiveView(double height, double azimuth, double distance);**

设置透视参数

**void view3DSetPerspectiveView(double height, double azimuth, wkPoint &centerPoint, double distance);**

设置透视参数

**void view3DSetPerspectiveView(wkPoint &eyePoint, wkPoint &targetPoint, wkPoint &topPoint, double eyeDistance);**

设置正视（太阳）参数

**void view3DSetOrthographicView(double height,double azimuth);**

设置正视（太阳）参数

**void view3DSetOrthographicView(wkPoint &eyePoint, wkPoint &targetPoint, wkPoint &topPoint);**

获取投影面

```
bool view3DGetProject(bool isContainShade, array& geomArray, array& info);
```

判断空间点是否正视可见（被太阳照射）

```
bool view3DTestSunPt(wkPoint &pt, bool &isVisible);
```

判断空间点是否正视可见（被太阳照射）

```
bool view3DTestSunPts(array& ptArr, array &res);
```

判断空间点是否透视可见

```
bool view3DTestViewPt(wkPoint &pt, bool &isVisible);
```

判断空间点是否透视可见

```
bool view3DTestViewPts(array &ptArr, array &res);
```

判断空间线可被正视的线段集（被太阳照射）

```
bool view3DTestSunLn(wkPoint &p1, wkPoint &p2, array& pointArr);
```

判断空间线可被透视的线段集

```
bool view3DTestViewLn(wkPoint &p1, wkPoint &p2, array& pointArr);
```

获取设定时间和地理位置的太阳高度角和方位角（以度为单位）

```
void view3DGetAngles(int year,int month,int day,int hour,int minute,int second,  
double longitude,double latitude,double &height,double &azimuth);
```

求空间点或面的投影：0-orthographic (正视), 1-perspectiv (透视)

```
void view3DProjection(wkPoint &p, int orthographic_perspectiv);
```

```
void view3DProjection(wkGeometry &geom, int orthographic_perspectiv);
```

```
void view3DProjection(array &geomArr, int orthographic_perspectiv);
```

参数 geomArr: 点型矩阵或者几何体型矩阵;

## 附录 1：常量定义

### 1、CMD 命令

1. gpedit.msc-----组策略
2. sndrec32-----录音机
3. Nslookup-----IP 地址侦测器
4. explorer-----打开资源管理器
5. logoff-----注销命令
6. tsshutdn-----60 秒倒计时关机命令
7. lusrmgr.msc----本机用户和组
8. services.msc---本地服务设置
9. oobe/msoobe /a---检查 XP 是否激活
10. notepad-----打开记事本
11. cleanmgr-----垃圾整理
12. net start messenger---开始信使服务
13. compmgmt.msc---计算机管理
14. net stop messenger-----停止信使服务
15. conf-----启动 netmeeting
16. dvdplay-----DVD 播放器
17. charmap-----启动字符映射表
18. diskmgmt.msc---磁盘管理实用程序
19. calc-----启动计算器
20. dfrg.msc-----磁盘碎片整理程序
21. chkdsk.exe-----Chkdsk 磁盘检查
22. devmgmt.msc--- 设备管理器
23. regsvr32 /u \*.dll---停止 dll 文件运行
24. drwtsn32----- 系统医生
25. rononce -p ----15 秒关机
26. dxdiag-----检查 DirectX 信息
27. regedt32-----注册表编辑器
28. Msconfig.exe---系统配置实用程序
29. rsop.msc-----组策略结果集
30. mem.exe-----显示内存使用情况
31. regedit.exe---注册表
32. winchat-----XP 自带局域网聊天
33. progman-----程序管理器
34. winmsd-----系统信息
35. perfmon.msc----计算机性能监测程序
2. 36. winver-----检查 Windows 版本
37. sfc /scannow-----扫描错误并复原

- 38. taskmgr----任务管理器 (2000 / xp / 2003
- 39. winver-----检查 Windows 版本
- 40. wmimgmt.msc---打开 windows 管理体系结构(WMI)
- 41. wupdmgr-----windows 更新程序
- 42. wscript-----windows 脚本宿主设置
- 43. write-----写字板
- 44. winmsd-----系统信息
- 45. wiaacmgr-----扫描仪和照相机向导
- 46. winchat-----XP 自带局域网聊天
- 47. mem.exe-----显示内存使用情况
- 48. Msconfig.exe---系统配置实用程序
- 49. mplayer2-----简易 windows media player
- 50. mspaint-----画图板
- 51. mstsc-----远程桌面连接
- 52. mplayer2-----媒体播放机
- 53. magnify-----放大镜实用程序
- 54. mmc-----打开控制台
- 55. mobsync-----同步命令
- 56. dxdiag-----检查 DirectX 信息
- 57. drwtsn32----- 系统医生
- 58. devmgmt.msc--- 设备管理器
- 59. dfrg.msc-----磁盘碎片整理程序
- 60. diskmgmt.msc---磁盘管理实用程序
- 61. dcomcnfg-----打开系统组件服务
- 62. ddshare-----打开 DDE 共享设置
- 63. dvdplay-----DVD 播放器
- 64. net stop messenger----停止信使服务
- 65. net start messenger---开始信使服务
- 66. notepad-----打开记事本
- 67. nslookup-----网络管理的工具向导
- 68. ntbackup-----系统备份和还原
- 69. narrator-----屏幕“讲述人”
- 70. ntmsmgr.msc---移动存储管理器
- 71. ntmsoprq.msc---移动存储管理员操作请求
- 72. netstat -an----(TC)命令检查接口
- 73. syncapp-----创建一个公文包
- 74. sysedit-----系统配置编辑器
- 75. sigverif-----文件签名验证程序
- 76. sndrec32-----录音机
- 77. shrpwb-----创建共享文件夹
- 78. secpol.msc----本地安全策略
- 79. syskey-----系统加密，一旦加密就不能解开，保护 windows xp 系统的多重密码

- 80. services.msc---本地服务设置
- 81. Sndvol32-----音量控制程序
- 82. sfc.exe-----系统文件检查器
- 83. sfc /scannow---windows 文件保护
- 84. tsshtutdn-----60 秒倒计时关机命令
- 85. tourstart-----xp 简介（安装完成后出现的漫游 xp 程序）
- 86. taskmgr-----任务管理器
- 87. eventvwr-----事件查看器
- 88. eudcedit-----造字程序
- 89. explorer-----打开资源管理器
- 90. packager-----对象包装程序
- 91. perfmon.msc---计算机性能监测程序
- 92. progman-----程序管理器
- 93. regedit.exe---注册表
- 94. rsop.msc-----组策略结果集
- 95. regedt32-----注册表编辑器
- 96. rononce -p ----15 秒关机
- 97. regsvr32 /u \*.dll---停止 dll 文件运行
- 98. regsvr32 /u zipfldr.dll-----取消 ZIP 支持
- 99. cmd.exe-----CMD 命令提示符
- 100. chkdsk.exe----Chkdsk 磁盘检查
- 101. certmgr.msc---证书管理实用程序
- 102. calc-----启动计算器
- 103. charmap-----启动字符映射表
- 104. cliconfg-----SQL SERVER 客户端网络实用程序
- 105. Clipbrd-----剪贴板查看器
- 106. conf-----启动 netmeeting
- 107. compmgmt.msc---计算机管理
- 108. cleanmgr-----垃圾整理
- 109. ciadv.msc-----索引服务程序
- 110. osk-----打开屏幕键盘
- 111. odbcad32-----ODBC 数据源管理器
- 112. oobe/msoobe /a----检查 XP 是否激活
- 113. lusrmgr.msc---本机用户和组
- 114. logoff-----注销命令
- 115. iexpress-----木马捆绑工具，系统自带
- 116. Nslookup-----IP 地址侦测器
- 117. fsmgmt.msc----共享文件夹管理器
- 118. utilman-----辅助工具管理器
- 119. gpedit.msc----组策略
- 120. explorer-----打开资源管理器

## 2、3x3 定位常量表-wc3x3Constant.h

3x3 定位常数

wc3x3_TL	1	
wc3x3_TC	2	
wc3x3_TR	3	
wc3x3_CL	4	
wc3x3_CC	5	
wc3x3_CR	6	
wc3x3_BL	7	
wc3x3_BC	8	
wc3x3_BR	9	
Pi	3.141592653589793	

### 3、wkView 常量表-wcViewConstant.h

捕捉类型

常量名	值	说明
wcSnapCircle	0	圆心
wcSnapCut	1	切线
wcSnapDivide	2	中间点
wcSnapExtend	3	延长求交
wcSnapIcon	4	格网点
wcSnapIntersection	5	交叉点
wcSnapNearest	6	线上点--最近点
wcSnapNode	7	端点
wcSnapParallelLine	8	平行线
wcSnapVertex	9	垂足点
wcSnapLineInter	10	直线直线交点
wcSnapVertInter	11	直线垂线交点
wcsnapVertPoint	12	垂线垂足
wcSnapAngleLine	13	定角线
wcSnapQuadrant	14	象限点
wcSnapNull	15	无效——不捕捉

图形输入类型

wcInputPoint	0	点
wcInputStraight	1	折线
wcInputBezier	2	Bezier 样条; 曲线不一定经过样点
wcInput3PArc	3	3点圆弧; 3点: 起点, 中点, 终点; 并标志了圆弧的方向
wcInputRArc	4	半径圆弧; 3点: 圆心, (半径, 圆弧方向:1-逆时针,-1-顺时针), (起点角, 终点角(以弧度为单位))
wcInputRectArc	5	矩形椭圆弧; 4点: 左上角坐标, 右下角坐标, 圆心到弧起点上的某点坐标, 点的坐标
wcInput3PCircle	6	3点圆; 圆上3点, 并标志了圆的方向
wcInputRCircle	7	半径圆; 2点: 圆心, (半径, 圆弧方向:1-逆时针,-1-顺时针)
wcInputRectCircle	8	矩形椭圆; 2点: 左上角坐标, 右下角坐标
wcInputStrainCurve	10	张力样条; 插值数和插值方法在系统参数中规定
wcInputVertAngLine	11	直角线
wcInputRectangle	14	矩形
wcInputTypeNocare	999	不关心输入类型

当前图形窗口鼠标移动到地物上时的提示状态:

wcMouseTipNull	0	无提示
wcMouseTipStyle	1	提示层和式样
wcMouseTipProperty	2	提示属性

当前图形窗口的鼠标选取地物和文字的状态

wcSelectNull	0	不许选择
wcSelectFeature	1	只许选地物
wcSelectAnnotation	2	只许选文字
wcSelectBoth	3	可选地物和文字

## 4、坐标系统和尺度单位常量表-wcSRSCoordinate.h

坐标投影类型常数

wcNonEarth	0	独立平面坐标系
wcLongLat	1	经纬度全球坐标系（默认 80 系）
wcGaussKrugerChina54	10054	support for china 6 度带，以米为单位
wcGaussKrugerChina80	10080	support for china 6 度带，以米为单位
wcGaussKruger3China54	10354	support for china 3 度带，以米为单位
wcGaussKruger3China80	10380	support for china 3 度带，以米为单位
wcUTMercatorWGS84	10084	support for china, 以米为单位
wcLongLatChina54	10154	support for china, 以度为单位
wcLongLatChina80	10180	support for china, 以度为单位
wcLongLatWGS84	10184	support for china, 以度为单位

坐标单位和尺度常数

wcUnitMile	0	英里
wcUnitKilometer	1	公里
wcUnitInch	2	英寸
wcUnitFoot	3	英尺
wcUnitYard	4	码
wcUnitMillimeter	5	毫米
wcUnitCentimeter	6	厘米
wcUnitMeter	7	米
wcUnitSurveyFoot	8	US Survey feet
wcUnitNauticalMile	9	海里
wcUnitTwip	10	
wcUnitPoint	11	
wcUnitPica	12	
wcUnitDegree	13	degree
wcUnitLink	30	令
wcUnitChain	31	链
wcUnitRod	32	竿

面积常数

wcUnitSquareMile	14	平方英里
wcUnitSquareKilometer	15	平方公里
wcUnitSquareInch	16	平方英尺
wcUnitSquareFoot	17	平方英寸
wcUnitSquareYard	18	平方码
wcUnitSquareMillimeter	19	平方毫米
wcUnitSquareCentimeter	20	平方厘米
wcUnitSquareMeter	21	平方米
wcUnitSquareSurveyFoot	22	平方 US Survey feet
wcUnitSquareNauticalMile	23	英亩(这个可能有问题)
wcUnitSquareTwip	24	
wcUnitSquarePoint	25	
wcUnitSquarePica	26	
wcUnitSquareDegree	27	
wcUnitAcre	28	公顷
wcUnitHectare	29	杆

wcUnitSquareLink	33	平方令
wcUnitSquareChain	34	平方链
wcUnitSquareRod	35	平方竿
wcUnitPerch	36	
wcUnitRood	37	路德
wcUnitMU	38	

## 5、wkLayer 常量表- wcLayerConstant.h

数据库类型

wcDbTypeMdb	0	access 的 mdb 类型
wcDbTypeSqlServer	1	sqlserver 类型

字段类型

wcSQL_CHAR	1	char
wcSQL_NUMERIC	2	numeric
wcSQL_DECIMAL	3	decimal
wcSQL_INTEGER	4	int
wcSQL_SMALL	5	short
wcSQL_FLOAT	6	float
wcSQL_REAL	7	real
wcSQL_DOUBLE	8	double
wcSQL_DATE	9	date10
wcSQL_TIME	10	time8
wcSQL_TIMESTAMP	11	timestamp19
wcSQL_VARCHAR	12	varchar
wcSQL_LONGVARCHAR	-1	text
wcSQL_BINARY	-2	binary
wcSQL_VARBINARY	-3	binary
wcSQL_LONGVARBINARY	-4	binary
wcSQL_BIG	-5	
wcSQL_TINY	-6	
wcSQL_BIT	-7	bool
wcSQL_GUID	-11	

层的类型

wcLayerNormal	1	一般层
wcLayerVirtual	2	虚层
wcLayerHideIndexed	6	隐式索引层

将层置于工作空间的

wcLayerOrderTop	0	置顶
wcLayerOrderBotton	-1	置底

## 6、wkAnnotation 常量表-wcAnnoConstant.h

文字和影像定位类型

wcAnnoLocBPoint	1	基点定位 标定 Annotation 的中心。如为字符串则为水平无间隔展开的字符串外接矩形的中心。
wcAnnoLocBLine	2	基线定位 标定 Annotation 的横向中心轴线（可以是折线或 Bezier 曲线）。
wcAnnoLocBRect	3	四角定位 标定 Annotation 的四角坐标，如用于图纸定位。
wcAnnoLocMultiP	4	多点定位 主要用于“大图象”的多点纠正。
wcAnnoLocExtRect	5	扩展四角定位 标定 Annotation 的四角坐标，并且撑满整个 Rect
wcAnnoLocBText48	6	文字块参考点定位
wcAnnoLocCurveMultiP	7	随线注记的多点定位类型。

修饰类型

wcAnnoTypeText	1	文字
wcAnnoTypeDib	6	图象
wcAnnoTypeBText	10	文字块注记

置取文字块信息（尺度以 0.1mm 为单位，角度以 0.1 度为单位）

wcAnnoInsPos	10	标注插入位置
wcAnnoInsPntX	11	标注插入点与插入位置的偏移 X
wcAnnoInsPntY	12	标注插入点与插入位置的偏移 Y
wcAnnoAngle	13	标注的旋转角度(以 0.1 度为单位)
wcAnnoFix	14	标注的块大小是否随比例缩放
wcAnnoCharH	15	标注的字符高度（以 0.1mm 为单位）
wcAnnoLineI	16	标注的文本行间隔
wcAnnoCharI	17	标注的文本的字符间隔
wcAnnoCharLeft	18	标注的文本字头的朝向：倾斜方向,分为左斜/右斜/垂直/不斜
wcAnnoCharAngle	19	标注的文本字头的倾斜角度
wcAnnoColTwo	20	标注是否使用双栏
wcAnnoColStager	21	标注两栏是否错行
wcAnnoColWidthSet	22	标注是否使用栏宽设定
wcAnnoCol1Width	23	标注的第一栏的栏宽
wcAnnoCol2Width	24	获取标注的第二栏的栏宽
wcAnnoRowAlian	25	标注文本左栏对齐方式
wcAnnoRow2Alian	26	标注文本右栏是对齐方式
wcAnnoBlkBox	27	标注是否绘制外包盒
wcAnnoLinGapLine	28	标注左栏是否绘制行间线
wcAnnoLineWidth	29	标注线宽
wcAnnoColLine	30	标注是否绘制栏间线
wcAnnoLineRGapLine	31	标注右栏是否绘制双数行间线
wcAnnoExline	32	标注的引出线类型
wcAnnoExlineGap	33	标注的引出线距参考点间距

wcAnnoExlinePos	34	标注引出线引向区位选择因子 1
wcAnnoExlinePos2	35	标注引出线引向区位选择因子 2（参见上）
wcAnnoCircle	36	取标注是否显示边框外接圆
wcAnnoFeatureID	37	标注的地物来源（FeatureID），若无参考地物返回 0
wcAnnoImageWidth	38	iDibWidth: 图像宽度
wcAnnoImageHeight	39	iDibHeight: 图像高度

标注文本栏内对齐方式

wcAnnoAlignCenter	0	居中
wcAnnoAlignLeft	1	左对齐
wcAnnoAlignRight	2	右对齐
wcAnnoAlignFit	3	两端拉伸对齐

## 7、wkStyle 常量表-wcStyleConstant.h

式样类型

wcStyleTypeLine	1	线样式
wcStyleTypeRegion	2	区域样式
wcStyleTypeSymbolBitmap	3	多媒体样式
wcStyleTypeSymbol	4	符号样式
wcStyleTypeText	5	文字样式

用户自定义样式类型

wcUserStyleL	1	(L) 对应简单线
wcUserStyleH	2	(H) 区域样式
wcUserStyleG	4	(G) 符号样式
wcUserStyleLG	6	(LG) 符号线
wcUserStyleLD	7	(LD) 简单双线
wcUserStyleLM	8	(LM) 图元线
wcUserStyleLC	9	(LC) 复合线
wcUserStyleDL	10	(DL) 双轴线

用户自定义符号中的坐标和尺度单位:

wcUserSymbolUnitDefaul	0	系统默认单位, 一般以 0.1mm 为单位
wcUserSymbolUnit01mm	1	以 0.1mm 为单位
wcUserSymbolUnit001mm	2	以 0.01mm 为单位
wcUserSymbolUnitPpoint	3	以 1/72 Inch 为单位

获取式样信息

wcStyleStyle	6	点线面式样值
wcStyleColor	7	前景色: 线颜色, 面前色, 文字颜色, 点符号颜色, 位图颜色
wcStyleBackColor	8	背景色: 线底色, 面底色, 文字框颜色, 点符号底色, 位图替换色
wcStyleSize	9	尺寸: 线宽, 面边界线宽, 文字体字高, 点符号高, 位图尺寸
wcStyleSizeUnit	10	尺寸单位: 线宽单位, 面边界线宽单位, 点符号高单位
wcStyleTransparent	11	透明或加框: 线透明, 面透明, 文字加框, 点符号透明, 位图透明
wcStyleLineJoin	12	线折角绘法, 或文字是否加下划线
wcStyleLineAlign	13	线端点绘法, 或文字是否加阴影
wcStyleLineLinearLen	14	绘 GDI+线的色宽
wcStyleBorderStyle	15	面边界式样号
wcStyleBorderColor	16	面边界颜色
wcStyleBorderBackColor	17	面边界背景色
wcStyleBorderTransparent	18	面边界透明

式样类型作用范围定义

wcWalkStyleIDBegin	1	Windows 式样号从 1 开始
wcSysStyleIDBegin	256	系统内部式样号从 256 开始
wcUserStyleIDBegin	2001	用户自定义式样号从 2001 开始

## 笔类型

wcPenNone	0	空（不绘线）
wcPenSolid	1	实线
wcPenDashPen	2	虚线: 3-DotPen, 4-DashdotPen, 5-DdashdotdotPen
wcPenPer05	6	纹理填充线: 7-Per10, ..., 17-Per90
wcPenLinearHori	53	8 个线性渐变刷: 54-LinearVert, ..., 60-LinearBackXY

## 填充类型

wcPatternHollow	0	空（不填充）
wcPatternSolid	1	实色填充
wcPatternHoriHatch	2	纹理填充: 3-VertHatch, 4-U45Hatch, 5-D45Hatch, 6-CrossHatch, 7-DiagHatch
wcPatternPer05	8	9-Per10, ..., 19-Per90
wcPatternLinearHori	55	8 个以线性渐变刷: 56-LinearVert, ..., 62-LinearBackXY
wcPatternPathBox	63	以面的外接矩形向中央进行颜色过渡
wcPatternPathVertex	64	以面的边界向中央进行颜色过渡

## 地物符号打散深度定义

wcDiscreteNo	0	不打散
wcDiscreteH	1	打散面的边界
wcDiscreteDL	2	打散到 DL
wcDiscreteLC	4	打散到 LC
wcDiscreteLG	8	打散到 LG
wcDiscreteLM	16	打散到 LM
wcDiscreteLD	32	打散到 LD
wcDiscreteG	64	打散到 G

## 8、式样颜色常量表-wcStyleColors.h

颜色常数表

wcColorBlack	0	
wcColorRed	255	
wcColorGreen	65280	
wcColorBlue	16711680	
wcColorMagenta	16711935	
wcColorCyan	16776960	
wcColorWhite	16777215	
wcColorLightGray	12632256	
wcColorDarkGray	4210752	
wcColorGray	8421504	
wcColorPaleYellow	13697023	
wcColorLightYellow	8454143	
wcColorYellow	65535	
wcColorLimeGreen	12639424	
wcColorTeal	8421440	
wcColorDarkGreen	16384	
wcColorMaroon	128	
wcColorPurple	8388736	
wcColorOrange	33023	
wcColorKhaki	7051175	
wcColorOlive	32896	
wcColorBrown	4210816	
wcColorNavy	8404992	
wcColorScrollBars	0x80000000	
wcColorDesktop	0x80000001	
wcColorActiveTitleBar	0x80000002	
wcColorInactiveTitleBar	0x80000003	
wcColorMenuBar	0x80000004	
wcColorWindowBackground	0x80000005	
wcColorWindowFrame	0x80000006	
wcColorMenuText	0x80000007	
wcColorWindowText	0x80000008	
wcColorTitleBarText	0x80000009	
wcColorActiveBorder	0x8000000A	
wcColorInactiveBorder	0x8000000B	
wcColorApplicationWorkspace	0x8000000C	
wcColorHighlight	0x8000000D	
wcColorHighlightText	0x8000000E	
wcColorButtonFace	0x8000000F	
wcColorButtonShadow	0x80000010	
wcColorGrayText	0x80000011	
wcColorButtonText	0x80000012	
wcColorInactiveCaptionText	0x80000013	
wcColor3DHighlight	0x80000014	
wcColor3DDarkShadow	0x80000015	
wcColor3DLight	0x80000016	
wcColorInfoText	0x80000017	
wcColorInfoBackground	0x80000018	

## 9、wkGeometry 常量表-wcGeomConstant.h

几何类型：

wcGeomPoint	1	点
wcGeomLine	2	线
wcGeomRegion	4	面

几何体的布尔计算结果

wcGeomOpIntersect	0	求交
wcGeomOpUnion	1	求和
wcGeomOpDifference	2	求差

几何体完整性检查

wcGeomCheckXY	1	坐标有效：x,y 的坐标值有效；线裂开（相邻 walkPoints 的端点应严格重合）
wcGeomCheckH	2	高程有效：z 值有效。
wcGeomCheckOverlapPt	4	线上重点（相邻点间距小于 eps）
wcGeomCheckIlliLine	8	非法线段（如 3 点弧只有 2 个点、单点线）
wcGeomCheckZeroClose	16	面的环首尾应闭合，环面积不得为零
wcGeomCheckSelfInter	32	线自交和尖刺（eps 为限差）
wcGeomCheckClockwise	64	要求外环顺时针、洞为逆时针
wcGeomCheckPolygon	128	环环无交、无触；洞内不得含洞
wcGeomCheckRegion	256	多边形与多边形相交
wcGeomCheckAll	-1	全部检查

9i 关系（若要设置多个关系，可用或 ‘|’ 连起来）

wcGeom9iNull	1	无关
wcGeom9iEqual	2	相等
wcGeom9iContain	4	包含
wcGeom9iIntersect	8	相交
wcGeom9iTouch	16	相触
wcGeom9iInnerTouch	32	内触
wcGeom9iOuterTouch	64	外触

线串线型

wcGeomLineStraight	1	折线
wcGeomLine3PArc	3	3 点圆弧，3 点:起点，中点,终点，并标志了圆弧的方向
wcGeomLine3PCircle	6	3 点圆，圆上 3 点,并标志了圆的方向
wcGeomLineBCurve	9	B 样条，插值数和插值方法在系统参数中规定
wcGeomLineSCurve	10	张力样条，插值数和插值方法在系统参数中规定

## 10、wkFile 等工具类常量表-wcToolConstant.h

生成地物缓冲区

拐角定义

wcJoinRound	0	圆角
wcJoinBevel	1	切角
wcJoinMiter	2	尖角

端点类型

wcEndcapRound	0	圆角
wcEndcapSquare	1	方角
wcEndcapBevel	2	平角

笔侧向

wcPenAlignCenter	0	(默认) 居中
wcPenAlignLeft	1	宽线绘制在中心线的左侧
wcPenAlignRight	2	宽线绘制在中心线的右侧
wcPenAlignCust	3	保留: 自定义 (沿笔宽方向的绘线间隔数组)

对专题要素数组分级方法

wcReclassNature	0	自然分级 (忽略 reclass_number 和 reclass_round)
wcReclassEqualInterval	1	等间隔
wcReclassEqualAccuracy	2	等精度 (忽略 reclass_number)
wcReclassEqualFrequency	3	等频度
wcReclassSquareRoot	4	平方
wcReclassSquare	5	平方根
wcReclassLn	6	自然对数

fopen 的 打开模式

wcFileModeRead	"rt"	按读方式打开文本文件
wcFileModeWrite	"wt"	按写方式打开文本文件

findFile 的参数

for findFile getMask

wcFileName	1	文件名 (含扩展名)
wcFilePath	2	全路径
wcFileTitle	3	无扩展名的文件名
wcFileURL	4	URL
wcRoot	5	根路径

for findFileIs isMask

IsArchived	1	是否存档
IsCompressed	2	是否为压缩文件
IsDirectory	3	是否为目录
IsDots	4	是否为 ./ 或 ../
IsHidden	5	是否为隐藏文件
IsNormal	6	是否正常
IsReadOnly	7	是否为只读文件
IsSystem	8	是否为系统文件

IsTemporary	9	是否为临时文件
-------------	---	---------

for findFileTime timeMask

wcCreationTime	1	文件的创建时间
wcLastAccessTime	2	文件的上次访问时间
wcLastWriteTime	3	上次写文件的时间

将几何体按指定方式 op 加入区域

wcRgnOpComplement	0	区域的 complement 操作
wcRgnOpExclude	1	区域的 exclude 操作
wcRgnOpIntersect	2	区域的 intersect 操作
wcRgnOpUnion	3	区域的 union 操作
wcRgnOpXor	4	区域的 xor 操作

根据两坐标系同名点集 xPoints 和 uPoints 创建转换模型, mo 配准模型号

wcTransMove	1	1 点--平移变换--2 系数
wcTransRotate	2	2 点--平移旋转(等比)变换--4 系数
wcTransScale	3	2 点--平移旋转缩放(贝尔曼托)变换--4 系数
wcTransAffine	4	3 点--仿射变换--6 系数
wcTransQuadrang	5	4 点--四边形畸变--8 系数
wcTransQuantic2	6	6 点--2 次方程--12 系数
wcTransQuantic3	7	10 点--3 次方程--20 系数
wcTransQuantic4	8	15 点--4 次方程--30 系数
wcTransQuantic5	9	21 点--5 次方程--42 系数

Dem 功能

wcDemMainDlg	-1	打开功能选择对话框
wcDemLoad	0	加载 DEM 文件生成 DEM 主题
wcDemFindDistance	1	测定距离
wcDemProximityMap	2	邻近制图
wcDemDensity	3	密度功能
wcDemDgx2Dem	4	由 dgx 生成 Grid-Dem
wcDemDgx	5	由 DEM 提取等高线
wcDemDgxPoint	6	提取过点的等高线
wcDemSection	7	提取一条线的 DEM 断面
wcDemSlope	8	坡度
wcDemAspect	9	坡向
wcDemVisual	10	可视区
wcDemReclass	11	重分类
wcDemLocalStat	12	5-1 局部统计功能 Local Statistical Function
2 个以上的 DEM 主题, 统计运算		
wcDemZonal	13	5-2 分区功能 Zonal Functions
2 个主题, 统计运算, 扩展层的属性到包含统计量		
wcDemMapQuery	14	6-1 选择功能 Map Query
从输入的栅格主题中提取和选择出一个栅格单元的子集		
wcDemMapCalc	15	6-2 数学运算功能 Map Calculator
对多个 DEM 主题的元素进行函数运算		
wcDemNeighborStat	16	7-1 邻域分析 Neighborhood Statistics
1 个 DEM 主题和 1 个矢量层 (暂时为点层)		

wcDemSaveGrdE00	17	将 DEM 保存为 GRD/E00
wcDemMessure	18	DEM 量测功能
wcDemThemeMan	19	DEM 主题层管理
wcDemImage2Dem	20	图像输出 DEM 文件
wcDemPolygonPoint	21	提取含点多边形
wcDemPolygon	22	提取主题多边形

## Excel 文件格式设置的常量

设置单元格格式化字符串		
XLS_FORMAT_GENERAL	"General"	单元格设置为普通
XLS_FORMAT_TEXT	"@"	设置为 Tex
XLS_FORMAT_INTEGER	"0"	设置为整型
XLS_FORMAT_DECIMAL	"0.00"	设置为浮点型
XLS_FORMAT_PERCENT	"0%"	百分比
XLS_FORMAT_DATE	"M/D/Y Y"	日期型
XLS_FORMAT_TIME	"h:mm:ss"	短时间
XLS_FORMAT_DATETIME	"M/D/Y Y h:mm"	长时间
设置单元格字体: option, weight, escapementType, underlineType, colorIndex		
option		
EXCEL_FONT_BOLD	0x01	
EXCEL_FONT_ITALIC	0x02	
EXCEL_FONT_UNDERLINED	0x04	
EXCEL_FONT_STRUCK_OUT	0x08	
EXCEL_FONT_OUTLINED	0x10	
EXCEL_FONT_SHADOWED	0x20	
EXCEL_FONT_CONDENSED	0x40	
EXCEL_FONT_EXTENDED	0x80	
weight		
EXCEL_FW_DONTCARE	0	
EXCEL_FW_THIN	100	
EXCEL_FW_EXTRALIGHT	200	
EXCEL_FW_LIGHT	300	
EXCEL_FW_NORMAL	400	
EXCEL_FW_MEDIUM	500	
EXCEL_FW_SEMIBOLD	600	
EXCEL_FW_BOLD	700	
EXCEL_FW_EXTRABOLD	800	
EXCEL_FW_HEAVY	900	
EXCEL_FW_ULTRALIGHT	EXCEL_FW_EXTRALIGHT	
EXCEL_FW_REGULAR	EXCEL_FW_NORMAL	
EXCEL_FW_DEMIBOLD	EXCEL_FW_SEMIBOLD	
EXCEL_FW_ULTRABOLD	EXCEL_FW_EXTRABOLD	
EXCEL_FW_BLACK	EXCEL_FW_HEAVY	
escapementType		
EXCEL_ESCAPEMENT_NONE	0	
EXCEL_ESCAPEMENT_SUPERSCRIPT	1	
EXCEL_ESCAPEMENT_SUBSCRIPT	2	

underlineType		
EXCEL_UNDERLINE_NONE	0x00	
EXCEL_UNDERLINE_SINGLE	0x01	
EXCEL_UNDERLINE_DOUBLE	0x02	
EXCEL_UNDERLINE_SINGLE_ACCOUNTING	0x21	
EXCEL_UNDERLINE_DOUBLE_ACCOUNTING	0x22	
colorIndex		
EGA_BLACK	0	000000H
EGA_WHITE	1	FFFFFFH
EGA_RED	2	FF0000H
EGA_GREEN	3	00FF00H
EGA_BLUE	4	0000FFH
EGA_YELLOW	5	FFFF00H
EGA_MAGENTA	6	FF00FFH
EGA_CYAN	7	00FFFFH
单元格设置		
horizontal alignment		
EXCEL_HALIGN_GENERAL	0x00	
EXCEL_HALIGN_LEFT	0x01	
EXCEL_HALIGN_CENTRED	0x02	
EXCEL_HALIGN_RIGHT	0x03	
EXCEL_HALIGN_FILLED	0x04	
EXCEL_HALIGN_JUSTIFIED	0x05	
EXCEL_HALIGN_SEL_CENTRED	0x06	
EXCEL_HALIGN_DISTRIBUTED	0x07	
vertical alignment		
EXCEL_VALIGN_TOP	0x00	
EXCEL_VALIGN_CENTRED	0x10	
EXCEL_VALIGN_BOTTOM	0x20	
EXCEL_VALIGN_JUSTIFIED	0x30	
EXCEL_VALIGN_DISTRIBUTED	0x40	
EXCEL_JUSTIFY_LAST_LINE	0x80	

## 11、wkDialog 常量表-wcDialogConstant.h

主图形窗口、对话框及控件的 ID

wdIdWalkMap	-99	主图形窗口的 ctrlId
wdIdStatic	-1	静态文本的 ctrlId
wdIdDialog	0	对话框的 ctrlId
wdIdOk	1	对话框【确认】键的 ctrlId
wdIdCancel	2	对话框【取消】键的 ctrlId
wdIdAbort	3	对话框 Windows 默认的 ctrlId
wdIdRetry	4	
wdIdIgnore	5	
wdIdYes	6	
wdIdNo	7	
wdIdClose	8	
wdIdHelp	9	

wdIdButton	110	按钮
wdIdCheck	120	<input type="checkbox"/> 按钮
wdIdRadio	130	多选一按钮
wdIdEdit	140	文本输入窗
wdIdListBox	150	列表
wdIdComboBox	160	下列列表
wdIdListCtrl	170	多列列表

对话框的控件类型

wdCtrlText	1	static_text
wdCtrlEdit	2	edit
wdCtrlButton	3	button
wdCtrlCheck	4	check button
wdCtrlRadio	5	radio button
wdCtrlListBox	6	list box
wdCtrlComboBox	7	combo box
wdCtrlListCtrl	8	list ctrl

控件默认尺寸

wdSizeDefault	-1	static text default height=8, width=25 check button default height=10 radio button default height=10 edit default height=12, width=40 button default height=14, width=50
---------------	----	--

对话框打开后，鼠标在主图形窗口活动的消息：

wdMapLbtnDown	1000	在主图形窗口鼠标左键压下
wdMapLbtnUp	1001	在主图形窗口鼠标左键抬起
wdMapMouseMove	1002	在主图形窗口鼠标移动
wdMapRbtnDown	1003	在主图形窗口鼠标右键压下
wdMapRbtnUp	1004	在主图形窗口鼠标右键抬起
wdMapLbtnDbclk	1005	在主图形窗口鼠标左键双击
wdMapRbtnDbclk	1006	在主图形窗口鼠标右键双击
wdMapFresh	1007	主图形窗口刷新

## 对话框消息

wdMsgBnClicked	1	按钮(Button Notify)压下
wdMsgSelchange	1	改变了 ListBox 当前选中项, 改变了 ComboBox 当前选中项, 改变了 ListCtrl 的当前 item
wdMsgEnKillFocus	1	Edit Box 失去焦点
wdMsgNmDblclk	2	在 ListCtrl 双击了鼠标
wdMsgNmRclicked	3	在 ListCtrl 点击了鼠标右键
wdMsgNone	0	无消息

## 设置控件的状态

wdSwHide	0	隐藏
wdSwShow	1	显示
wdSwDisable	2	置灰
wdSwEnable	3	取消置灰

## 获取控件信息 getInfo(ctrlId, mask) 的 mask

wdMaskCtrlType	1	获取控件类型: wdCtrlText,...,wdCtrlListCtrl
wdMaskHide	2	是否 hide
wdMaskDisable	3	是否 Disable
wdMaskReadOnly	4	是否 ReadOnly
wdMaskItemCount	5	for list item count
wdMaskColumnCount	6	for list ctrl column count
wdMaskCtrlCount	7	dialog ctrl count. ctrlId
0		
wdMaskCtrlHwndAt	8	获取 hWnd. 遍历控件: for(at=getInfo(0,wdMaskCtrlCount); --at>=0;) hWnd=getInfo(at,wdMaskCtrlHwndAt);
wdMaskLeft	9	获取控件矩形的左边界
wdMaskTop	10	获取控件矩形的顶边界
wdMaskWidth	11	获取控件矩形的宽
wdMaskHeight	12	获取控件矩形的高
wdMaskHwnd	13	获取控件句柄: m_hWnd= getInfo(ctrlId,wdMaskHwnd)
wdMaskCtrlId	14	获取控件 id: ctrlId=getInfo(hWnd,wdMaskCtrlId)

## 添加控件时的扩展式样 (与 MFC 同)

## Window Styles

WS_BORDER	0x00800000	
WS_VSCROLL	0x00200000	
WS_HSCROLL	0x00100000	
WS_THICKFRAME	0x00040000	
WS_GROUP	0x00020000	
WS_TABSTOP	0x00010000	

## Static Control Constants

SS_LEFT	0x0000	
SS_CENTER	0x0001	
SS_RIGHT	0x0002	
SS_ICON	0x0003	
SS_BLACKRECT	0x0004	
SS_GRAYRECT	0x0005	
SS_WHITERECT	0x0006	
SS_BLACKFRAME	0x0007	
SS_GRAYFRAME	0x0008	
SS_WHITEFRAME	0x0009	
SS_SIMPLE	0x000B	
SS_LEFTNOWORDWRAP	0x000C	
SS_ETCHEDHORZ	0x0010	
SS_ETCHEDVERT	0x0011	
SS_ETCHEDFRAME	0x0012	
SS_NOPREFIX	0x0080	
SS_NOTIFY	0x0100	
SS_RIGHTJUST	0x0400	
SS_SUNKEN	0x1000	

## Edit Control Styles

ES_LEFT	0x0000	
ES_CENTER	0x0001	
ES_RIGHT	0x0002	
ES_MULTILINE	0x0004	
ES_UPPERCASE	0x0008	
ES_LOWERCASE	0x0010	
ES_PASSWORD	0x0020	
ES_AUTOVSCROLL	0x0040	
ES_AUTOHSCROLL	0x0080	
ES_NOHIDESEL	0x0100	
ES_READONLY	0x0800	
ES_WANTRETURN	0x1000	
ES_NUMBER	0x2000	

## Button Control Styles

BS_DEFPUSHBUTTON	0x0001	
BS_AUTOCHECKBOX	0x0003	
BS_3STATE	0x0005	
BS_AUTO3STATE	0x0006	
BS_GROUPBOX	0x0007	
BS_AUTORADIOBUTTON	0x0009	
BS_LEFTTEXT	0x0020	
BS_LEFT	0x0100	
BS_RIGHT	0x0200	
BS_CENTER	0x0300	
BS_TOP	0x0400	
BS_BOTTOM	0x0800	
BS_VCENTER	0x0C00	
BS_PUSHLIKE	0x1000	
BS_MULTILINE	0x2000	
BS_NOTIFY	0x4000	
BS_FLAT	0x8000	

## Listbox Styles

LBS_SORT	0x0002	
LBS_MULTIPLESEL	0x0008	
LBS_USETABSTOPS	0x0080	
LBS_NOINTEGRALHEIGHT	0x0100	
LBS_MULTICOLUMN	0x0200	
LBS_WANTKEYBOARDINP UT	0x0400	
LBS_DISABLENOSCROLL	0x1000	
LBS_NOSEL	0x4000	

## Combo Box styles

CBS_SIMPLE	0x0001	
CBS_DROPDOWN	0x0002	
CBS_DROPDOWNLIST	0x0003	
CBS_AUTOHSCROLL	0x0040	
CBS_SORT	0x0100	
CBS_HASSTRINGS	0x0200	
CBS_NOINTEGRALHEIGHT	0x0400	
CBS_DISABLENOSCROLL	0x0800	
CBS_UPPERCASE	0x2000	
CBS_LOWERCASE	0x4000	

## ListView styles

LVS_SINGLESEL	0x0004	
LVS_SHOWSELALWAYS	0x0008	
LVS_SORTASCENDING	0x0010	
LVS_SORTDESCENDING	0x0020	
LVS_NOLABELWRAP	0x0080	
LVS_AUTOARRANGE	0x0100	
LVS_EDITLABELS	0x0200	
LVS_NOSCROLL	0x2000	
LVS_ALIGNTOP	0x0000	
LVS_ALIGNLEFT	0x0800	
LVS_NOCOLUMNHEADER	0x4000	
LVS_NOSORTHEADER	0x8000	

## 列表控件扩展式样

LVS_EX_GRIDLINES	0x0001	
LVS_EX_CHECKBOXES	0x0004	
LVS_EX_FULLROWSELECT	0x0020	
LVS_EX_INFOTIP	0x0400	

## 列表控件 listCtrl 的列对齐

LVCFMT_LEFT	0	
LVCFMT_RIGHT	1	
LVCFMT_CENTER	2	

## 附录 2：脚本错误代码

### 1、编译错误

#### 1000-1099 文件错误

- 1000: 非法的文件名
- 1001: 文件名不存在

#### 1100-1105 词法分析错误

- 1100: 字符串常量语法分析错误——没有找到相应的”符号
- 1101: 注释语法分析错误——没有找到相应的\*/符号
- 1102: 无法识别的字符
- 1103: 8 进制数语法分析错误
- 1104: 16 进制数语法分析错误
- 1105: 浮点数语法分析错误

#### 1106-1199 宏错误

- 1106: 非法的#
- 1107: 非法的 define 类语句，找不到 define 后的字符串
- 1108: define 类应独占一行
- 1109: 无法找到#else 对应的#if
- 1110: 无法找到#endif 对应的#if
- 1111: 结尾还有未处理的#if
- 1112: 代码过长，可能由于 include 的无限循环导致
- 1113: 非法的 include 格式，找不到对应的<>或者\"
- 1114: undef 语句错误，undef 的对象以前没有定义过
- 1115: 宏展开错误，参数个数不正确
- 1116: define 语句长度过长，可能由于 define 无限循环导致
- 1117: define 语句中不能定义#符号
- 1118: define 语句错误，对象重复定义

#### 1200-1299 括号匹配错误

- 1200: 括号匹配错误——无法匹配的括号
- 1201: 括号匹配错误——结尾缺少}

1202: 括号匹配错误——结尾缺少)或者]

## 1300-1399 语句分析错误

1300: 函数定义错误——找不到正确的函数返回类型

1301: 函数定义错误——找不到函数名

1302: 函数定义错误——函数名与系统函数名冲突

1303: 函数定义错误——函数名与前面定义的函数名冲突

1304: 函数定义错误——函数参数定义错误, 找不到匹配的(和)

1305: 函数定义错误——函数参数定义错误, 找不到参数名

1306: 函数定义错误——函数参数定义错误, 找不正确的参数类型

1307: 函数定义错误——main 函数必须是无参数函数

1308: 函数定义错误——函数不能嵌套定义

1311: return 语句错误——函数的返回类型应为 void, return 后不应有参数

1313: return 语句错误——函数的返回类型不为 void, return 后应有参数

1320: break 语句错误——break 命令后不应有参数

1321: break 语句错误——仍有无法处理的 break 语句

1330: continue 语句错误——continue 命令后不应有参数

1331: continue 语句错误——仍有无法处理的 continue 语句

1340: 变量定义错误——无法识别的数据类型

1341: 变量定义错误——找不到变量名

1342: 变量定义错误——变量名与保留字冲突

1343: 变量定义错误——变量名重复定义

1344: 变量定义错误——变量名间需用,分隔

1350: for 语句错误——for 结构后缺少语句

1351: for 语句错误——找不到相应的(

1353: for 语句错误——找不到相应的}

1355: switch 语句错误——switch 语句结果不全

1356: switch 语句错误——case 语句错误

1357: switch 语句错误——default 语句错误

1360: if 语句错误——if 后缺少语句

1361: if 语句错误——找不到相应的(

1362: if 语句错误——(内部错误)找不到相应的}

1363: else 语句错误——找不到相应的 if 语句

1364: else 语句错误——(内部错误)找不到相应的}

1370: while 语句错误——while 后缺少语句

1371: while 语句错误——找不到相应的(

1372: while 语句错误——(内部错误)找不到相应的}

1390: 找不到运行起点，没有找到 main 函数

## **1400-1499 表达式分析错误**

1410: 赋值语句为空

1411: 函数调用时参数个数错误

1414: ?: 分析错误

1420: 无法分析的表达式

## 2、运行错误：

### 10000-10099 内部错误：

- 10000: Internal Error: 非法 CS 值
- 10001: Internal Error: 无法识别的语句

### 10100-10199 溢出错误：

- 10100: 程序运行空间溢出
- 10101: 栈溢出
- 10102: 堆溢出

### 10200-10299 类型匹配错误

- 10200: 赋值类型不符
- 10201: 判断条件语句取值失败
- 10202: 常量赋值类型不符

### 10300-10399 函数调用错误

- 10300: 函数调用错误(类型不符)
- 10301: 找不到相应的成员函数
- 10302: 数组[]操作错误, []只适用于 array 类型
- 10303: 函数调用错误, 参数个数不符
- 10304: 函数调用错误, 参数类型不符
- 10305: 除数为 0
- 10306: Walk 函数调用错误
- 10307: chr()函数参数应在[0-255]之间
- 10308: 数组下标越界
- 10309: 赋值语句类型不符
- 10310: 字符串下标越界

### 3、系统限制

程序中 include 的层数不能超过 64;

代码文件长度不能超过=1024KB=1MB

程序使用的数据变量个数不能超过 1048576 个（数组元素个数 $\leq$ 1,000,000）

程序使用（全局变量）堆空间不能超过 65536;

程序使用（递归和函数参数）栈空间不能超过 65536;

## 附录 3：开发工具

WalkLan 语言解释器，定义了自身的语法与结构，提供了基本的系统函数以及针对系统特有对象而编写的较为完整的分类函数，从而为用户进行二次开发提供了强有力的支持。

同时，二次开发也具有一定的方法和技巧，尤其是在编写的过程中，常常会因被一些小错误困扰，误时误力而迟迟不得其要领，讲究开发的方法与技巧，也就非常地有必要而且有效果了。下面，根据解释器在开发与调试时的经验总结，将会对二次开发编写可能存在的错误与问题，编辑器的使用进行阐述与示例。同时，也对该解释器中凸显 walk 产品优势的内容进一步的分析与应用举例，以供读者及开发人员参考。

## 附录 4：脚本语言的特点

与 c/c++语言语法相同，简单易学。

对话框交互能力强。

提供了简洁的数据库 SQL 操纵和执行环境。

扩展到 Walk 平台环境，具有强大的 GIS 数据处理能力。

### 1、WalkLan 和 c 语言代码对比

#### 1) 计算 pi

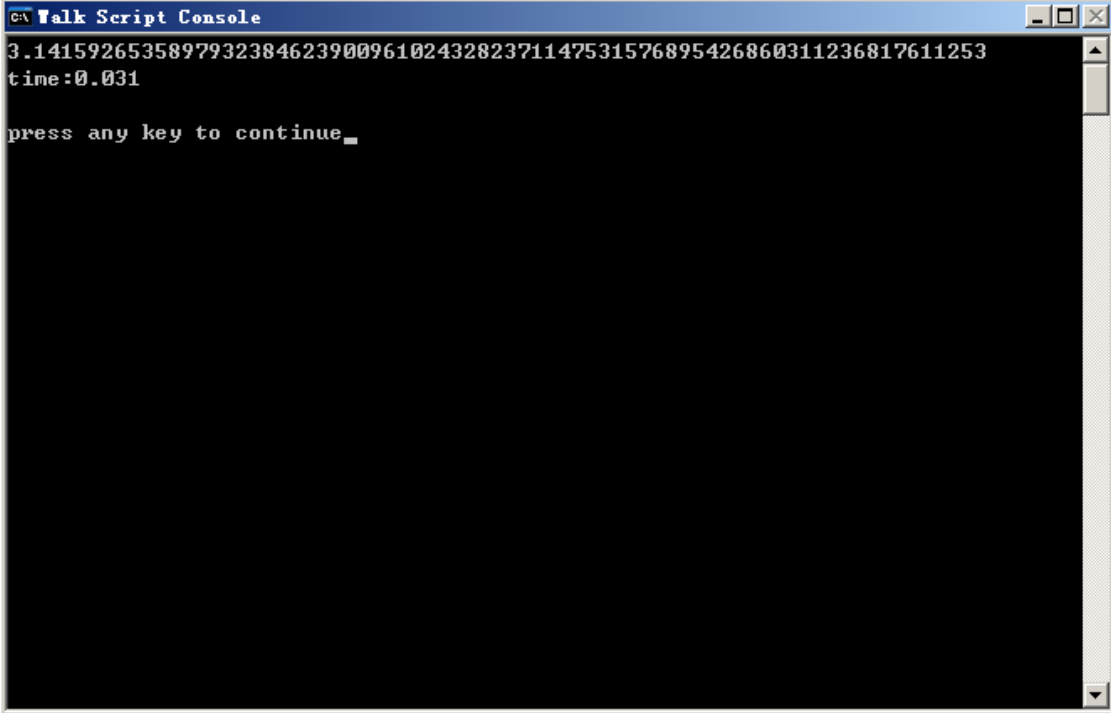
以计算 pi 为例，计算公式： $PI/2 = 1 + 1!/3!! + 2!/5!! + \dots + i!/(2i+1)!! + \dots$

// WalkLan for pi	//c for pi
	#include "stdio.h"
	#include "time.h"
#define N 72 // the size of digits.	#define N 72 // the size of digits.
void main()	void main()
{	{
array a; a.setSize(N+1);	int a[N+1];
array result; result.setSize(N+1);	int result[N+1];
int begin_t=clock();	int begin_t=clock();
// initialize the buffer (for i!/(2i+1)!!)	// initialize the buffer (for i!/(2i+1)!!)
for (int i = 0; i <= N; i++)	for (int i = 0; i <= N; i++)
a.setAt(i,2)=2;	a[i]=2;
// now we calculate. Please wait..	// now we calculate. Please wait...
for (int j=N, last=0, q=0; j > 0; j--)	for (int j=N, last=0, q=0; j > 0; j--)
{	{
int digit = 0;	int digit = 0;
for (i = j; i > 0; i--)	for (i = j; i > 0; i--)
{	{
digit = digit * i + a[i] * 10;	digit = digit * i + a[i] * 10;
int k = i + i - 1;	int k = i + i - 1;
a[i] = digit % k;	a[i] = digit % k;
digit = digit / k;	digit = digit / k;
}	}
result[q++] = (last + digit/10) % 10;	result[q++] = (last + digit/10) % 10;
last = digit % 10;	last = digit % 10;
}	}
double t= (clock() - begin_t)/1000.0;	double t= (clock() - begin_t)/1000.0;
// now output	// now output
printf("%d.", result[0]);	printf("%d.", result[0]);

<code>for (i = 1; i &lt; N; i++)</code>	<code>for (i=1; i&lt;N; i++)</code>
<code>printf("%d", result[i]);</code>	<code>printf("%d", result[i]);</code>
<code>printf("\ntime:%.3lfs\n", t);</code>	<code>printf("\ntime:%.3lfs\n", t);</code>
<code>}</code>	<code>}</code>

对比可见，WalkLan 与 c 语言在数组定义上有细微的差异，而运算符和程序结构上完全相同。c 语言常用于编写算法，许多精妙的算法代码可直接移植到 WalkLan 上，当然，因为脚本是解释性语言，运行效率远低于由 c 编译的执行程序。

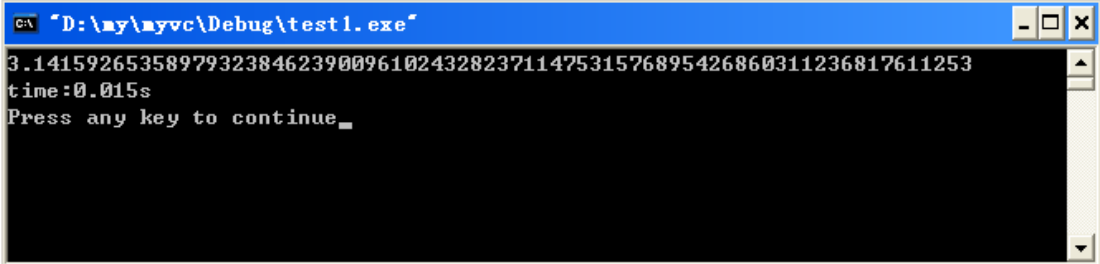
脚本运行结果



```
3.14159265358979323846239009610243282371147531576895426860311236817611253
time:0.031

press any key to continue_
```

c 语言运行结果



```
3.14159265358979323846239009610243282371147531576895426860311236817611253
time:0.015s
Press any key to continue_
```

## 2) 典型的递归用法——枚举目录下文件

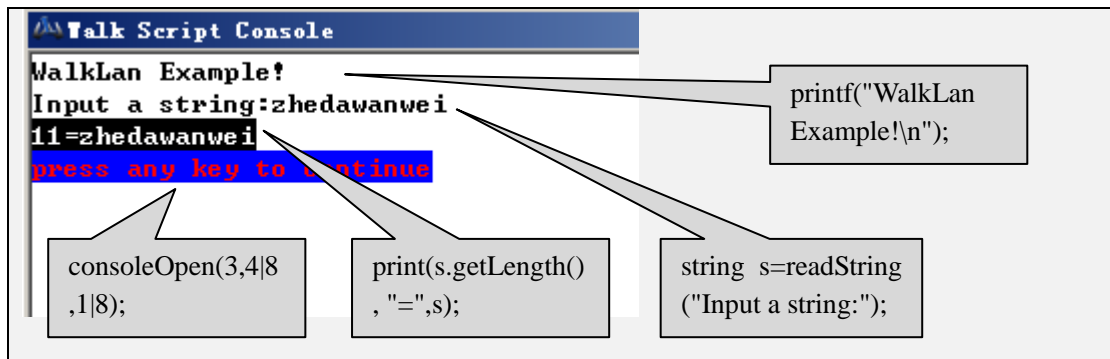
```
#include "wcToolConstant.h"
void FindFileWithinPath(string path)
{
    wkFile finder;
    path.trimRight("\\"); path += "\\*. *";
    bool suc = finder.findFile(path);
    while (suc)
    {
        suc = finder.findNextFile();
        // 跳过 . 和 .. 目录
        if (finder.findFileIs(IsDots))
            continue;
        // 对子目录递归
        if (finder.findFileIs(IsDirectory))
            findFileWithinPath(finder.findFileGet(wcFilePath));
        else // 在脚本编辑器中 trace 输出到结果框
            printf(finder.findFileGet(wcFilePath)+"\n");
    }
    finder.findClose();
}
void main()
{
    //列出执行程序下 WalkLan 中的所有文件
    findFileWithinPath(WsGetAppFolder()+"WalkLan");
}
```

其中 WsGetAppFolder() 获取执行程序所在目录。

## 2、Console 控制台

```
void main()
{
    consoleOpen(3,0,15);//打开 console, 字符=黑色, 背景=白色
    consoleOpen(4);//按颜色设置刷新 console
    printf("WalkLan Example!\n");
    string s = readString("Input a string:");//从 console 读入一个字符串
    consoleOpen(3,15,0);//改变颜色: 字符=白色, 背景=黑色
    print(s.getLength(),"=",s);
    consoleOpen(3,4|8,1|8);//改变颜色: 字符=红色, 背景=蓝色
}

运行结果
```



### 3、脚本对话框

#### 1) 与 MFC 对话框的关系

WalkLan 对话框创建形式与 Unix 的 Motif 类似，通过代码直接设置和创建，而在 MS-Visual Studio 中提供了可视化的 MFC 对话框创建环境。但 WalkLan 对话框中控件风格和性能定义与 MFC 基本相同，都支持 static text、edit、button、check、radio、list box、combo box、list control (report) 等常用控件。在 wcDialogConstant.h 对话框控件风格常量的定义采用了与 MFC 完全相同的宏，能力超强的程序员可以自己编写一个程序将 MFC 的.rc 对话框转换为 WalkLan 对话框，甚至可以将 MFC 的对话框控件响应函数转换到脚本中。

在 WalkLan 中脚本对话框都是无模态的，支持与 Walk 图形窗口的鼠标操作进行交互。脚本对话框创建、控件响应和图形窗口交互一般都在一个脚本中完成，最简单的对话框例子：

1	<code>#include "wcDialogConstant.h"</code>
2	<code>void CreateDlg(wkDialog &amp;dlg)</code>
3	<code>{</code>
4	<code>    dlg.create("My Dialog");</code>
5	<code>    dlg.addCtrl(100, wdCtrlEdit, 10, 10,128,12,0);</code>
6	<code>    dlg.show();</code>
7	<code>}</code>
8	<code>void main()</code>
9	<code>{</code>
10	<code>    wkDialog dlg;</code>
11	<code>    CreateDlg(dlg);</code>
12	<code>    dlg.setAction(wdIdOk, wdMsgBnClicked);</code>
13	<code>    for (int ctrlId,act=0; act!=-1; act=dlg.peekAction(ctrlId))</code> <code>    {</code>
14	<code>        if (act==wdMsgBnClicked &amp;&amp; ctrlId==wdIdOk)</code>
15	<code>            message(dlg.getText(100));</code>
16	<code>    }</code>
17	<code>}</code>

行 1，包含头文件，其中定义了与对话框相关的宏

行 2~7，创建对话框 create、添加对话框控件 addCtrl、显示对话框 show

行 12，设置控件响应消息 setAction

行 13，循环等待消息 `act=peekAction`，若 `act==-1`，则对话框被关闭

行 14~15，消息处理，显示编辑框中的字符串

Walk 脚本对话框总是自动包含【确认】和【取消】按钮，其 `id` 分别为 `wdIdOk` 和 `wdIdCancel`。用户可通过设置消息响应接管按钮压下时要处理的事情，否则系统将自动处理（关闭对话框）。



## 2) 一个加法测试器对话框例子

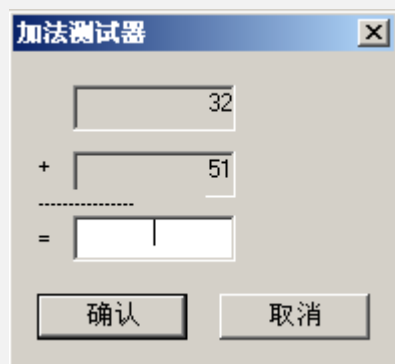
```
#include "wcDialogConstant.h"
//产生随机算例
void FillDigit(wkDialog &dlg, int base)
{
    dlg.setText(101, toString(abs(random())%base));
    dlg.setText(102, toString(abs(random())%base));
    dlg.setText(100, ""); //清空
    dlg.show(100,4); //设置焦点
}
//创建对话框
void CreateDlg(wkDialog &dlg)
{
    int x=20, y=10, w=54, h=14;
    dlg.create("加法测试器");
    dlg.addCtrl(101, wdCtrlEdit, x, y, w, h, ES_READONLY|ES_RIGHT);
    y+=20; dlg.addCtrl("+", 8, y);
    dlg.addCtrl(102, wdCtrlEdit, x, y, w, h, ES_READONLY|ES_RIGHT);
    dlg.addCtrl("-----", 8, y+12);
    y+=20; dlg.addCtrl("=", 8, y+2);
    dlg.addCtrl(100, wdCtrlEdit, x, y, w, h, ES_CENTER);
    dlg.show(); //显示对话框
}
void main()
{
    wkDialog dlg;
    CreateDlg(dlg);
    randomize();
    FillDigit(dlg, 100);
    dlg.setAction(wdIdOk, wdMsgBnClicked); //要求响应“确认”键
    for (int ctrlId, act=0; act!=-1; act=dlg.peekAction(ctrlId))
```

```

{
    //循环到关闭对话框(act==1)
    if (act==wdMsgBnClicked && ctrlId==wdIdOk)
    {
        //响应"确认"消息
        bool right=(atoi(dlg.getText(100)) == atoi(dlg.getText(101)) +
        atoi(dlg.getText(102)));
        //判断正确与否
        message(right? "正确":"错误");
        FillDigit(dlg, 100);
    }
}
}

```

结果:



创建的对话框(加法测试器)

### 3) 显示目录下的图片

```

#include "wcToolConstant.h"
#include "wcDialogConstant.h"
#define _PicList 101 //图片名列表
#define _PicFrame 200 //图片显示框
int gzItem=0;
// 列出 sPath 目录下(不含子目录)的文件
void FindFileWithinPath(string path, wkDialog& dlg)
{
    wkFile finder;
    path.trimRight("\\"); path += "\\*.jpg";
    bool suc = finder.findFile(path);
    while (suc)
    {
        suc = finder.findNextFile();
        if (!finder.findFileIs(IsDots) && !finder.findFileIs(IsDirectory))
            dlg.insertItem(_PicList, gzItem++, finder.findFileGet(wcFileTitle));
    }
}

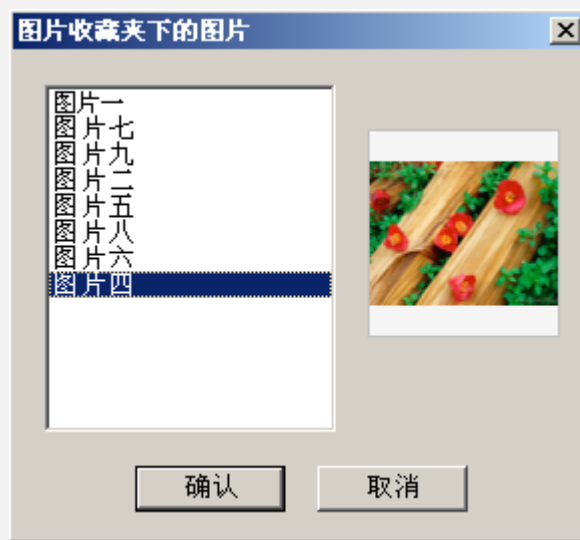
```

```

    }
    finder.findClose();
}
//创建对话框
void CreateDlg(wkDialog &dlg)
{
    int x=10, y=10, w=98, h=128;
    dlg.create("图片收藏夹里面的文件");
    dlg.addCtrl(_PicList, wdCtrlListBox, x, y, w, h, 0);
    dlg.addCtrl(_PicFrame, wdCtrlText, 118, 24, 64, 64, SS_ETCHEDFRAME);
    dlg.show(); //显示对话框
}
void main()
{
    wkDialog dlg;
    CreateDlg(dlg);
    string gzPath= "C:\\Documents and Settings\\huangff\\My Documents\\My Pictures\\";
    FindFileWithinPath(gzPath, dlg);
    dlg.setAction(_PicList, wdMsgSelchange); //设置响应列表选中项改变消息
    //循环, 直到关闭对话框(act==-1)
    for (int ctrlId, act=0; act!=-1; act=dlg.peekAction(ctrlId))
        if (act==wdMsgSelchange && ctrlId==_PicList)
            dlg.setPic(_PicFrame, gzPath+dlg.getText(_PicList, dlg.getCurSel(_PicList))+
".jpg");
}

```

结果:



创建的对话框(图片收藏夹下的图片)

## 4、使用脚本响应层编辑和数据存取

介绍使用库中脚本进行地物编辑约束和数据存取控制的方法。因为脚本在库中, 因此要求脚本不得有错误。脚本响应层的编辑和数据存取事件就要求脚本的代码运行效率要高, 所

以响应脚本应由有经验的高级程序员来编写。

## 1) 库层脚本登记表结构

与层相关的约束脚本采用数据库中存储的脚本登记表中的脚本。

### 1.1 层脚本控制登记表：

constrainScriptReg

层脚本控制登记表结构：

layerName varchar(32) 层名

ctrlType int 控制类型

ctrlContext int 控制状态（控制内容）

scriptId int 脚本在脚本表的 ID

constrainScript : 表			
	sid	stype	script
	1	0	wk_层数据加载存盘控制.cpp
	5	0	wk_地物编辑控制.cpp
▶	(自动编号)	0	

### 1.2 脚本表：

constrainScript

脚本结构：

sid autoIncrease （自增）ID

stype int 脚本类型 1-- script 为脚本，0-- script 为脚本文件名

script text 脚本或文件名

constrainScriptReg : 表				
	layerName	ctrlType	ctrlContext	scriptId
	wk_JS	1	7	5
	wk_XZTB	1	7	5
	wk_JS	2	4	1
	wk_XZTB	2	4	1
	wk_JS	3	1	1
	wk_XZTB	3	1	1
	wk_JS	4	1	1
	wk_XZTB	4	1	1
▶		0	0	0

### 1.3 控制类型 ctrlType:

- 1、地物、文字编辑
- 2、层数据加载
- 3、层保存前
- 4、层保存后

#### 一、地物编辑控制

控制类型：1

控制内容：0-不控制、

地物：1-增、2-删、4-改；

文字：8-增、16-删、32-改

脚本 main 形式：

void main(array args)

其中： args[0]-- ctrl type: "1"

args[1]-- layer handle

args[2]-- event: 地物： 1-增、 2-删、 3-改、4-改几何、5-改属性；

文字： 11-增、 12-删、 13-改

#### 二、层数据加载控制

控制类型：2

控制内容：0-不控制、1-初始化；2-加载式样；4-加载地物；8-加载文字；16-结束处理

脚本 main 形式：

void main(array args)

其中： args[0]-- ctrl type: "2"

args[1]-- layer handle

args[2]-- event: 1-初始化；2-加载式样；3-加载地物；4-加载文字；5-结束

#### 三、层数据保存前

控制类型：3

控制内容：0-不控制、1-控制

脚本 main 形式：

void main(array args)

其中： args[0]-- ctrl type: "3"

args[1]-- layer handle

args[2]-- event: 1-初始化；2-保存式样；3-保存地物；4-保存文字

#### 四、层数据保存后

控制类型：4

控制内容：0-不控制、1-控制

脚本 main 形式：

void main(array args)

其中： args[0]-- ctrl type: "4"

```
args[1]-- layer handle  
args[2]-- "1" layer save ok!. "0"-- layer save fail!
```

## 2) 脚本函数

wkGeoset 的成员函数 onConstrainGet 和 onConstrainSet 只能用在层的地物编辑和数据存取响应脚本中。

### 2.1 Geoset.onConstrainGet

```
//----获取当前在控制的层  
wkLayer onConstrainGetLayer();  
  
//----获取当前在编(删改)的原地物 ID 或文字 ID  
int onConstrainGetId();  
  
//----获取当前在编的新地物几何(可直接修改, 不要 free)  
wkGeometry onConstrainGetGeom();  
  
//----获取当前在编的新地物属性  
string onConstrainGetAttr(string fieldName);  
  
//----获取当前在编的文字(可直接修改, 不要 free)  
wkAnnotation onConstrainGetAnno();  
  
//----获取层加载时的过滤条件  
string onConstrainGetLoadFilter();
```

### 2.2 Geoset.onConstrainSet

```
//----放弃编辑, 或层数据存取  
void onConstrainSetAbort();  
  
//----设置当前在编的新地物属性  
void onConstrainSetAttr(string fieldName, string newVal);  
  
//----设置层加载时的过滤条件  
void onConstrainSetLoadFilter(string loadFilter);
```

## 2.3 响应脚本例子

```

#ifdef _VS_WALK
#include "wkClassDefine.h"
#endif

wkGeoset geoset;
void main(array args)
{
    if (args.getSize() != 3)
    {
        message("不是【层数据编辑和数据存取响应脚本】");
        return;
    }
    string s;

    int ctrlType = atoi(args[0]); // 控制类型
    wkLayer layer = geoset.onConstrainGetLayer(); // 当前工作层
    int ctrlContext = atoi(args[2]); // 控制内容

    if (1 == ctrlType)
        s = "层编辑";
    else if (2 == ctrlType)
        s = "层加载";
    else if (3 == ctrlType)
        s = "层保存前";
    else if (4 == ctrlType)
        s = "层保存后";
    else
    {
        message("控制类型错误");
        return;
    }

    s += "\n 工作层名=" + layer.getName();
    s += "\n 当前的控制内容为=" + toString(ctrlContext);

    if (1 == ctrlType)
    {
        if (1 == ctrlContext)
            s += "\n 新增地物";
        else if (2 == ctrlContext)
            s += "\n 删除地物";
        else if (3 == ctrlContext)
            s += "\n 修改地物";
    }
}

```

```

else if (4==ctrlContext)
    s+="\n 修改地物几何";
else if (5==ctrlContext)
    s+="\n 修改地物属性";
else if (11==ctrlContext)
    s+="\n 新增文字";
else if (12==ctrlContext)
    s+="\n 删除文字";
else if (13==ctrlContext)
    s+="\n 修改文字";
if (ctrlContext<10)
{
    int feaId=geoset.onConstrainGetId();
    wkGeometry newGeom=geoset.onConstrainGetGeom();
    string fieldName="featureName";
    string property=geoset.onConstrainGetAttr(fieldName);
    if (1!=ctrlContext)//非新增地物
        s+="\n 原地物 ID="+toString(feaId);
    if (2!=ctrlContext)//非删除地物
    {
        s+="\n 新地物几何类型="+toString(newGeom.type());
        //可直接修改 newGeom 几何体
        s+="\n 新地物的"+fieldName+"的属性值="+property;
    }
    if (1==ctrlContext)
    {
        wkView view;
        if (view.dlgQuestion(s+"\n\n 是否放弃新增?"))
            geoset.onConstrainSetAbort();
        return;
    }
}
else
{
    int annId=geoset.onConstrainGetId();
    wkAnnotation newAnn=geoset.onConstrainGetAnno();
    if (1!=ctrlContext)//非新增文字
        s+="\n 原文字 ID="+toString(annId);
    if (2!=ctrlContext)//非删除文字
        s+="\n 新文字="+newAnn.getText();
    //可直接修改 newAnn 文字
}
}
message(s);
}

```

## 附录 5 样例及说明

### 1) 几何体

#### 【example】

```
void main()
{
    wkGeoset geoset;
    wkGeometry geom;
    wkDb db;
    db.open("E:\\ walk\\walkfld\\hff0003.mdb",true,true);
    wkLayer layer=geoset.addLayer("实测点层",db);
    layer.empty();
    wkBox box;

    array arr1= CreatePointArray();
    array arr2,arr3,arr4;
    for(int i=0;i<4;i++)
    {
        arr2.add(arr1[i]);
        box.add(arr1[i]);
    }

    for(int j=5;j<8;j++)
        arr3.add(arr1[j]);
    for(int k=8;k<13;k++)
        arr4.add(arr1[k]);

    CreateGeometry(geom,layer,arr1[4],arr1[14],box,arr3,arr4);
    layer.addFeature(arr1[13]);

    int n1=layer.getFeatureCount();
    array featurearry;
    array geometryarry;

    array polygonarryn;
    for(int l=0;l<n1;l++)
        featurearry.add(layer.getFeatureAt(l));
    for(int m=0;m<n1;m++)
        geometryarry.add(featurearry[m].getGeometry());
    for(int n=0;n<n1;n++)
    {
        polygonarryn.add(geometryarry[n].getPolygonCount());
        trace("%d\\n",polygonarryn[n]);
    }
}
```

```

}
wkPolygon poly1=geometryarry[2].getPolygonAt(0);
wkPolygon poly2=geometryarry[3].getPolygonAt(0);
wkPolygon poly3=geometryarry[4].getPolygonAt(0);
geometryarry[2].insertPolygonAt(poly2);
geometryarry[2].deletePolygonAt(1);
geometryarry[2].replacePolygonAt(poly2, 0);
geometryarry[2].insertPolygonAt(poly3);

int n2=layer.getFeatureCount();
array featurearry1;
array geometryarry1;
array polygonarryn1;

int n3=geometryarry[2].getPartsCount();//得到第三个几何体里面的 Parts 的个数
wkParts parts1=geometryarry[2].getPartsAt(0);
wkParts parts2=geometryarry[3].getPartsAt(0);
wkParts parts3=geometryarry[4].getPartsAt(0);
geometryarry[2].insertPartsAt(parts2,true,2);
geometryarry[2].deletePartsAt(1);
geometryarry[2].replacePartsAt(parts3,1);

int n4=geometryarry[0].getPointCount();//得到三个几何体里面点的个数
int n5=geometryarry[1].getPointCount();
int n6=geometryarry[5].getPointCount();
wkPoint point1=geometryarry[0].getPointAt(0);
wkPoint point2=geometryarry[1].getPointAt(0);
wkPoint point3=geometryarry[5].getPointAt(0);
geometryarry[0].insertPointAt(point2,1);
geometryarry[0].deletePointAt(1);
geometryarry[1].replacePointAt(point3, 1);
int n7=geometryarry[0].getPointCount();//得到三个几何体里面点的个数
int n8=geometryarry[1].getPointCount();
int n9=geometryarry[5].getPointCount();
int n10=geometryarry[2].get9i(geometryarry[3]);
int n11=geometryarry[2].get9i(geometryarry[3],1,1,1);
wkGeometry geom1=geometryarry[2].boolOp(geometryarry[3],0);//求两个几何体的和
//将本几何体擦除 B 几何体的结果
wkGeometry geom2=geom1.trim(geometryarry[2],true);/
double d1=geometryarry[2].area();
double d2=geometryarry[2].perimeter();
wkPoint point4=geometryarry[3].centroid();
wkBox b1=geometryarry[2].bounds();
int n12=geom1.handle();
int n13=geom1.type();

trace("%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",n1,n2,n3,n4,n5,n6,n7,n8,n9,n

```

```

    10,n11,n12,n13);
    wkGeometry geom3=geometryarry[2].convex();
    wkGeometry geom4=geometryarry[2].convex(arr4);
    wkPoint point5;
    bool b2=geometryarry[3].isNormal(0.001,1, point5);
    trace("%s,%s\n",toString(b1),toString(b2));

    double d3=geometryarry[2].distance(geometryarry[5]);
    double d4=geometryarry[2].distance(point1);

    trace("%f,%f,%f,%f\n,%s\n,%s\n,%s\n,%s\n,%s\n",d1,d2,d3,d4,toString(point1),toString(p
    oint2),toString(point3),toString(point4),toString(point5));
    geometryarry[3].transform(1, 1, 1, 1,1,1);
    geometryarry[2].transform(1, 1);
    geometryarry[2].putInOrder();
    geometryarry[2].putInOtherOrder();
    geometryarry[2].rebuild();
    geometryarry[2].adjByArea(2000, arr4);
}

//创建数组及点
array CreatePointArray()
{
    array pts1;
    wkPoint pt1,pt2,pt3,pt4,pt5,pt6,pt7,pt8,pt9,pt10,pt11,pt12,pt13,pt14,pt15;
    pt1.x=0;
    pt1.y=0;
    pt2.x=0;
    pt2.y=50;
    pt3.x=50;
    pt3.y=50;
    pt4.x=50;
    pt4.y=0;
    pt5.x=25;
    pt5.y=25;
    pt6.x=10;
    pt6.y=10;
    pt7.x=10;
    pt7.y=20;
    pt8.x=100;
    pt8.y=40;
    pt9.x=55;
    pt9.y=50;
    pt10.x=60;
    pt10.y=30;
    pt11.x=80;

```

```
pt11.y=20;
pt12.x=100;
pt12.y=40;
pt13.x=70;
pt13.y=80;
pt14.x=65;
pt14.y=50;
pt15.x=100;
pt15.y=100;

pts1.add(pt1);
pts1.add(pt2);
pts1.add(pt3);
pts1.add(pt4);

pts1.add(pt5);

pts1.add(pt6);
pts1.add(pt7);
pts1.add(pt8);

pts1.add(pt9);
pts1.add(pt10);
pts1.add(pt11);
pts1.add(pt12);
pts1.add(pt13);

pts1.add(pt14);
pts1.add(pt15);

return pts1
}

//创建几何体
void CreateGeometry(wkGeometry geom,wkLayer layer,wkPoint pt1,wkPoint pt2,wkBox
box,array pts1,array pts2)
{
    geom.create( pt1);
    layer.addFeature(geom);
    geom.free();
    geom.create(pt2,30);
    layer.addFeature(geom);
    geom.free();

    geom.create(box,true);
    layer.addFeature(geom);
    geom.free();
}
```

```
geom.create( pts1,1,true);  
layer.addFeature(geom);  
geom.free();  
  
geom.create( pts2,1,true);  
layer.addFeature(geom);  
geom.free();  
}
```

## 2) 线素整理和拓扑构面

### 【example】

```
void main()
{
    wkGeoset geoset;
    wkDb db;
    db.open("E:\\ 实习专用文件夹（黄菲菲） \\walk\\walkfld（培训使用）
\\hff0003.mdb",true,true);
    wkLayer layer=geoset.addLayer("实测点层",db);
    layer.empty();
    wkClean clean;
    wkBox range;
    range.add(0,0);
    range.add(50,50);
    clean.create(range);
    wkPoint pt1,pt2,pt3,pt4,pt5,pt6;
    pt1.x=0;
    pt1.y=10;
    pt2.x=30;
    pt2.y=10;
    pt3.x=19;
    pt3.y=7;
    pt4.x=20;
    pt4.y=30;
    pt5.x=15;
    pt5.y=5;
    pt6.x=25;
    pt6.y=12;
    wkParts parts;

    array pts1;
    pts1.add(pt1);
    pts1.add(pt2);
    parts.create(pts1,1);
    layer.addFeature(parts,true);
    parts.free();

    array pts2;
    pts2.add(pt3);
    pts2.add(pt4);
    parts.create(pts2,1);
    layer.addFeature(parts,true);
    parts.free();
}
```



### 3) 向层中添加地物、文字和影像

【example】:

```
void main()
{
    wkGeoset geoset;
    wkLayer layer;
    wkFeature feature;
    layer = geoset.getLayerAt(2);
    layer.setEditable(true);
    layer.empty();

    //构造地物
    wkPoint pt1;
    pt1.x=60;pt1.y=60;//点 p1

    wkPoint pt2;
    pt2.x=70;pt2.y=70;//点 p2

    wkBox bx;
    bx.add(80,80);
    bx.add(pt1);//矩形盒

    array arrpts1;
    wkPoint pt3;
    pt3.x=20;pt3.y=20;pt3.z=0;
    arrpts1.add(pt3);
    wkPoint pt4;
    pt4.x=50;pt4.y=20;pt4.z=0;
    arrpts1.add(pt4);
    wkPoint pt5;
    pt5.x=30;pt5.y=40;pt5.z=0;
    arrpts1.add(pt5);

    array arrpts2;
    wkPoint pt6;
    pt6.x=5;pt6.y=5;pt6.z=0;
    arrpts2.add(pt6);
    wkPoint pt7;
    pt7.x=20;pt7.y=5;pt7.z=0;
    arrpts2.add(pt7);
    wkPoint pt8;
    pt8.x=15;pt8.y=20;pt8.z=0;
    arrpts2.add(pt8);

    //构造多边形
```

```
wkPolygon polygon;
polygon.create(arrpts2,1);

array arrpts3;
wkPoint pt9;
pt9.x=0;pt9.y=0;pt9.z=0;
arrpts3.add(pt9);
wkPoint pt10;
pt10.x=50;pt10.y=0;pt10.z=0;
arrpts3.add(pt10);
wkPoint pt11;
pt11.x=50;pt11.y=50;pt11.z=0;
arrpts3.add(pt11);
wkPoint pt12;
pt12.x=0;pt12.y=50;pt12.z=0;
arrpts3.add(pt12);

//构造线串
wkPoints points;
points.create(arrpts3,1);

wkPoint pt13;
pt13.x=51;pt13.y=51;pt13.z=0;

//构造线流
wkParts parts1,parts2;
parts1.create(pt10,5,300,600,true);

//向层中添加地物
layer.addFeature(pt1);
layer.addFeature(pt2,30.00);
layer.addFeature(bx,true);
feature=layer.addFeature(arrpts1,1,true);
layer.addFeature(polygon);
layer.addFeature(parts1,true);
layer.addFeature(points,true);

//向层中添加文字式样
layer.addStyle(5,"式样一","");

//向层中添加文字和影像
layer.addAnnotation("文字一",pt13,32);
layer.addAnnotation("E:\\sx\\walk\\图片\\图片一.jpg",arrpts3);
}
```