开发者指南

介-绍

概念

类别

Teigha for .dwg文件有若干类别的C++类组成，它们由类名的前缀做区分。

OdRx，运行时类的注册和类别检查的类

OdDb，数据库类

OdGi，用来矢量化数据库对象的类

OdGe，普通的几何类

OdBr，用作边界表示遍历的类

OdGs，图形系统类，提供一个矢量化的客户端接口。

OdRxObject和运行时类型检查

OdRxObject是所有需要注册和运行时类型检查的类的基类，OdRxObject的每一个子类都有一个提供运行时类型检查的OdRxClass对象(可以由OdRxObject::dexc()函数得到)，它也提供确定一个对象是否是一个特定类、或者是其子类的实例(OdRxObject:;IsA()函数)。

ObjectIDs

Teigha for .dwg文件使用对象ID(OdDbObjectId的实例)来引用数据库对象。对象ID是数据库对象驻留内存的占位符，我们总可以从

一个有效的对象ID得到一个存在的数据库对象指针。一般地，用户可以通过使用一个对象ID明确地得到一个数据库对象指针。一个打开的数据库对象在其相应的智能指针跳出作用域、或者直接使用智能指针的release()函数时将被关闭。一个数据库对象除非在他们明确被打开时才会进驻内存。这个机制提供了一个透明的部分载入功能，一些关键的对象将会被载入内存，其余的数据库对象只有在被打开的时候才有必要载入内存，这个极大地提高了哪些只需要绘制一部分内容的应用程序的执行速度。

智能指针和引用计数

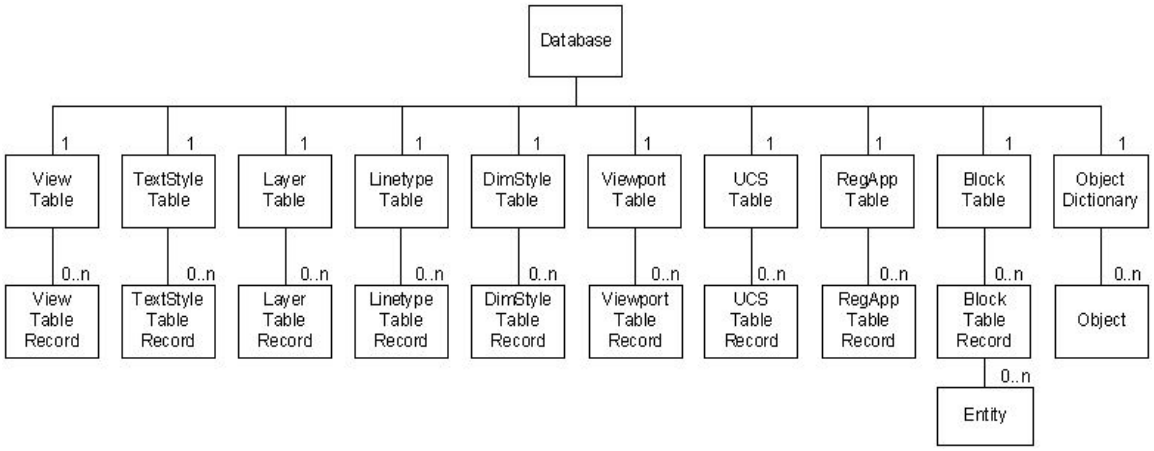
Teigha for .dwg文件使用引用计数来管理大多数堆对象，一组智能指针类提供为对象提供了便捷的使用，可以在顶级包含路径中的SmartPtr.h中找到主要的智能指针模板OdSmartPtr，

大多数数据库类都有为其定义的形式为类名加'Ptr'前缀的智能指针类，例如

OdDbCircle的智能指针是OdDbCriclePtr等等。

数据库结构

一个绘图文件在内存中由一个OdDbDatabase类的实例表示。本篇将一个OdDbDatabase实例简单视为一个"database"(以后成为"数据库"),一个数据库将作为所有从文件中载入的数据的容器。并包括以下数据：



一个数据库也包含一些函数用来设置和得到文件总系统变量的值。

实体和块 Entities and Blocks

我们通过先访问第一个引用了实体对象的OdDbBlockTableRecord，而后访问整个实体列表来遍历所有实体对象。这里有两个特殊的数据块，*Model_Space和*Paper_Space，他们分别管理模型空间和布局空间。一些实体对象还作为其他对象的容器，例如一个OdDb3dPolyline实体作为它的顶点的容器和所有者，一个OdDbBlockreference包含实体的属性，这个有别于C工具库中数据块中所有实体存放于一个平坦的列表中。

表 Tables

在一个.dwg或者.dxf文件中包含了以下表

图层 Layers

包含绘制的图层，每个图层中包含诸如开/关，冻结/解冻，颜色、线型之类的属性，相应的类是OdDbLayerTableRecord.

线型 LineStyles

包含了绘图线型，每个线型都有一个名字，一个描述其显示方式的字符描述，以及一个包含决定线型外观控制参数的数组，对应的类是OdDbLinetypeTableRecord。

文本风格 TextStyles

包含绘制文本的风格定义，每一个文本风格有一个名字，一个使用的字体的说明，高度，斜度和其他参数。对应的类是OdDbTextStyleTableRecord

视图 Views

包含绘制的命名视图定义，每一个视图定义指定了一个三维视图。对应的类是OdDbViewTableRecord。

视口 viewport

包含绘制视口的定义，这个不同于布局空间视图的实体，他们更像是显示模型空间数据的模型空间视口命名的视口组由一组视口组成，他们共享一个名字，当前视口或者其他视口的名字格式为“*ACTIVE”。对应的类是OdDbViewportTableRecord。

尺寸风格 DimStyle

包含了绘制维度风格的定义，维度风格是一些定制如何创建维度的变量的集合。对应的类是OdDbViewportTalbeRecord。

UCSs用户坐标系统 user coordination system

包含了绘制用的命名的用户坐标系统。一个UCS是一个用来替代世界坐标系统的本地坐标系统的特例。对应的类是OdDbUCSTableRecord。

注册的应用程序 registered application

包含绘图的注册应用程序。这是一些注册到CAD系统的应用程序，对应的类是OdDbRegAppTableRecord。

对象字典 object dictinary

对象字典存放一定数量的AutoCAD对象，包括字典的入口，多行风格，组，还有应用程序定义的自定义对象，对象字典是一个OdDbDictionary类型的对象。

内存管理

所有堆内存的申请都需要调用一下定义在OdAlloc.h中的函数：

```
extern ALLOCDLL_EXPORT void* odrxAlloc(size_t nBytes);
```

extern ALLOCDLL_EXPORT void* odrxRealloc(void* pMemBlock, size_t newSize, size_t oldSize);

extern ALLOCDLL_EXPORT void odrxFree(void* pMemBlock);

另外，Teigha for .dwg文件调用全局的new和delete C++操作符，这些函数(odrx*函数，new，delete)的默认实现是在TD_Alloc模块里面。客户可以使用合适的形式来链接这些模块的默认实现(只是简单滴使用系统的malloc，recalloc和free函数)。

此外，客户可以实现这些函数的自定义版本，并把他们链接到自己的应用程序中。考虑到客户程序需要完全控制Teigha for dwg文件的堆申请，Teigha for dwg文件允许使用由客户提供这些函数的自定义实现。

页支持

绘图文件的全部和部分载入

默认情况下，当Teigha for .dwg文件载入一个.dwg或者.dxf文件时，文件中所有的对象全部被加载进内存。然而传递给OdDbDAtabase::readFile或者OdDbHostAppServices::readFile的bPartial参数为true的话，对于.dwg文件只有部分数据会被加载进来。部分加载时，一个最小的父对象集合加载进内存，其余的对象在客户程序打开时根据需要加载进内存。这项技术对于只需要访问一小部分数据的情况是十分理想的。例如，应用程序只扫描图层表来部分加载数据，只有图层表加载进来，明显节省了内存和处理时间。

页支持

尽管部分载入对于一些程序时理想的，但是如果一个程序需要访问数据库中大多数或者全部对象的话，它并不能显著降低硬件资源的要求，因为最终的结果是全部对象都要被载入内存的，就像是全部载入一样了。

对于一些限制堆空间的环境，载入一个文件时，Teigha for .dwg文件包含2个可以用来减少全部堆大小要求的优化，

·关闭对象时，卸载一部分打开的数据库中的数据库对象。在这个方案中，用户部分载入一个dwg文件，然后一个特定的对象打开时，它会以部分加载方式加载进来。当它关闭时，会被加进一个“卸载队列”中。它会在下一个对OdDbPageObjects调用时卸载出内存。这允许客户程序在任何时候都可以控制全部存放在堆中的对象。在主数据库对象只需要打开一次的情形下，这个优化是很有用的，例如把OdDbDatabase导入一个第三方应用程序中。这个优化可以通过对用户的OdDbPageController::pagingType()设置kUnload标志来完成(见下面)。只有没有经过修改的对象才可以被卸载出内存。

·页对象会在关闭的时候临时缓存起来，这些页对象适用于任何数据库(不限于部分载入得到的数据库)中的任何数据库对象（全部载入的和部分载入的）。比如kUnload情形下，只有客户程序调用odDbPageObjects才会发生实际的页操作。临时缓存由函数

OdDbPageController::read和OdDbPageController::write函数决定，这个优化可以通过设置客户重载OdDbPageController::pagingType()的返回值kPage来开启(参见下面)。

·如果卸载和页操作同时被设置好，那么对象关闭的时候，卸载操作首先作用于部分打开数据库中未修改的对象。修改过的对象和全部载入数据库情形下的对象将会在关闭时发生页操作换出，正如卸载不适应这些情形。

要使用上述任何策略都必须做这些步骤：

·创建一个自定义类，继承OdDbPageController，重载开启想要的页操作所需的函数.

·返回一个重载了OdDbHostAppServices::newPageController的客户端所的自定义的页控制对象示例

·使用oDDbPageObjects来调用页操作

参见Examples/ExServices中的ExPageController.cpp文件，这是一个实现了卸载和页操作的类。OdaMfcApp演示了如何使用这些示例中的类。开启页操作能够明显地减少Teigha for .dwg文件程序对整个堆的需求。

给对象命名

对象的名字最多可达255个字符，名字中可以包含字母(a-z和A-Z)，数字(0-9)，下划线(_)，一些特殊符号($、!、+、-、等等)和空白符。大多数名字也可以包含UNICODE字符，例如图层的名字就可以包含符号。

对象的名字不能包含以下特殊字符：

·大于号和小于号(< >)

·斜线和反斜线(/ \)

·引号(")

·冒号(:)

·分号(;)

·问号(?)

·逗号(,)

·星号(*)

·竖线(|)

·等号(=)

·单引号(')

·Unicode字体创建的特殊符号

注意：一些对象的名字前或后都不能够包含空格。

字体处理

Teigha for .dwg文件需要访问字体文件以便计算文本的尺寸以及矢量化文本。Teigha for .dwg文件支持SHX字体和Truetype字体两种。但是，目标系统必须拥安装有正确的字体文件，以便Teigha for .dwg文件能够矢量化并得到特定字体文本的尺寸大小，造成文本矢量化/大小计算不正确的一个最常见的原因是找不到字体文件。这种情况下，Teigha for .dwg文件会为这个找不到的字体使用一种新的字体来替代。但是这常常会造成矢量化和大小计算的误差，因为在替代字体中的字符的尺寸会不同于找不到的字体的字符尺寸。参见下面的本地字体和字体文件得到更多关于字体替代处理的详细信息。

字体库

Teigha for .dwg文件包含了平台平等的代码来读取和处理SHX字体文件。Truetype字体以一种平台特化的方式来处理，在windows中，Teigha for .dwg文件系统会使用Windows调用来绘制TTF字符，在所有其他的字体库则使用中Freetype字体库(TD_XXXX_Freetype.*)。

字体表示法

SHX字体(常规的SHX字体和bigfonts都有)对应一个文件名称，例如txt.shx。Truetype字体可由一个文件名称来表示，例如arial.ttf，或者是一个字体名称如"Time New Roman"。

定位字体和字体文件

当Teigha for .dwg文件系统尝试访问一个字体文件时按照以下步骤进行：

·调用OdDbHostAppServices::getPreferableFont()，它使得客户程序在Teigha for .dwg文件系统得到指定字体前使用另一种字体进行替代。默认的操作是根据当前字体映射文件(参见下面的字体映射文件格式)的内容来尝试替换当前的字体。

·如果OdDbHostAppServices::getPerferableFont()返回一个替代字体的名称，这个字体将被传递给OdDbHostAppServices::findFile()函数(用户可以重载)。这些函数遵循以下步骤，一旦找到想要的文件就会停止：

*尝试访问as-is文件（没有修改路径）

*尝试访问当前路径下的文件

*查询系统变量“ACAD”，而后尝试定位这个变量包含的路径中的字体（ACAD变量应该包含一组路径，并使用分号隔开）

*尝试访问当前绘图程序所在的路径中的字体文件（如果可以的话）

*对于Trutype字体文件，尝试定位系统目录下的字体文件（仅对于Windows系统）

·如果getPreferrableFont没有返回替换字体，后者findFile调用替换字体没有找到有效的字体而失败，那么将进行一下动作：

*如果一个字体文件名字可用，Teigha for .dwg文件系统将用这个文件名调用OdDbHostAppServices::findFile函数。

*如果只有一个字体名字可用，Teigha for .dwg文件系统调用OdDbHostAppServices::ttfFileNameByDescriptor用字体来匹配文件名称。而后为对应的文件名调用OdDbHostAppServices::findFile。

·如果以上步骤失败没有找到一个字体文件，或者如果一个字体可用，Teigha for .dwg文件系统尝试创建一个Windows字体而不使用文件名(仅限于Windows系统)

·如果以上步骤全部失败，那么Teigha for .dwg文件调用OdDbHostAppServices::getSubstituteFont来得到一个丢失字体的替换字体。OdDbHostAppServices::getSubstituteFont的默认实现是返回OdDbHostAppServices::getAlternateFontName()的调用结果，它返回的是FONTALT系统变量。

字体映射文件

一个字体映射文件是一个文本文件，其中每一行指定了一个替换字体，字体替换的形式是：

<原字体>；<替换字体>

原字体可以使字体名字或者是文件名，但是替换字体必须是一个文件名，例如：

romb;ROMAB__.TTF

romi;ROMAI__.TTF

AutoCAD提供的acad.fmp文件可以拿来使用，因为它的格式和Teigha for .dwg文件系统的格式是一样的。

默认字体

Teigha for .dwg文件包含两个内建的默认SHX文件，一个像txt.shx一样简单的字体，和一个gdt字体。如果上述方法都没有得到一个可用的字体或字体文件时，这些字体将用于矢量化字体和字体尺寸计算时使用。

创建自定义命令

Teigha维护了一个OdEdCommandStack类型的全局命令行栈，这个栈存储了客户端程序创建的自定义命令。使用odedRefCmds可以访问这个命令行栈。OdEdCommandStack保存了若干OdEdCommand对象，一个OdEdCommand有一个全球的、本地的以及组的名字。全局的名字没有经过本地化，本地的是本地化过的，组名作为一种将一系列命令包含在栈中的机制，这样就可以由OdEdCommandStack::newGroupIterator函数访问。OdEdCommandStack包含一个模拟执行函数void execute(OdEdCommandContext* pCmdCtx)，用来执行命令的。

执行命令的时候，命令会被传递一个可以访问当前数据的OdEdCommandContext实例，同时还传给一个I/O接口，用来实现交互命令。

参见下列话题得到更多信息：

创建自定义命令的示例；

注册自定义命令；

调用自定义命令；

创建自定义命令的示例

下面是 OdEdCommand 子类的一个简单示例，执行的时候这个对象从内部传递的 I/O 对象中返回一个字符串，并将字符串写回。

```
class SampleCmd : public OdStaticRxObject<OdEdCommand>
{
 public:
    const OdString    localName() const { return globalName(); }
    const OdString    groupName() const { return "ODA Example Commands"; }
    const OdString    globalName() const { return OdString("SampleCommand"); }

    // Body of the custom command.
    void execute(OdEdCommandContext* pCmdCtx)
    {
        OdDbCommandContextPtr    pDbCmdCtx(pCmdCtx);            // downcast for database access
        OdDbDatabasePtr    pDb = pDbCmdCtx->database();       // Current database
        OdEdUserIO*    pIO = pDbCmdCtx->userIO();

        OdString    sKey = pIO->getString("Enter a value: ", "< default >");
        pIO->putString(sKey + " was entered");
     }
};
```

自定义命令可以执行数据操作，创建新对象，查询输入并执行得到的输入命令等，参见ExCustObj_dll示例工程，该示例创建基于用户输入的示例命令(这些命令可以在

OdaMfcAppDll示例中选择Tools/Load Applications加载ExCustObj_dll创建的TX模块来执行)。一旦一个app被加载进系统，就可以通过Edit/Registered Commands访问命令。

注册自定义命令

在全局命令栈中，注册命令用OdEdCommandStack::addCommand函数，例如:

SampleCmd cmd;

odedRegCmds()->addCommand(&cmd);

cmd对象一旦被传入，直到其被从全局栈中删除才会被析构。

调用自定义命令

创建一个 OdEdBaseIO 子类

执行自定义命令需要一个OdEdBaseIO对象实例。OdEdBaseIO的子类必须重载OdEdBaseIO::getString()和OdEdBaseIO::putString()函数。这个函数提供了用户自定义命令初级的输入输出功能。OdEdUserIO提供了一个更丰富的输入输出集合（读取整型、浮点型...数据）。

创建 OdDbCommandContext 对象并执行命令

接下来要创建OdDbCommandContext对象，它将被传递给OdEdCommand的执行函数。而后调用全局命令栈中的命令。例如 :

```
//创建命令上下文，同时还有一个OdEdbaseIO子类，和一个OdDbDatabase

OdDbCommandContextPtr pCmdCtx = OdDbCommandContext::createObject(pMyOdEdBaseIO, pDatabase);

//检索全局命令栈

OdEdCommandStackPtr pCommands = ::odedRegCmds();

//连同一个全局名字"CustomCommand"执行注册命令，
```

pCommands->excuteCommand("CustomComand", pCmdCtx);

创建自定义对象

这一节介绍如何从 OdDbEntity 类中派生一个自定义类，并且包含了 OdDbEntity 类中提供的虚函数的重载的例子。还包括重载普通实体的操作，比如对象管点，握点、拉伸点。

OdDbEntity 是所有拥有图形表示的数据库对象的基类。OdDbEntity 类继承自 OdDbObject 类。创建一个自定义类包含以下步骤：

派生自定义实体类
实现绘制函数
实现保存和加载函数
实现映射函数
实现管点函数
实现握点函数
实现炸开函数

派生一个自定义实体类

创建自定义实体类的第一步是继承 OdDbEntity 或者 OdDbObject 类得到自定义实体类。继承自定义实体类你必须做一下声明：

```
class MyEntityObj : public OdDbEntity
{
 public:
    ODDB_DECLARE_MEMBERS(MyEntityObj);

    MyEntityObj();
    ~MyEntityObj();
};
```

对于自定义实体，必须定义构造函数和析构函数，这可以通过使用 ODDB_DECLARE_MEMBERS 宏的方法。所有的数据库对象类都需要这个宏。

自定义实体类的实现必须调用 ODRX_DFX_DEFINE_MEMBERS 宏，它创建类声明中 ODDB_DECLARE_MEMBERS 宏声明的函数。

```
ODRX_DXF_DEFINE_MEMBERS(MyEntityObj,                    // class name
                        OdDbEntity,                     // parent class name
                        DBOBJECT_CONSTR,                // creation pseudo-constructor
                        OdDb::kDHL_CURRENT,             // file version
                        OdDb::kMReleaseCurrent,         // maintenance release version
                        OdDbProxyEntity::kTransformAllowed |
                        OdDbProxyEntity::kColorChangeAllowed |
                        OdDbProxyEntity::kLayerChangeAllowed,   // proxy flags
                        MYENTITYOBJ,                    // dxf class name
```

MyCustObjs|Description: Run-time Extension Example) //
application name
注意程序名字 MyCustObjs 必须与包含这个对象(MyCustObjs.drx)的实现的共享库名吻合。
要实现一个最小自定义实体，必须重载以下函数：worldDraw(),dwgOutFields() 和 dwgInFields()。同时也需要实现以下函数：dxfOutFields()和 dxfInFields（）。

实现绘制函数

自定义类调用worldDraw()和viewDraw()函数绘制实体，所有继承自OdDbEntity类的子类都必须实现worldDraw()函数，viewportDraw()函数可以不重载。

virtual bool OdDbEntity::worldDraw(OdGiWorldDraw* pWd)const;

virtual bool OdDbEntity::viewportDraw(OdGiWorldDraw* pVd)const;

worldDraw()函数可以在模型空间视图或者paper-space视图上下文指定展示实体图形的一部分。viewportDraw()函数是创建实体的视图独立的图形。如果实体没有独立于视图的图形。那么worldDraw函数必须返回true。并且不能实现voewportDraw()函数，如果有任何独立于视图的图形，那么worldDraw()函数就必须返回false，还必须实现viewportDraw()函数。

worldDraw() 函数带有一个 OdGiWorldDraw 对象的指针，OdGiWorldDraw 是一个OdGiWorldGeometry和OdGiSubEntityTraits对象的容器类。在worldDraw()函数中，可以使用geometry() 函数得到 OdGiWorldGeometry 对象，用 subEntityTraits() 函数得到OdGiSubEntityTraits对象。

OdGiWorldGeomitry对象使用一系列绘图原语改写阵列，刷新内存。原语是最低级的绘图命令。在世界坐标中全局的几何对象有以下绘图原语的绘图函数：circle, circularArc, ellipArc, polygon,mesh,shell,text, xline, ray,例如：

pWd->geometry().circularArc(center, radius, normal, startvec, angle, kOdGiArcChord);

OdGiSubEntityTraits对象设置图形属性值，使用它的一些属性函数：color,layer, line type, polygon fill type, selection mark等，例如:

pWd->subEntityTraits().setTrueColor(entityColor());

pWd->subEntityTraits().setFillType(kOdGiFillNever);

viewportDraw()函数带一个OdGiViewportDraw对象的指针参数，它构建一个特定视图的实体图形。视口绘图对象也是一个包含其他对象的容器。它包含了OdGiViewportGeometry, OdGiSubEntityTraits, AcGiViewport。视口对象提供了一些查询变换矩阵和视图参数的函数.

如果你在实现worldDraw()函数或者viewportDraw()函数时使用了选中标记，那么你必须在调

用任何OdGiWorldGeometry或者OdGiViewGeometry的操作(包括调用mesh()和shell()函数)生成图形之前设置第一个选中标记，例如：

pWd->subEntityTraits().setSelectionMarker(1);

pWd->geometry().circle(center, radius, normal);

如果需要，你可以将第一个标记设置一个臆造的数值，比如-1.

odGiWorldDraw或者OdGiViewportDrawd对象不应该作为全局的或者是静态的变量用来存储数据。不要越过worldDraw()和vireportDraw()函数保存这些对象的副本。一旦这些函数返回了，OdGi类对象就不可用了。例如smiley实体的实现。

```
bool AsdkSmiley::worldDraw(OdGiWorldDraw *wd) const
{
    assertReadEnabled();
    OdGeVector3d offset(0,0,0);

    // If dragging, don't fill the smiley
    if( wd->isDragging() )
    {
        wd->subEntityTraits().setTrueColor(entityColor());
        wd->subEntityTraits().setFillType( kOdGiFillNever );
    }
    else
        wd->subEntityTraits().setFillType( kOdGiFillAlways );
    // Drawing circle of face
    wd->subEntityTraits().setSelectionMarker( 1 );
    wd->geometry().circle( center(), radius(), mnormal );
    if( !wd->isDragging() )
        wd->subEntityTraits().setTrueColor(mbackcolor);
    // Drawing circle of left eye
```

```
    wd->subEntityTraits().setSelectionMarker( 2 );
    wd->geometry().circle( leftEyeCenter(), meyesize, mnormal );

    // Drawing circle of right eye
    wd->subEntityTraits().setSelectionMarker( 3 );
    wd->geometry().circle( rightEyeCenter(), meyesize, mnormal );
    // Drawing arc of mouth
    OdGePoint3d    smilecen( mouthCenter() + offset ),
                   startpt( mouthLeft() + offset ),
                   endpt( mouthRight() + offset );

    OdGeVector3d startvec = startpt - smilecen,
                   endvec = endpt - smilecen;

    double mouthangle = startvec.angleTo( endvec );
    wd->subEntityTraits().setSelectionMarker( 4 );
    wd->geometry().circularArc( smilecen, mouthRadius(), mnormal, startvec, mouthangle,
kOdGiArcChord );
    return true;
}
```

实现保存和加载函数

自定义对象通过dwgInFields()函数.dwg文件中加载实体对象，通过dwgOutFields()函数将一个实体对象保存到.dwg文件中。还有其他的一些用处，比如说克隆。这些函数不是必须的。但是强烈要求为每个实体对象实现他们。

```
virtual OdResult OdDbEntity::dwgInFields(OdDbDwgFiler* filer);
virtual void OdDbEntity::dwgOutFileds(OdDbDwgFiler* filer)const;
```

每个函数都带有一个filer指针作为其唯一的参数。实体对象通过一个filer读写数据。读取数据是通过filer对象中'rd'类函数实现的。写数据是由

filer对象中'wr'类函数实现的。这些函数是由不同的数据结构区分的。以下是一些让你根据不同数据类型使用的函数。

rdBool() / wrBool() - read / write a boolean value

rdInt8() / wrInt8() - read / write a byte

rdInt16() / wrInt16() - read / write a integer value of int type

rdInt32() / wrInt32() - read / write a longer value of long type

rdDouble() / wrDouble() - read / write a real value of double type

rdString() / wrString() - read / write a string value

rdPoint3d() / wrPoint3d() - read / write a 3d point of OdGePoint3d type

rdVector3d() / wrVector3d() - read / write a 3d vector of OdGeVector3d type

rdScale3d() / wrScale3d() - read / write a 3d scale of OdGeScale3d type

加载数据必须和保存数据的顺序一致。在加载和保存数据时，这些函数必须根据基类的dwgInFields()或者dwgOutFields()函数执行。对于每一个实体对象建议你保持同一个版本编号。

在实现dwgInFields()函数中，开始从filer对象中读取数据时，有必要先检查一下当前对象是要写入数据吗？这个使用assertWriteEnabled()函数实现的，如果当前对象不是为了写入数据而打开的就会抛出一个异常，这个对象的控制器会自动撤销改动消息。

在改变了实体的数据之后，有必要调用recordGraphicsModified()函数，它会设置标记指示实体的图形有改动。实现dwgOutFields()函数时，在写入filer数据前，有必要检查当前的对象是为了读取数据而打开的吗？这个由assertReadEnabled()函数实现。如果当前的对象不是为了读取数据而打开的，就会抛出异常。如果读数据的结果是成功的那么'In'类函数应返回eOk。

例如smiley实体的实现：

```
const double kCurrentVersionNumber = 1.0;

void AsdkSmiley::dwgOutFields(OdDbDwgFiler* filer) const

{

    assertReadEnabled();
```

```
    OdDbEntity::dwgOutFields( filer );                    // Method of base class
    filer->wrDouble( kCurrentVersionNumber );
    filer->wrPoint3d( center() );
    filer->wrDouble( radius() );
    filer->wrVector3d( normal() );
    filer->wrDouble( eyesApart() );
    filer->wrDouble( eyesHeight() );
    filer->wrDouble( eyeSize() );
    filer->wrPoint3d( mouthLeft() );
    filer->wrPoint3d( mouthBottom() );
    filer->wrPoint3d( mouthRight() );
    filer->wrInt32( backColor().color() );
}
OdResult AsdkSmiley::dwgInFields(OdDbDwgFiler* filer)
{
    assertWriteEnabled();
    OdDbEntity::dwgInFields( filer );                     // Method of base class
    // Read version number
    if( filer->rdDouble() > kCurrentVersionNumber ) return eMakeMeProxy;

    // Read face data
    setCenter( filer->rdPoint3d() );
    setRadius( filer->rdDouble() );
    setNormal( filer->rdVector3d() );

    // Read eyes data
```

```
    setEyesApart( filer->rdDouble() );
    setEyesHeight( filer->rdDouble() );
    setEyeSize( filer->rdDouble() );

    // Read mouth data
    OdGePoint3d mouthleftpt=filer->rdPoint3d(),
                mouthbottompt=filer->rdPoint3d(),
                mouthrightpt=filer->rdPoint3d();
    setMouth( mouthleftpt, mouthbottompt, mouthrightpt );

    // Read color data
    mbackcolor.setColor( filer->rdInt32() );
    recordGraphicsModified();
    return eOk;
}
```

dxfInFields()函数和dxfOutFields()函数的实现方法类似，要注意的是DFX格式是以文本的方式存储数据的，同一数据可以位于文件中的不同位置。例如一个.dxf文件从一个程序中获得数据，但是可能没有存放了所有的数据。一个函数加载.dxf文件或许不能全部读取文件中的数据。另外，DXF格式允许你跳过一行数据。

在DXF格式中的每个成组数据都有一个组编号，用来标示数据。写入函数('wr'类函数)会为每个成组数据设置一个组编号并写入到dxf文件中。读取函数('rd'类函数)不知道文件中的成组数据是什么。为了在读取数据的时候确定成组数据，你必须使用OdDbDxfFiler对象的nextItem()函数，它会返回一个成组数据的组编号，可以使用switch-case结构检查接收到的组编号。可以使用OdDbDxfFiler对象的atEOF()函数结束一个实体数据的读取，如果filer对象正好在对象数据的结尾，这个函数返回true，丢失的数据可以用dxfInFields()函数在读取数据的时候初始化，或者是在读取数据前设置好。

在dxf文件中要确定一个自定义实体的话，有必要保存实体的名称，它可以由desc()->name()函数得到。这个desc()函数返回自定义实体的描述字符。

它在dxfOutFields()函数向文件写入数据时作为实体对象的名称，在使用dxfInFields()读取数据的时候用来对比实体的名字。你可以使用atSubclassData()函数来对比名字，如果一个filer

对象是指定参数的子类数据标志的话返回true。这里建议你在实体的名字之后、数据之前存放一个版本编号，例如，smiley实体的实现如下：

```
void AsdkSmiley::dxfOutFields(OdDbDxfFiler *filer) const
{
    assertReadEnabled();
    OdDbEntity::dxfOutFields( filer );                    // Method of base class
    filer->wrSubclassMarker( desc()->name() );
    filer->wrDouble( OdResBuf::kDxfReal, kCurrentVersionNumber );
    filer->wrPoint3d( OdResBuf::kDxfXCoord, center() );
    filer->wrDouble( OdResBuf::kDxfReal+1, radius() );
    if( filer->includesDefaultValues() || normal() != OdGeVector3d( 0, 0, 1 ))
    {
        filer->wrVector3d( OdResBuf::kDxfNormalX, normal() );
    }
    filer->wrDouble( OdResBuf::kDxfReal+2, eyesApart() );
    filer->wrDouble( OdResBuf::kDxfReal+3, eyesHeight() );
    filer->wrDouble( OdResBuf::kDxfReal+4, eyeSize() );
    filer->wrPoint3d( OdResBuf::kDxfXCoord+1, mouthLeft() );
    filer->wrPoint3d( OdResBuf::kDxfXCoord+2, mouthBottom() );
    filer->wrPoint3d( OdResBuf::kDxfXCoord+3, mouthRight() );
}
OdResult AsdkSmiley::dxfInFields(OdDbDxfFiler *filer)
{
    assertWriteEnabled();
    OdResult es;
    OdGePoint3d center = center(),
                mouthleftpt = mouthLeft(),
```

```
                    mouthbottompt = mouthBottom(),
                    mouthrightpt = mouthRight();
if( eOk != (es = OdDbEntity::dxfInFields( filer )) ) return es;

// Check that we are at the correct subclass data
if( !filer->atSubclassData( desc()->name() )) return eBadDxfSequence;

// Read version number (must be first)
if( filer->nextItem() != OdResBuf::kDxfReal ) return eMakeMeProxy;
if( filer->rdDouble() > kCurrentVersionNumber ) return eMakeMeProxy;

// Read of smiley data
while( !filer->atEOF() )
{
    switch( filer->nextItem() )
    {
        case OdResBuf::kDxfXCoord:
                filer->rdPoint3d( center );
                setCenter( center );
                break;
        case OdResBuf::kDxfReal+1:
                setRadius( filer->rdDouble() );
                break;
        case OdResBuf::kDxfNormalX:
                filer->rdVector3d( mnormal );
                break;
```

```
            case OdResBuf::kDxfReal+2:
                setEyesApart( filer->rdDouble() );
                break;
            case OdResBuf::kDxfReal+3:
                setEyesHeight( filer->rdDouble() );
                break;
            case OdResBuf::kDxfReal+4:
                setEyeSize( filer->rdDouble() );
                break;
            case OdResBuf::kDxfXCoord+1:
                filer->rdPoint3d( mouthleftpt );
                break;
            case OdResBuf::kDxfXCoord+2:
                filer->rdPoint3d( mouthbottompt );
                break;
            case OdResBuf::kDxfXCoord+3:
                filer->rdPoint3d( mouthrightpt );
                break;
        }
    }
    setMouth( mouthleftpt, mouthbottompt, mouthrightpt );
    recordGraphicsModified();
    return eOk;
}
```

实现映射函数

每个实体使用transformBy()函数将一个3D转换矩阵来做实体转化。并使用getTransformCopy()函数创建自身的一个副本。getTransformCopy()函数会将提供的转换应用到新创建的副本上。

virtual OdResult OdDbEntity::transformBy(const OdGeMatrix3d& xform);

virtual OdResult OdDbEntity::getTransformedCopy(const OdGeMatrix3d& xform, OdDbEntityPtr& pCopy) const;

例如，smiley实体的实现：

```
OdResult AsdkSmiley::transformBy(const OdGeMatrix3d& xform)
{
    assertWriteEnabled();

    // Transform the center point and get the translation vector
    OdGePoint3d oldCenter( center() ),
                newCenter( center() );
    newCenter.transformBy( xform );

    OdGeVector3d transVec = newCenter - center();

    // Get the equivalent transformation
    OdGeMatrix3d newXform;
    newXform.setToTranslation( transVec );
    // Only translate the face and mouth - do not transform!
    mfacecircle.transformBy( newXform );
    mmoutharc.transformBy( newXform );
```

```
    // Get the point at a quadrant, transform it
    OdGePoint3d oldXquad = center() + OdGeVector3d( radius(), 0, 0 ),
                        newXquad( oldXquad );
    newXquad.transformBy( xform );

    // ... then scale the Smiley accordingly
    if( xform.isEqualTo( OdGeMatrix3d::kIdentity ) == false )
      scaleRadius( radius() * newCenter.distanceTo( newXquad ) /
oldCenter.distanceTo( oldXquad ));
    return eOk;
}
void AsdkSmiley::scaleRadius(const double r)          // stretch and constrict of smiley
{
    assertWriteEnabled();
    OdGePoint3d center( center() );
    double rad = radius(),
           factor = r / rad;

    setEyesApart( factor * eyesApart() );
    setEyesHeight( factor * eyesHeight() );
    setEyeSize( factor * eyeSize() );
    setMouth( mouthLeft().scaleBy( factor, center ),
              mouthBottom().scaleBy( factor, center ),
              mouthRight().scaleBy( factor, center ) );
    setRadius( r );
    recordGraphicsModified();
}
```

实现握点(Grip Point)函数

用户使用指点设备选中一个自定义实体时，会有一个握点显示出来。getGripPoints()函数必须用自定义实体中的握点填充OdGePoint3DArray数组，握点的编辑引起的实体变化可由moveGripPointsAt()函数实现。

virtual OdResult OdDbEntity::getGripPoints(OdGePoint3dArray& gripPoints) const;

virtual OdResult OdDbEntity::moveGripPointsAt(const OdGePoint3dArray& gripPoints, const OdIntArray& indices) const;

实体定义了它的握点以及如何解释用户发出的动作。使用握点编辑一个自定义实体，你必须重载getGripPoints()和moveGripPointsAt()函数，使用getGripPoints()函数可以将握点添加到一个专门的数组中，每个点从0开始依次指示了握点被添加来的顺序(第一个点=0, 第二个点=1,第三个点=2...),数组中附加的点使用数组对象的append()函数来实现。

例如，smiley实体是这样实现的 ：

```
OdResult AsdkSmiley::getGripPoints(OdGePoint3dArray& gripPoints) const
{
    assertReadEnabled();
                                                        // indices
    // Grip points to face
    OdGePoint3d center( center() );
    OdGeVector3d xoff( radius(), 0, 0 ),
                 yoff( 0, radius(), 0 );
    gripPoints.append( center );                    // 0
    gripPoints.append( center + xoff );             // 1
    gripPoints.append( center + yoff );             // 2
    gripPoints.append( center - xoff );             // 3
    gripPoints.append( center - yoff );             // 4
    // Grip points to mouth
    gripPoints.append( mouthLeft() );               // 5
```

```
    gripPoints.append( mouthRight() );                // 6
    gripPoints.append( mouthBottom() );               // 7
    gripPoints.append( OdGeLineSeg3d(mouthLeft(),mouthRight()).midPoint() );    // 8

    // Grip points to eye
    xoff.x = yoff.y = meyesize;

    // Left eye
    center = leftEyeCenter();
    gripPoints.append( center );                      // 9
    gripPoints.append( center + xoff );               // 10
    gripPoints.append( center + yoff );               // 11
    gripPoints.append( center - xoff );               // 12
    gripPoints.append( center - yoff );               // 13

    // Left eye
    center = rightEyeCenter();
    gripPoints.append( center );                      // 14
    gripPoints.append( center + xoff );               // 15
    gripPoints.append( center + yoff );               // 16
    gripPoints.append( center - xoff );               // 17
    gripPoints.append( center - yoff );               // 18
    return eOk;
}
```

moveGripPointsAt()用来修改握点数组中的点。握点编辑允许你在拉伸模式先通过移动选中的握点到新的位置来拉伸对象。当用户处于拉伸模式时自定义对象调用moveGripPointsAt()函数。然而，对于一些实体，一些握点是起移动而非拉伸的作用。这些握点有文本对象、

块、线段的中点、圆的中心、椭圆的中心、还有点对象的的握点。这时moveGripPointsAt()会调用TransformBy()。moveGripPointsAt()函数习惯于调用transformBy()函数。如果没有在自定义对象中重载moveGripPointsAt()函数，transformBy()函数将会被调用。

moveGripPointsAt()函数使用一个switch操作来检查握点是否被编辑过。函数的第一个参数是由getGripPoints()函数创建的握点数组，函数的第二个参数是用户在拉伸模式下编辑握点数组的下标数组，修改过的握点的下标就存放在这个数组中，握点的新位置就存放在握点数组中，你可以选中一个握点，但一般你会选中多个握点。下面的例子说明了选中多个握点的用法：

```
OdResult AsdkSmiley::moveGripPointsAt(const OdGePoint3dArray& gripPoints, const OdIntArray& indices)
{
    assertWriteEnabled();
    int    idx, count = indices.length();
    OdGePoint3d idxpoint;
    for( int i = 0; i < count ; i++ )                    // A few grip points can be selected concurrently
    {
        idx = indices[i];
        idxpoint = gripPoints[idx];
        if( idx == 0 )                                   // Move the smiley center
        {
            setCenter( idxpoint );
        }
        else if( idx >= 1 && idx <= 4 )                  // Stretch the smiley radius
        {
            scaleRadius( idxpoint.distanceTo( center() ));
        }
        else if( idx == 5 )                              // Stretch the left edge of mouth
```

```
        {
            setMouth( idxpoint, mouthBottom(), mouthRight() );
            ensureRadiusMouth();
        }
        else if( idx == 6 )                                     // Stretch the right edge
of mouth
        {
            setMouth( mouthLeft(), mouthBottom(), idxpoint );
            ensureRadiusMouth();
        }
        else if( idx == 7 )                                     // Stretch the bottom
edge of mouth
        {
            setMouth( mouthLeft(), idxpoint, mouthRight() );
            ensureRadiusMouth();
        }
        else if( idx == 8 )                                     // Move the mouth
        {
            moveMouthToPoint( idxpoint );
            ensureRadiusMouth();
        }
        else if( idx == 9 || idx == 14 )                        // Move the center of eyes
        {
            setEyesApart( (idxpoint.x - center().x) * 2 );              // Apart >= 2*eyeSize
            if( eyesApart() < 2 * eyeSize() ) setEyesApart( 2 * eyeSize() );
            setEyesHeight( idxpoint.y - center().y );                   // Height >= eyeSize
            if( eyesHeight() < eyeSize() ) setEyesHeight( eyeSize() );
```

```
            ensureRadiusEyes();
        }
        else if((idx >= 10 && idx <= 13) ||
                (idx >= 15 && idx <= 18))                    // Stretch the radius of
eyes
        {
            setEyeSize( idxpoint.distanceTo( (idx < 14) ? leftEyeCenter() : rightEyeCenter() ) );

            if ( 2 * eyeSize() > eyesApart() ) setEyeSize( eyesApart() / 2 );

            ensureRadiusEyes();
        };
    }
    return    eOk;
}
void AsdkSmiley::moveMouthToPoint(const    OdGePoint3d point)
{
    OdGePoint3d middle( OdGeLineSeg3d(mouthLeft(), mouthRight()).midPoint() );
    OdGeVector3d offset( point.x - middle.x, point.y - middle.y, point.z - middle.z );
    setMouth( mouthLeft() + offset, mouthBottom() + offset, mouthRight() + offset );
}
void AsdkSmiley::ensureRadiusMouth()
{
    double    d;
    OdGePoint3d center( center() );
    if( (d = center.distanceTo( mouthLeft() )) > radius() / 1.1) setRadius( 1.1 * d );
```

```
    if( (d = center.distanceTo( mouthRight() )) > radius() / 1.1) setRadius( 1.1 * d );
    if( (d = center.distanceTo( mouthBottom() )) > radius() / 1.1) setRadius( 1.1 * d );
}
void AsdkSmiley::ensureRadiusEyes()
{
    double    d = center().distanceTo(leftEyeCenter()) + eyeSize();
    if (d > radius() / 1.1) setRadius( 1.1 * d );
}
```

实现管理点(Snap Point)函数

为了支持自定义实体的管理模式，要重载getOsnapPoints()函数。自定义实体对象调用这个函数得到当前模式下的相关管理点。如果你不想要实体在某些模式下支持管理点。你可以将这些你想支持的模式过滤出来做操作，而对于其他情况直接返回eOk。如果有多个对象处于管理模式，那么每个对象会调用一次这个函数。

```
virtual OdResult OdDbEntity::getOsnapPoints( OdDb::OsnapMode osnapMode, int gsSelectionMark,
                                             const OdGePoint3d& pickPoint, const OdGePoint3d& lastPoint,
                                             const OdGeMatrix3d& xfm, const OdGeMatrix3d& ucs, OdGePoint3dArray& snapPoints ) const;
```

getOsnapPoints()函数必须用定义在自定义实体中的管理点填充OdGePoint3dArray数组。数组中额外的点将使用数组对象的append()函数重新分配。函数中通常是一个switch结构操作。它会检查管理模式并填充管理点。

例如，smiley实体的实现：

```
OdResult AsdkSmiley::getOsnapPoints( OdDb::OsnapMode osnapMode,
                                     int gsSelectionMark,
                                     const OdGePoint3d& pickPoint,
                                     const OdGePoint3d& lastPoint,
                                     const OdGeMatrix3d& xfm,
```

```
                                        const OdGeMatrix3d& ucs,
                                        OdGePoint3dArray& snapPoints ) const
{
    assertReadEnabled();
    switch( osnapMode )
    {
        case OdDb::kOsModeCen:                          // Osnap center point
            snapPoints.append( center() );
            snapPoints.append( leftEyeCenter() );
            snapPoints.append( rightEyeCenter() );
            return eOk;
        case OdDb::kOsModeNear:
        case OdDb::kOsModeQuad:                         // Osnap quad points
            {
            OdGeVector3d xoff(radius(),0,0),
                         yoff(0,radius(),0);
            OdGePoint3d center( center() );

            snapPoints.append( center + xoff );
            snapPoints.append( center + yoff );
            snapPoints.append( center - xoff );
            snapPoints.append( center - yoff );
            }
            return eOk;
                                                        // Osnap middle points
        case OdDb::kOsModeMid:
```

```
            snapPoints.append( OdGeLineSeg3d( mouthLeft(), mouthRight() ).midPoint() );
            return eOk;
        case OdDb::kOsModeEnd:                          // Other osnap mode is ignored
        case OdDb::kOsModeNode:
        case OdDb::kOsModeIns:
        case OdDb::kOsModePerp:
        case OdDb::kOsModeTan:

        default: break;
    }
    return eInvalidInput;
}
```

实现炸开函数

一般来说，一个自定义对象由一些简单的实体组成，比如线条、圆形、弧形等等。这些简单实体作为一个单独的对象处理。在一些情况下，我们很有必要将一个自定义实体炸成多个简单实体。例如，一个窗口对象可以炸成窗台、窗框、双层玻璃，还有通风口。每个实体自行定义了如何将自己炸成简单实体。要将一个自定义实体炸成简单实体，你必须重载explode()函数。

virtual OdResult OdDbEntity::explorde(OdRxObjectPtrArray& entitySet) const;

explord()函数处理实体对象的智能指针数组，用createObject()函数创建简单实体，这个函数返回对象的智能指针。每个简单实体有一个智能指针类型，这个类型的名称由类名和一个'Ptr'后缀组成。你必须使用简单实体类的对应智能指针类型。

使用append()函数重设数组大小并将对象添加进数组中。要想得到实体的指针，必须使用智能指针的get()函数。

使用setPropertitiesFrom()函数可以将一个指定的对象的属性拷贝到这个简单对象中，你必须使用这个函数设置一个创建对象的默认属性。例如：

OdResult AsdkSmiley::explode(OdRxObjectPtrArray& entitySet) const

```
{
  assertReadEnabled();
  OdDbCirclePtr pFace = OdDbCircle::createObject();              // Create circle for face
  pFace->setPropertiesFrom(this);
  pFace->setNormal( normal() );
  pFace->setRadius( radius() );
  pFace->setCenter( center() );
  entitySet.append( pFace.get() );
  OdDbCirclePtr pLeftEye = OdDbCircle::createObject();          // Create circle for left eye
  pLeftEye->setPropertiesFrom( this );
  pLeftEye->setNormal( normal() );
  pLeftEye->setRadius( eyeSize() );
  pLeftEye->setCenter( leftEyeCenter() );
  entitySet.append( pLeftEye.get() );
  OdDbCirclePtr pRightEye = OdDbCircle::createObject();         // Create circle for right eye

  pRightEye->setPropertiesFrom( this );
  pRightEye->setNormal( normal() );
  pRightEye->setRadius( eyeSize() );
  pRightEye->setCenter( rightEyeCenter() );
  entitySet.append( pRightEye.get() );
  OdDbLinePtr pLine = OdDbLine::createObject();                  // Create arc chord for mouth

  pLine->setPropertiesFrom( this );
  pLine->setNormal( normal() );
  pLine->setStartPoint( mouthLeft() );
```

```
    pLine->setEndPoint( mouthRight() );
    entitySet.append( pLine.get() );

    OdGeVector3d normvec( 1, 0, 0 );
    double dStartAngle = 2 * kPi - (mouthLeft() - mouthCenter()).angleTo( normvec );
    double dEndAngle = 2 * kPi - (mouthRight() - mouthCenter()).angleTo( normvec );
    OdDbArcPtr pArc = OdDbArc::createObject();                    // Create arc for mouth

    pArc->setPropertiesFrom( this );
    pArc->setNormal( normal() );
    pArc->setRadius( mouthRadius() );
    pArc->setCenter( mouthCenter() );
    pArc->setStartAngle( dStartAngle );
    pArc->setEndAngle( dEndAngle );
    entitySet.append( pArc.get() );
    return eOk;
}
```

explode()函数会将一个实体炸成简单一些的实体，如果得到的实体不是元实体，函数将会返回eExplodeAgain。

Simley 对象例子

这个例子演示了如何实现一个自定义对象。AsdkSmiley类用到下面的定义：

```
#define kPi 3.14159265358979323846

class AsdkSmiley : public OdDbEntity
{
```

```
  private:

    OdGeVector3d      mnormal;                                  // smiley normal
vector

    OdGeCircArc3d    mfacecircle;                              // smiley face

    OdGeCircArc3d    mmoutharc;                                // smiley mouth arc
(not written out)


    double    meyesapart;                                      // smiley eyes apart

    double    meyesheight;                                     // smiley eyes height

    double    meyesize;                                        // smiley eyes size


    OdCmEntityColor mbackcolor;                                // smiley color


  protected:

    void ensureRadiusMouth();

    void ensureRadiusEyes();


  public:

    ODDB_DECLARE_MEMBERS(AsdkSmiley);                          // Declare
database methods


    AsdkSmiley();                                              // Constructor

    virtual ~AsdkSmiley();                                     // Destructor


    // Step 1

    bool worldDraw(OdGiWorldDraw* mode) const;                 // for drawing

    OdResult dwgInFields(OdDbDwgFiler* pFiler);                // for loading from .dwg
file
```

```
    void dwgOutFields(OdDbDwgFiler* pFiler) const;                    // for saving in .dwg file

    // Step 2
    OdResult dxfInFields(OdDbDxfFiler *filer);                        // for loading from .dxf file
    void dxfOutFields(OdDbDxfFiler *filer) const;                     // for saving in .dxf file

    // Step 3
    OdResult transformBy(const OdGeMatrix3d& xform);                  // for moving and rotating
    void scaleRadius(const double r);                                 // for scaling

    // Step 4
    OdResult getGripPoints(OdGePoint3dArray& gripPoints) const;
    OdResult moveGripPointsAt(const OdGePoint3dArray& gripPoints, const OdIntArray&
indices);

    // Step 5
    OdResult getOsnapPoints(OdDb::OsnapMode osnapMode, int gsSelectionMark,
                            const OdGePoint3d& pickPoint, const OdGePoint3d&
lastPoint,
                            const OdGeMatrix3d& xfm, const OdGeMatrix3d& ucs,
                            OdGePoint3dArray& snapPoints) const;

    // Step 6
    OdResult explode(OdRxObjectPtrArray& entitySet) const;        // to destroy the smiley object

    //
    // Smiley property functions
```

```
    //

    virtual OdCmEntityColor backColor() const;                    // smiley back color
    virtual void setBackColor(const OdCmEntityColor& color);
    virtual OdGeVector3d normal() const;                          // smiley normal
    virtual void setNormal(const OdGeVector3d v);
    virtual OdGePoint3d center() const;                           // smiley center
    virtual void setCenter(const OdGePoint3d c);
    virtual double radius() const;                                // smiley radius
    virtual void setRadius(const double r);
    virtual double eyesApart() const;                             // eyes apart
    virtual void setEyesApart(const double a);
    virtual double eyesHeight() const;                            // eyes height
    virtual void setEyesHeight(const double h);
    virtual double eyeSize() const;                               // eyes size
    virtual void setEyeSize(const double s);

    virtual OdGePoint3d leftEyeCenter() const;                    // center of left eye
    virtual OdGePoint3d rightEyeCenter() const;                   // center of right eye
    virtual double mouthRadius() const;                           // radius of mouth arc
    virtual OdGePoint3d mouthCenter() const;                      // center of mouth arc
    virtual OdGePoint3d mouthLeft() const;                        // left point of mouth arc
    virtual OdGePoint3d mouthRight() const;                       // right point of mouth arc
    virtual OdGePoint3d mouthBottom() const;                      // bottom point of mouth
arc

                                                                  // whole mouth
```

```
    virtual void setMouth(const OdGePoint3d& left, const OdGePoint3d& bottom, const OdGePoint3d& right);

    void moveMouthToPoint(const OdGePoint3d point);

};
```

（译者注：关键是要看看自定义图形绘制类都重载了哪些函数）

构建应用程序

创建自定义程序

你可以动态加载自定义程序和自定义对象。

创建自定义程序

为了创建一个自定义程序，用户应该创建一个 OdRxModule 的子类，像下面这样：

```
class CustomObjectsModule : public OdRxModule
{
 protected:
    void initApp()
    {
      // register the custom object types defined in this module
      ExCustObject::rxInit();
      ExCustEntity::rxInit();
      ExSphere::rxInit();


      // custom commands can be registered here
    }


    void uninitApp()
    {
      // unregister the custom object types defined in this module
      ExSphere::rxUninit();
      ExCustEntity::rxUninit();
      ExCustObject::rxUninit();
```

```
        // remove custom commands here
    }
};

// macro that defines the entry point for this library
ODRX_DEFINE_DYNAMIC_MODULE(CustomObjectsModule);
```

一个用户程序应该包含有且仅有一个 OdRxModule 的子类，但是可以定义任意数量的自定义对象。

用户程序的命名规范

包含 OdRxModule 子类的共享库应该用<应用程序名字>.drx 的形式命名，其中<应用程序名字>是本模块中用来定义自定义对象类的应用程序名字。

自动加载客户程序

当 Teigha for .dwg 文件系统加载一个包含自定义对象或实体时，它调用 OdDbHostAppServices::loadApp 来加载与自定义对象类相关的程序。默认 情况下，系统查找与 Teigha for .dwg 文件共享库相同的目录。可以使用 odrxSetModuleSearchPath 来更改默认位置。

自定义程序的静态使用

如果需要系统为客户程序提供静态链接到另一个程序的支持。在使用自定义app的程序中包含以下宏定义，就可以让CustomObjectsModule示例实现这样的功能。

```
#ifndef _TOOLKIT_IN_DLL_

// Declare the entry point function for the custom module (one for each module).
ODRX_DECLARE_STATIC_MODULE_ENTRY_POINT(CustomObjectsModule);
ODRX_DECLARE_STATIC_MODULE_ENTRY_POINT(ModelerModule);

ODRX_BEGIN_STATIC_MODULE_MAP()
  ODRX_DEFINE_STATIC_APPLICATION("ExCustObjs",          CustomObjectsModule)
  ODRX_DEFINE_STATIC_APPLICATION("ModelerGeometry",   ModelerModule)
ODRX_END_STATIC_MODULE_MAP()

#endif
```

在静态模块映射声明好之后，客户程序需要使用ODRX_INIT_STATIC_MODULE_MAP宏一次性初始化静态映射。这个应该在odInitilize()函数之前，例如：

```
#ifndef _TOOLKIT_IN_DLL_
    ODRX_INIT_STATIC_MODULE_MAP();
#endif
    odInitialize(this);
```

一旦这些调用完成，注册的自定义模块将会按需加载（例如，当这个模块需要序列化一个自定义对象时），或者可以明确地调用OdRxDynamicLinker::loadModule来加载模块，并给ODRX_DEFINE_STATIC_APPLICATION宏的第一个参数传递应用程序的名字。

创建自定义对象

创建自定义对象的类要子类化一个已存在的类，比如说OdDbEntity或者OdDbObject。例如参考下面这个最简单的自定义对象的声明：

```
class EXCUSTOBJEXPORT ExCustObject : public OdDbObject
{
  public:
    // Declare default functions necessary for all database objects.
    ODDB_DECLARE_MEMBERS(ExCustObject);

    ExCustObject();
    virtual ~ExCustObject();

    // Required overrides
    OdResult dwgInFields(OdDbDwgFiler* pFiler);
    void dwgOutFields(OdDbDwgFiler* pFiler) const;
    OdResult dxfInFields(OdDbDxfFiler* pFiler);
    void dxfOutFields(OdDbDxfFiler* pFiler) const;
};
```

注意在一个自定义对象中必须重载dwgInFields，dwgOutFields，dxfInFields和dxfOutFields这几个函数，以便提供对.dwg和.dxf文件的支持。其他数据成员和成员函数可以按需加进来。

一个自定义类的实现必须调用ODRX_DXF_DEFINE_MEMBERS宏，它将实现类中由宏ODDB_DECLARE_MEMBERS声明所创建的函数。

```
ODRX_DXF_DEFINE_MEMBERS(ExCustObject,      // class name
OdDbObject,                                // parent class name
DBOBJECT_CONSTR,                           // creation macro
OdDb::vAC15,                               // dwg version
OdDb::kMRelease0,                          // maintenance release version
0,                                         // proxy flags
EXCUSTOBJECT,                              // DXF name
ExCustObjs|Description: Run-time Extension Example)    // Application name
```

注意这里应用程序名字：ExCustObjs，必须和包含这个对象实现的共享库的名字一样（这个例子中是ExCustObjs.drx）。

关于创建自定义对象的详细情况，参见“创建自定义对象”一节。

创建自定义图形实体

创建一个自定义图形实体与创建一个非图形数据对象相同，下列情况例外：

·OdDbEntity（或者是OdDbEntity的子类）应该作为自定义实体的父类使用。

·作为附加条件，必须重载dwg/dxf的I/O函数，一个自定义实体类也应该重载OdDbEntity::worldDraw（并选择性地重载OdDbEntity::viewportDraw）。

参见ExCustObjs例子，这是一个自定义数据库对象的完整例子，同时它可以由Teigha for .dwg文件系统框架渲染其自定义实体。

代理图形

根据PROXYGRAPHICS系统变量的值（0表示不保存，1表示保存），Teigha for .dwg文件系统可以选择性地将自定义实体的代理图形保存到文件中。可以使用OdDbDatabase::setPROXYGRAPHICS函数控制该设置。

创建客户程序

初始化和终止

所有的程序都应该坚持下面的初始化和终止规则：

· 一个用户程序应该在创建任何对象以及调用任何函数之前先调用odInitialize（）函数

· 传递给ocInitialize（）函数的OdDbSystemServices指针不能是NULL，并且必须到调用odUninitialize（）之前一直有效。如果这个对象被删除或者是在调用odUninitialize（）之前跳出作用域，就会发生异常。

· odUninitialize（）应该作为最后一个调用使用，并且在此之前所有的对象（除了传给odInitialize（）函数的OdDbSystemServices指针）都应该被删除。

主机应用服务

Teigha for .dwg 文件系统允许用户程序为具体的实现提供一定的服务，包括状态通知(对于文件加载、保存等等而言)，字体文件定位等等。这些服务由类 OdDbHostAppServices 定义。Teigha for .dwg 文件系统有一个默认的 OdDbHostAppServices 实现。
每一个数据库实例都有一个关联的 OdDbHostAppServices 对象（这使得在打开文件或者保存文件期间同时允许将通知发送给不同的 OdDbHostAppServices 实例）。这种关联可以有下列步骤定制。
·OdDbHostAppServices 的一个实例创建一个空数据库（由 OdDbHostAppServices::createDatabase()函数创建），那么与其关联的服务对象就是创建它的那个。
·一个客户调用 OdDbHOstAppServices::readFile()或者 OdDbHostAppServices::recoverFile()将使返回的数据库如愿关联到调用函数的 OdDbHostAppServices 实例。

监视文件加载和保存

提 供一个 OdDbHOstAppProgressMeter 类的实现可以监视文件的加载和保存。客户端应该重载 OdDbHostAppServices::newProgressMeter() 函数，并返回一个自定义的 OdDbHostAppProgressMeter 的实现。当加载或者保存操作需要一个新的进度指示的时候，C++库就会调用 OdDbHostAppServices::newProgressMeter() 函数，进度条将接收到一个 OdDbHostAppProgressMeter::start() 调用，这个函数初始化进度条，然后接收到 OdDbHostAppProgressMeter::setLimit()调用，这个函数设置进度条的整个时间数，设为 n。在此之后，进度条会收到其余 的 n 次对 OdDbHostAppProgress()的调用。当获得指定次数的调用时，进度条将会收到一个 OdDbHostAppProgressMeter::stop()的调用。

取消加载或者保存操作

客户端程 序可以通过提供 OdDbHostAppServices 和 OdDbHostAppProgressMeter 的自定义实现来终止一个加载、保存操作，然后 在 OdDbHostAppProgressMeter::meterProgress()函数中抛出异常。如果一个加载操作被取消了，那么返回的数据库智 能指针就会使 null。

数据库用例

OdDbDatabase 对象在内存中代表了一个.dwg 或.dxf 文件的内容，同时一个 OdDbDatabase 对象也是一次绘制中第一个访问点。下面的代码段演示了如何将一个绘制文件加进内存中：

```
OdRxObjectImpl<ExHostAppServices> svcs;        // Create a custom application

odInitialize(&svcs);                                    // Initialization

OdDbDatabasePtr pDb;                                    // A smart pointer to a database

pDb = svcs.readFile("drawing1.dwg");            // Create a database and load the drawing into it

// Do something with the database.

pDb.release();                                          // Delete the database before uninitialization

odUninitialize();                                       // Termination
```

创建数据库的那个 OdDbHostAppServices 实例(使用 OdDbHostAppServices::readFile(),OdDbHostAppServices::createDatabase()等等)将会与数据库实例相关联。所以在上面的例子中，ExHostAppServices 实例 svcs 将会与新创建的数据库关联，而后对 newProgressMeter 的调用将会先于 "drawing1.dwg" 的调用。返回的 OdDbHostAppProgressMeter 将会收到关于文件加载的通知。

如果你想用多线程编程，你需要为每一个线程差 ungjainyigeOdDbHostAppServices 实例。否则，来自多线程的进程通知将会发送到相同的服务对象。

保存 R12 文件时控制曲线仿真

在保存为 R12或者更早的文件时，用户可以通过调用 Teigh for .dwg 文件系统包含的2个函数 OdDbHostAppServices::setR12SaveDeviation 和 OdDbHostAppServices::setR12SaveAccuracy 来控制椭圆和样条曲线如何弯曲成简单实体，这些函数在 SysVarDef.h 中由 REG_VAR_DEF 宏定义，特别地：

·曲线“精度”是1/4椭圆的段数，或者是样条曲线上控制点的个数

·如果“偏差”是0（默认），精度值使用默认值8.

·如果偏差不是0,那么曲线偏差值
·如果间隔不是0,那么曲线分割就要使用曲线间隔值。如果按照间隔得到的点数少于精度要求的点数，那么曲线会再次分割以保证得到最终点的个数至少是精度值要求的点数。

访问数据库对象

这一节描述如何从数据库中得到不同类型的数据。

访问系统头变量

头变量可以直接通过OdDbDatabase实例访问，例如：

```
pDb->setLTSCALE(1.2);

std::cout << "LTSCALE:" << pDb->getLTSCALE() << std::endl;
```

一般情况下，所有头变量的访问函数形式都是：OdDbDatabase::getVARNAME() 和 OdDbDatabase::setVARNAME()。

访问表入口

打开表对象会返回表数据，然后使用一个迭代器就可以进入表的入口。一下函数遍历图层表并且打印出每个图层的名字，假设pDb是一个可用的数据库，os是一个输出流。

```
void DbDumper::dumpLayers(OdDbDatabase* pDb, int indent)

{

  /************************************************************************/

  /* Get a SmartPointer to the LayerTable                                 */

  /************************************************************************/

  OdDbLayerTablePtr pTable = pDb->getLayerTableId().safeOpenObject();


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

```
/* Dump the Description                                                    */
/*************************************************************************/
writeLine();
writeLine(indent++, toString(pTable->desc()));

/*************************************************************************/
/* Get a SmartPointer to a new SymbolTableIterator                         */
/*************************************************************************/
OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

/*************************************************************************/
/* Step through the LayerTable                                             */
/*************************************************************************/
for (pIter->start(); !pIter->done(); pIter->step())
{
  /***********************************************************************/
  /* Open the LayerTableRecord for Reading                                 */
  /***********************************************************************/
  OdDbLayerTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

  /***********************************************************************/
  /* Dump the LayerTableRecord                                             */
  /***********************************************************************/
  writeLine();
  writeLine(indent, toString(pRecord->desc()));
  writeLine(indent, OD_T("Name"),          toString(pRecord->getName()));
  writeLine(indent, OD_T("In Use"),        toString(pRecord->isInUse()));
```

```
      writeLine(indent, OD_T("On"),                    toString(!pRecord->isOff()));
      writeLine(indent, OD_T("Frozen"),                toString(pRecord->isFrozen()));
      writeLine(indent, OD_T("Locked"),                toString(pRecord->isLocked()));
      writeLine(indent, OD_T("Color"),                 toString(pRecord->color()));
      writeLine(indent, OD_T("Linetype"),              toString(pRecord->linetypeObjectId()));
      writeLine(indent, OD_T("Lineweight"),            toString(pRecord->lineWeight()));
      writeLine(indent, OD_T("Plotstyle"),             toString(pRecord->plotStyleName()));
      writeLine(indent, OD_T("Plottable"),             toString(pRecord->isPlottable()));
      writeLine(indent, OD_T("New VP Freeze"),         toString(pRecord->VPDFLT()));
      dumpSymbolTableRecord(pRecord, indent);
    }
}

/*****************************************************************************/
/* Dump the LinetypeTable                                                    */
/*****************************************************************************/
void DbDumper::dumpLinetypes(OdDbDatabase* pDb, int indent)
{
  /*****************************************************************************/
  /* Get a SmartPointer to the LinetypeTable                                   */
  /*****************************************************************************/
  OdDbLinetypeTablePtr pTable = pDb->getLinetypeTableId().safeOpenObject();

  /*****************************************************************************/
  /* Dump the Description                                                      */
  /*****************************************************************************/
  writeLine();
```

```
writeLine(indent++, toString(pTable->desc()));

/************************************************************************/
/* Get a SmartPointer to a new SymbolTableIterator                      */
/************************************************************************/
OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

/************************************************************************/
/* Step through the LinetypeTable                                       */
/************************************************************************/
for (pIter->start(); !pIter->done(); pIter->step())
{
   /************************************************************************/
   /* Open the LinetypeTableRecord for Reading                             */
   /************************************************************************/
   OdDbLinetypeTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

   /************************************************************************/
   /* Dump the LinetypeTableRecord                                         */
   /************************************************************************/
   writeLine();
   writeLine(indent, toString(pRecord->desc()));
   /************************************************************************/
   /* Dump the first line of record as in ACAD.LIN                         */
   /************************************************************************/
   OdString buffer;
   buffer = OdString(OD_T("*")) + pRecord->getName();
```

```
    if (pRecord->comments() != OD_T(""))
    {
      buffer = buffer +    OD_T(",") + pRecord->comments();
    }
    writeLine(indent, buffer);

    /*****************************************************************/
    /* Dump the second line of record as in ACAD.LIN                 */
    /*****************************************************************/
    if (pRecord->numDashes())
    {
      buffer = (pRecord->isScaledToFit() ? OD_T("S") : OD_T("A"));
      for (int i=0; i < pRecord->numDashes(); i++) {
          buffer = buffer + OD_T(",") + toString(pRecord->dashLengthAt(i));
          OdUInt16 shapeNumber    = pRecord->shapeNumberAt(i);
          OdString text = pRecord->textAt(i);

          /*************************************************************/
          /* Dump the Complex Line                                     */
          /*************************************************************/
          if (shapeNumber || text != OD_T(""))
          {
            OdDbTextStyleTableRecordPtr pTextStyle =
pRecord->shapeStyleAt(i).safeOpenObject();
            if (shapeNumber)
            {
              buffer = buffer + OD_T(",[") + toString(shapeNumber) + OD_T(",") +
pTextStyle->fileName();
```

```
                }
                else
                {
                   buffer = buffer + OD_T(",[") + toString(text) + OD_T(",") +
pTextStyle->getName();
                }
                if (pRecord->shapeScaleAt(i))
                {
                   buffer = buffer + OD_T(",S") + toString(pRecord->shapeScaleAt(i));
                }
                if (pRecord->shapeRotationAt(i))
                {
                   buffer = buffer + OD_T(",R") + toDegreeString(pRecord->shapeRotationAt(i));
                }
                if (pRecord->shapeOffsetAt(i).x)
                {
                   buffer = buffer + OD_T(",X") + toString(pRecord->shapeOffsetAt(i).x);
                }
                if (pRecord->shapeOffsetAt(i).y)
                {
                   buffer = buffer +    OD_T(",Y") + toString(pRecord->shapeOffsetAt(i).y);
                }
                buffer = buffer + OD_T("]");
             }
          }
          writeLine(indent, buffer);
       }
    dumpSymbolTableRecord(pRecord, indent);
```

```
    }
}
/************************************************************************/
/* Dump the TextStyleTable                                              */
/************************************************************************/
void DbDumper::dumpTextStyles(OdDbDatabase* pDb, int indent)
{
    /************************************************************************/
    /* Get a SmartPointer to the TextStyleTable                             */
    /************************************************************************/
    OdDbTextStyleTablePtr pTable = pDb->getTextStyleTableId().safeOpenObject();

    /************************************************************************/
    /* Dump the Description                                                 */
    /************************************************************************/
    writeLine();
    writeLine(indent++, toString(pTable->desc()));

    /************************************************************************/
    /* Get a SmartPointer to a new SymbolTableIterator                      */
    /************************************************************************/
    OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

    /************************************************************************/
    /* Step through the TextStyleTable                                      */
    /************************************************************************/
    for (pIter->start(); !pIter->done(); pIter->step())
```

```
{
  /******************************************************************/
  /* Open the TextStyleTableRecord for Reading                      */
  /******************************************************************/
  OdDbTextStyleTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

  /******************************************************************/
  /* Dump the TextStyleTableRecord                                  */
  /******************************************************************/
  writeLine();
  writeLine(indent, toString(pRecord->desc()));
  writeLine(indent, OD_T("Name"),                 toString(pRecord->getName()));
  writeLine(indent, OD_T("Shape File"),           toString(pRecord->isShapeFile()));
  writeLine(indent, OD_T("Text Height"),          toString(pRecord->textSize()));
  writeLine(indent, OD_T("Width Factor"),         toString(pRecord->xScale()));
  writeLine(indent, OD_T("Obliquing Angle"),    toDegreeString(pRecord->obliquingAngle()));
  writeLine(indent, OD_T("Backwards"),            toString(pRecord->isBackwards()));
  writeLine(indent, OD_T("Vertical"),             toString(pRecord->isVertical()));
  writeLine(indent, OD_T("Upside Down"),          toString(pRecord->isUpsideDown()));
  writeLine(indent, OD_T("Filename"),             shortenPath(pRecord->fileName()));
  writeLine(indent, OD_T("BigFont Filename"),   shortenPath(pRecord->bigFontFileName()));

  OdString typeface;
  bool bold;
  bool italic;
  int charset;
  int pitchAndFamily;
```

```
      pRecord->font(typeface, bold, italic, charset, pitchAndFamily);
      writeLine(indent, OD_T("Typeface"),                toString(typeface));
      writeLine(indent, OD_T("Character Set"),           toString(charset));
      writeLine(indent, OD_T("Bold"),                    toString(bold));
      writeLine(indent, OD_T("Italic"),                  toString(italic));
      writeLine(indent, OD_T("Font Pitch & Family"),  toHexString(pitchAndFamily));
      dumpSymbolTableRecord(pRecord, indent);
    }
  }
/************************************************************************/
/* Dump the DimStyleTable                                               */
/************************************************************************/
void DbDumper::dumpDimStyles(OdDbDatabase* pDb, int indent)
{
  /************************************************************************/
  /* Get a SmartPointer to the DimStyleTable                              */
  /************************************************************************/
  OdDbDimStyleTablePtr pTable = pDb->getDimStyleTableId().safeOpenObject();

  /************************************************************************/
  /* Dump the Description                                                 */
  /************************************************************************/
  writeLine();
  writeLine(indent++, toString(pTable->desc()));

  /************************************************************************/
  /* Get a SmartPointer to a new SymbolTableIterator                      */
```

```
  /******************************************************************/
  OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

  /******************************************************************/
  /* Step through the DimStyleTable                                 */
  /******************************************************************/
  for (pIter->start(); !pIter->done(); pIter->step())
  {
    /****************************************************************/
    /* Open the DimStyleTableRecord for Reading                     */
    /****************************************************************/
    OdDbDimStyleTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

    /****************************************************************/
    /* Dump the DimStyleTableRecord                                 */
    /****************************************************************/
    writeLine();
    writeLine(indent, toString(pRecord->desc()));
    writeLine(indent, OD_T("Name"), toString(pRecord->getName()));
    writeLine(indent, OD_T("Arc Symbol"), toArcSymbolTypeString(pRecord->getArcSymbolType()));

    OdCmColor bgrndTxtColor;
    OdUInt16 bgrndTxtFlags = pRecord->getBgrndTxtColor(bgrndTxtColor);
    writeLine(indent, OD_T("Background Text Color"), toString(bgrndTxtColor));
    writeLine(indent, OD_T("BackgroundText Flags"), toString(bgrndTxtFlags));
    writeLine(indent, OD_T("Extension Line 1 Linetype"), toString(pRecord->getDimExt1Linetype()));
```

```
    writeLine(indent, OD_T("Extension Line 2 Linetype"), toString(pRecord->getDimExt2Linetype()));

    writeLine(indent, OD_T("Dimension Line Linetype"), toString(pRecord->getDimExt2Linetype()));

    writeLine(indent, OD_T("Extension Line Fixed Len"), toString(pRecord->getExtLineFixLen()));

    writeLine(indent, OD_T("Extension Line Fixed Len Enable"), toString(pRecord->getExtLineFixLenEnable()));

    writeLine(indent, OD_T("Flip Arrow"), toString(pRecord->getFlipArrow()));

    writeLine(indent, OD_T("Jog Angle"), toDegreeString(pRecord->getJogAngle()));

    writeLine(indent, OD_T("Modified For Recompute"), toString(pRecord->isModifiedForRecompute()));

    writeLine(indent, OD_T("DIMADEC"),      toString(pRecord->dimadec()));

    writeLine(indent, OD_T("DIMALT"),       toString(pRecord->dimalt()));

    writeLine(indent, OD_T("DIMALTD"),      toString(pRecord->dimaltd()));

    writeLine(indent, OD_T("DIMALTF"),      toString(pRecord->dimaltf()));

    writeLine(indent, OD_T("DIMALTRND"),    toString(pRecord->dimaltrnd()));

    writeLine(indent, OD_T("DIMALTTD"),     toString(pRecord->dimalttd()));

    writeLine(indent, OD_T("DIMALTTZ"),     toString(pRecord->dimalttz()));

    writeLine(indent, OD_T("DIMALTU"),      toString(pRecord->dimaltu()));

    writeLine(indent, OD_T("DIMALTZ"),      toString(pRecord->dimaltz()));

    writeLine(indent, OD_T("DIMAPOST"),     toString(pRecord->dimapost()));

    writeLine(indent, OD_T("DIMASZ"),       toString(pRecord->dimasz()));

    writeLine(indent, OD_T("DIMATFIT"),     toString(pRecord->dimatfit()));

    writeLine(indent, OD_T("DIMAUNIT"),     toString(pRecord->dimaunit()));

    writeLine(indent, OD_T("DIMAZIN"),      toString(pRecord->dimazin()));

    writeLine(indent, OD_T("DIMBLK"),       toString(pRecord->dimblk()));

    writeLine(indent, OD_T("DIMBLK1"),      toString(pRecord->dimblk1()));
```

```
writeLine(indent, OD_T("DIMBLK2"),          toString(pRecord->dimblk2()));
writeLine(indent, OD_T("DIMCEN"),           toString(pRecord->dimcen()));
writeLine(indent, OD_T("DIMCLRD"),          toString(pRecord->dimclrd()));
writeLine(indent, OD_T("DIMCLRE"),          toString(pRecord->dimclre()));
writeLine(indent, OD_T("DIMCLRT"),          toString(pRecord->dimclrt()));
writeLine(indent, OD_T("DIMDEC"),           toString(pRecord->dimdec()));
writeLine(indent, OD_T("DIMDLE"),           toString(pRecord->dimdle()));
writeLine(indent, OD_T("DIMDLI"),           toString(pRecord->dimdli()));
writeLine(indent, OD_T("DIMDSEP"),          toString(pRecord->dimdsep()));
writeLine(indent, OD_T("DIMEXE"),           toString(pRecord->dimexe()));
writeLine(indent, OD_T("DIMEXO"),           toString(pRecord->dimexo()));
writeLine(indent, OD_T("DIMFRAC"),          toString(pRecord->dimfrac()));
writeLine(indent, OD_T("DIMGAP"),           toString(pRecord->dimgap()));
writeLine(indent, OD_T("DIMJUST"),          toString(pRecord->dimjust()));
writeLine(indent, OD_T("DIMLDRBLK"),        toString(pRecord->dimldrblk()));
writeLine(indent, OD_T("DIMLFAC"),          toString(pRecord->dimlfac()));
writeLine(indent, OD_T("DIMLIM"),           toString(pRecord->dimlim()));
writeLine(indent, OD_T("DIMLUNIT"),         toString(pRecord->dimlunit()));
writeLine(indent, OD_T("DIMLWD"),           toString(pRecord->dimlwd()));
writeLine(indent, OD_T("DIMLWE"),           toString(pRecord->dimlwe()));
writeLine(indent, OD_T("DIMPOST"),          toString(pRecord->dimpost()));
writeLine(indent, OD_T("DIMRND"),           toString(pRecord->dimrnd()));
writeLine(indent, OD_T("DIMSAH"),           toString(pRecord->dimsah()));
writeLine(indent, OD_T("DIMSCALE"),         toString(pRecord->dimscale()));
writeLine(indent, OD_T("DIMSD1"),           toString(pRecord->dimsd1()));
writeLine(indent, OD_T("DIMSD2"),           toString(pRecord->dimsd2()));
writeLine(indent, OD_T("DIMSE1"),           toString(pRecord->dimse1()));
```

```
        writeLine(indent, OD_T("DIMSE2"),              toString(pRecord->dimse2()));
        writeLine(indent, OD_T("DIMSOXD"),              toString(pRecord->dimsoxd()));
        writeLine(indent, OD_T("DIMTAD"),               toString(pRecord->dimtad()));
        writeLine(indent, OD_T("DIMTDEC"),              toString(pRecord->dimtdec()));
        writeLine(indent, OD_T("DIMTFAC"),              toString(pRecord->dimtfac()));
        writeLine(indent, OD_T("DIMTIH"),              toString(pRecord->dimtih()));
        writeLine(indent, OD_T("DIMTIX"),              toString(pRecord->dimtix()));
        writeLine(indent, OD_T("DIMTM"),                toString(pRecord->dimtm()));
        writeLine(indent, OD_T("DIMTOFL"),              toString(pRecord->dimtofl()));
        writeLine(indent, OD_T("DIMTOH"),               toString(pRecord->dimtoh()));
        writeLine(indent, OD_T("DIMTOL"),               toString(pRecord->dimtol()));
        writeLine(indent, OD_T("DIMTOLJ"),             toString(pRecord->dimtolj()));
        writeLine(indent, OD_T("DIMTP"),               toString(pRecord->dimtp()));
        writeLine(indent, OD_T("DIMTSZ"),              toString(pRecord->dimtsz()));
        writeLine(indent, OD_T("DIMTVP"),               toString(pRecord->dimtvp()));
        writeLine(indent, OD_T("DIMTXSTY"),             toString(pRecord->dimtxsty()));
        writeLine(indent, OD_T("DIMTXT"),               toString(pRecord->dimtxt()));
        writeLine(indent, OD_T("DIMTZIN"),             toString(pRecord->dimtzin()));
        writeLine(indent, OD_T("DIMUPT"),               toString(pRecord->dimupt()));
        writeLine(indent, OD_T("DIMZIN"),              toString(pRecord->dimzin()));

        dumpSymbolTableRecord(pRecord, indent);
    }
}

/*****************************************************************/
/* Dump the RegAppTable                                          */
```

```
/************************************************************************/
void DbDumper::dumpRegApps(OdDbDatabase* pDb, int indent)
{
  /************************************************************************/
  /* Get a SmartPointer to the RegAppTable                                */
  /************************************************************************/
  OdDbRegAppTablePtr pTable = pDb->getRegAppTableId().safeOpenObject();

  /************************************************************************/
  /* Dump the Description                                                 */
  /************************************************************************/
  writeLine();
  writeLine(indent++, toString(pTable->desc()));

  /************************************************************************/
  /* Get a SmartPointer to a new SymbolTableIterator                      */
  /************************************************************************/
  OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

  /************************************************************************/
  /* Step through the RegAppTable                                         */
  /************************************************************************/
  for (pIter->start(); !pIter->done(); pIter->step())
  {
    /************************************************************************/
    /* Open the RegAppTableRecord for Reading                               */
    /************************************************************************/
```

```
        OdDbRegAppTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

        /******************************************************************/
        /* Dump the RegAppTableRecord                                     */
        /******************************************************************/
        writeLine();
        writeLine(indent, toString(pRecord->desc()));
        writeLine(indent, OD_T("Name"), toString(pRecord->getName()));
    }
}

/**********************************************************************/
/* Dump the AbstractViewTableRecord                                   */
/**********************************************************************/
void DbDumper::dumpAbstractViewTableRecord(OdDbAbstractViewTableRecordPtr pView, int indent) {

        /******************************************************************/
        /* Dump the AbstractViewTableRecord                               */
        /******************************************************************/
        writeLine(indent, OD_T("Back Clip Dist"),          toString(pView->backClipDistance()));
        writeLine(indent, OD_T("Back Clip Enabled"),     toString(pView->backClipEnabled()));
        writeLine(indent, OD_T("Front Clip Dist"),         toString(pView->frontClipDistance()));
        writeLine(indent, OD_T("Front Clip Enabled"),    toString(pView->frontClipEnabled()));
        writeLine(indent, OD_T("Front Clip at Eye"),     toString(pView->frontClipAtEye()));
        writeLine(indent, OD_T("Elevation"),               toString(pView->elevation()));
        writeLine(indent, OD_T("Height"),                  toString(pView->height()));
        writeLine(indent, OD_T("Width"),                   toString(pView->width()));
```

```
    writeLine(indent, OD_T("Lens Length"),          toString(pView->lensLength()));
    writeLine(indent, OD_T("Render Mode"),          toString(pView->renderMode()));
    writeLine(indent, OD_T("Perspective"),          toString(pView->perspectiveEnabled()));
    writeLine(indent, OD_T("UCS Name"),             toString(pView->ucsName()));

    OdDb::OrthographicView orthoUCS;
    writeLine(indent, OD_T("UCS Orthographic"),
toString(pView->isUcsOrthographic(orthoUCS)));
    writeLine(indent, OD_T("Orthographic UCS"),  toString(orthoUCS));
    OdGePoint3d origin;
    OdGeVector3d xAxis;
    OdGeVector3d yAxis;
    pView->getUcs(origin, xAxis, yAxis);
    writeLine(indent, OD_T("UCS Origin"),           toString(origin));
    writeLine(indent, OD_T("UCS x-Axis"),           toString(xAxis));
    writeLine(indent, OD_T("UCS y-Axis"),           toString(yAxis));
    writeLine(indent, OD_T("Target"),               toString(pView->target()));
    writeLine(indent, OD_T("View Direction"),       toString(pView->viewDirection()));
    writeLine(indent, OD_T("Twist Angle"),          toDegreeString(pView->viewTwist()));
    dumpSymbolTableRecord(pView, indent);
}

/************************************************************************/
/* Dump the ViewportTable                                               */
/************************************************************************/
void DbDumper::dumpViewports(OdDbDatabase* pDb, int indent)
{
  /************************************************************************/
```

```
/* Get a SmartPointer to the ViewportTable                                  */
/***************************************************************************/
OdDbViewportTablePtr pTable = pDb->getViewportTableId().safeOpenObject();

/***************************************************************************/
/* Dump the Description                                                    */
/***************************************************************************/
writeLine();
writeLine(indent++, toString(pTable->desc()));

/***************************************************************************/
/* Get a SmartPointer to a new SymbolTableIterator                         */
/***************************************************************************/
OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

/***************************************************************************/
/* Step through the ViewportTable                                          */
/***************************************************************************/
for (pIter->start(); !pIter->done(); pIter->step())
{
  /*************************************************************************/
  /* Open the ViewportTableRecord for Reading                              */
  /*************************************************************************/
  OdDbViewportTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

  /*************************************************************************/
  /* Dump the ViewportTableRecord                                          */
```

```
        /*****************************************************************/
        writeLine();
        writeLine(indent, toString(pRecord->desc()));
        writeLine(indent, OD_T("Name"),                    toString(pRecord->getName()));
        writeLine(indent, OD_T("Circle Sides"),          toString(pRecord->circleSides()));
        writeLine(indent, OD_T("Fast Zooms Enabled"),toString(pRecord->fastZoomsEnabled()));
        writeLine(indent, OD_T("Grid Enabled"),          toString(pRecord->gridEnabled()));
        writeLine(indent, OD_T("Grid Increments"),     toString(pRecord->gridIncrements()));
        writeLine(indent, OD_T("Icon at Origin"),       toString(pRecord->iconAtOrigin()));
        writeLine(indent, OD_T("Icon Enabled"),          toString(pRecord->iconEnabled()));
        writeLine(indent, OD_T("Iso snap Enabled"),    toString(pRecord->isometricSnapEnabled()));
        writeLine(indent, OD_T("Iso Snap Pair"),         toString(pRecord->snapPair()));
        writeLine(indent,                OD_T("UCS                Saved                w/Vport"),
toString(pRecord->isUcsSavedWithViewport()));
        writeLine(indent, OD_T("UCS follow"),             toString(pRecord->ucsFollowMode()));
        writeLine(indent, OD_T("Lower-Left Corner"), toString(pRecord->lowerLeftCorner()));
        writeLine(indent, OD_T("Upper-Right Corner"),toString(pRecord->upperRightCorner()));
        writeLine(indent, OD_T("Snap Angle"),             toDegreeString(pRecord->snapAngle()));
        writeLine(indent, OD_T("Snap Base"),               toString(pRecord->snapBase()));
        writeLine(indent, OD_T("Snap Enabled"),          toString(pRecord->snapEnabled()));
        writeLine(indent, OD_T("Snap Increments"),     toString(pRecord->snapIncrements()));
        dumpAbstractViewTableRecord(pRecord, indent);
    }
}

/*****************************************************************/
/* Dump the ViewTable                                            */
/*****************************************************************/
```

```
void DbDumper::dumpViews(OdDbDatabase* pDb, int indent)
{
  /************************************************************************/
  /* Get a SmartPointer to the ViewTable                                  */
  /************************************************************************/
  OdDbViewTablePtr pTable = pDb->getViewTableId().safeOpenObject();

  /************************************************************************/
  /* Dump the Description                                                 */
  /************************************************************************/
  writeLine();
  writeLine(indent++, toString(pTable->desc()));

  /************************************************************************/
  /* Get a SmartPointer to a new SymbolTableIterator                      */
  /************************************************************************/
  OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

  /************************************************************************/
  /* Step through the ViewTable                                           */
  /************************************************************************/
  for (pIter->start(); !pIter->done(); pIter->step())
  {
    /************************************************************************/
    /* Open the ViewTableRecord for Reading                                 */
    /************************************************************************/
    OdDbViewTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();
```

```
    /******************************************************************/
    /* Dump the ViewTableRecord                                       */
    /******************************************************************/
    writeLine();
    writeLine(indent, toString(pRecord->desc()));
    writeLine(indent, OD_T("Name"),                toString(pRecord->getName()));
    writeLine(indent, OD_T("Category Name"),       toString(pRecord->getCategoryName()));
    writeLine(indent, OD_T("Layer State"),         toString(pRecord->getLayerState()));

    OdString layoutName(OD_T(""));
    if (!pRecord->getLayout().isNull())
    {
      OdDbLayoutPtr pLayout = pRecord->getLayout().safeOpenObject();
      layoutName = pLayout->getLayoutName();
    }
    writeLine(indent, OD_T("Layout Name"),         toString(layoutName));
    writeLine(indent, OD_T("PaperSpace View"),     toString(pRecord->isPaperspaceView()));
    writeLine(indent, OD_T("Associated UCS"), toString(pRecord->isUcsAssociatedToView()));
    writeLine(indent, OD_T("PaperSpace View"), toString(pRecord->isViewAssociatedToViewport()));
    dumpAbstractViewTableRecord(pRecord, indent);
  }
}
/******************************************************************/
/* Dump the MlineStyle Dictionary                                 */
/******************************************************************/
```

```
void DbDumper::dumpMLineStyles(OdDbDatabase* pDb, int indent)
{
  OdDbDictionaryPtr pDictionary = pDb->getMLStyleDictionaryId().safeOpenObject();

  /******************************************************************/
  /* Dump the Description                                           */
  /******************************************************************/
  writeLine();
  writeLine(indent++, toString(pDictionary->desc()));

  /******************************************************************/
  /* Get a SmartPointer to a new DictionaryIterator                 */
  /******************************************************************/
  OdDbDictionaryIteratorPtr pIter = pDictionary->newIterator();

  /******************************************************************/
  /* Step through the MlineStyle dictionary                         */
  /******************************************************************/
  for (; !pIter->done(); pIter->next())
  {
    OdDbMlineStylePtr pEntry = pIter->objectId().safeOpenObject();
    if(pEntry.isNull())
      continue;
    /******************************************************************/
    /* Dump the MLineStyle dictionary entry                           */
    /******************************************************************/
    writeLine();
```

```
writeLine(indent, toString(pEntry->desc()));
writeLine(indent, OD_T("Name"),                 toString(pEntry->name()));
writeLine(indent, OD_T("Description"),          toString(pEntry->description()));
writeLine(indent, OD_T("Start Angle"),          toDegreeString(pEntry->startAngle()));
writeLine(indent, OD_T("End Angle"),            toDegreeString(pEntry->endAngle()));
writeLine(indent, OD_T("Start Inner Arcs"),     toString(pEntry->startInnerArcs()));
writeLine(indent, OD_T("End Inner Arcs"),       toString(pEntry->endInnerArcs()));
writeLine(indent, OD_T("Start Round Cap"),      toString(pEntry->startRoundCap()));
writeLine(indent, OD_T("End Round Cap"),        toString(pEntry->endRoundCap()));
writeLine(indent, OD_T("Start Square Cap"),     toString(pEntry->startRoundCap()));
writeLine(indent, OD_T("End Square Cap"),       toString(pEntry->endRoundCap()));
writeLine(indent, OD_T("Show Miters"),          toString(pEntry->showMiters()));
/*****************************************************************/
/* Dump the elements                                             */
/*****************************************************************/
if (pEntry->numElements())
{
   writeLine(indent, OD_T("Elements:"));
}
for (int i = 0; i < pEntry->numElements(); i++)
{
   double offset;
   OdCmColor color;
   OdDbObjectId linetypeId;
   pEntry->getElementAt(i, offset, color,linetypeId);
   writeLine(indent,       OD_T("Index"),          toString(i));
   writeLine(indent + 1, OD_T("Offset"),           toString(offset));
```

```
            writeLine(indent + 1, OD_T("Color"),                    toString(color));
            writeLine(indent + 1, OD_T("Linetype"),                 toString(linetypeId));
        }
    }
}
/************************************************************************/
/* Dump the UCSTable                                                    */
/************************************************************************/
void DbDumper::dumpUCSTable(OdDbDatabase* pDb, int indent)
{
    /************************************************************************/
    /* Get a SmartPointer to the UCSTable                                   */
    /************************************************************************/
    OdDbUCSTablePtr pTable = pDb->getUCSTableId().safeOpenObject();

    /************************************************************************/
    /* Dump the Description                                                 */
    /************************************************************************/
    writeLine();
    writeLine(indent++, toString(pTable->desc()));

    /************************************************************************/
    /* Get a SmartPointer to a new SymbolTableIterator                      */
    /************************************************************************/
    OdDbSymbolTableIteratorPtr pIter = pTable->newIterator();

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

```
  /* Step through the UCSTable                                              */
  /*************************************************************************/
  for (pIter->start(); !pIter->done(); pIter->step())
  {
    /***********************************************************************/
    /* Open the UCSTableRecord for Reading                                 */
    /***********************************************************************/
    OdDbUCSTableRecordPtr pRecord = pIter->getRecordId().safeOpenObject();

    /***********************************************************************/
    /* Dump the UCSTableRecord                                             */
    /***********************************************************************/
    writeLine();
    writeLine(indent, toString(pRecord->desc()));
    writeLine(indent, OD_T("Name"),          toString(pRecord->getName()));
    writeLine(indent, OD_T("UCS Origin"),    toString(pRecord->origin()));
    writeLine(indent, OD_T("UCS x-Axis"),    toString(pRecord->xAxis()));
    writeLine(indent, OD_T("UCS y-Axis"),    toString(pRecord->yAxis()));
    dumpSymbolTableRecord(pRecord, indent);
  }
}

/*************************************************************************/
/* Dump the Entity                                                       */
/*************************************************************************/
void DbDumper::dumpEntity(OdDbObjectId id, int indent)
{
```

```
/******************************************************************/
/* Get a SmartPointer to the Entity                               */
/******************************************************************/
OdDbEntityPtr pEnt = id.safeOpenObject();

/******************************************************************/
/* Retrieve the Protocol Extension registered for this object type*/
/******************************************************************/
OdSmartPtr<OdDbEntity_Dumper> pEntDumper = pEnt;

/******************************************************************/
/* Dump the entity                                                */
/******************************************************************/
writeLine();
pEntDumper->dump(pEnt, indent);

/******************************************************************/
/* Dump the Xdata                                                 */
/******************************************************************/
dumpXdata(pEnt->xData(), indent);

/******************************************************************/
/* Dump the Extension Dictionary                                  */
/******************************************************************/
if (!pEnt->extensionDictionary().isNull())
{
   dumpObject(pEnt->extensionDictionary(), OD_T("ACAD_XDICTIONARY"), indent);
```

```
    }
}

/************************************************************************/
/* Dump the BlockTable                                                  */
/************************************************************************/
void DbDumper::dumpBlocks(OdDbDatabase* pDb, int indent)
{
    /************************************************************************/
    /* Get a SmartPointer to the BlockTable                                 */
    /************************************************************************/
    OdDbBlockTablePtr pTable = pDb->getBlockTableId().safeOpenObject();

    /************************************************************************/
    /* Dump the Description                                                 */
    /************************************************************************/
    writeLine();
    writeLine(indent++, toString(pTable->desc()));

    /************************************************************************/
    /* Get a SmartPointer to a new SymbolTableIterator                      */
    /************************************************************************/
    OdDbSymbolTableIteratorPtr pBlkIter = pTable->newIterator();

    /************************************************************************/
    /* Step through the BlockTable                                          */
    /************************************************************************/
```

```
  for (pBlkIter->start(); ! pBlkIter->done(); pBlkIter->step())
  {
    /*****************************************************************/
    /* Open the BlockTableRecord for Reading                         */
    /*****************************************************************/
    OdDbBlockTableRecordPtr pBlock = pBlkIter->getRecordId().safeOpenObject();

    /*****************************************************************/
    /* Dump the BlockTableRecord                                     */
    /*****************************************************************/
    writeLine();
    writeLine(indent, toString(pBlock->desc()));
    writeLine(indent+1, OD_T("Name"), toString(pBlock->getName()));
    writeLine(indent+1, OD_T("Anonymous"), toString(pBlock->isAnonymous()));
    writeLine(indent+1, OD_T("Comments"), toString(pBlock->comments()));
    writeLine(indent+1, OD_T("Origin"), toString(pBlock->origin()));
    writeLine(indent+1, OD_T("Block Insert Units"), toString(pBlock->blockInsertUnits()));
    writeLine(indent+1, OD_T("Block Scaling"), toString(pBlock->blockScaling()));
    writeLine(indent+1, OD_T("Explodable"), toString(pBlock->explodable()));

    OdGeExtents3d extents;
    if (eOk == pBlock->getGeomExtents(extents)) {
      writeLine(indent+1, OD_T("Min Extents"), toString(extents.minPoint()));
      writeLine(indent+1, OD_T("Max Extents"), toString(extents.maxPoint()));
```

```
    }

    writeLine(indent+1, OD_T("Layout"), toString(pBlock->isLayout()));

    writeLine(indent+1, OD_T("Has Attribute Definitions"), toString(pBlock->hasAttributeDefinitions()));

    writeLine(indent+1, OD_T("Xref Status"), toString(pBlock->xrefStatus()));

    if (pBlock->xrefStatus()!=OdDb::kXrfNotAnXref)

    {

      writeLine(indent+1, OD_T("Xref Path"), toString(pBlock->pathName()));

      writeLine(indent+1, OD_T("From Xref Attach"), toString(pBlock->isFromExternalReference()));

      writeLine(indent+1, OD_T("From Xref Overlay"), toString(pBlock->isFromOverlayReference()));

      writeLine(indent+1, OD_T("Xref Unloaded"), toString(pBlock->isUnloaded()));

    }

    /*****************************************************************/

    /* Get a SmartPointer to a new ObjectIterator                    */

    /*****************************************************************/

    OdDbObjectIteratorPtr pEntIter = pBlock->newIterator();


    /*****************************************************************/

    /* Step through the BlockTableRecord                             */

    /*****************************************************************/

    for (; !pEntIter->done(); pEntIter->step())

    {

     /*****************************************************************/

     /* Dump the Entity                                               */

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

```
        dumpEntity(pEntIter->objectId(), indent+1);
      }
    }
}

/************************************************************************/
/* Dump Xdata                                                           */
/************************************************************************/
void DbDumper::dumpXdata(OdResBuf* resbuf, int indent)
{
  if (!resbuf)
    return;

  writeLine(indent++, OD_T("Xdata:"));

  /************************************************************************/
  /* Step through the ResBuf chain                                        */
  /************************************************************************/
  for (; resbuf != 0; resbuf = resbuf->next())
  {
    int code = resbuf->restype();

    /************************************************************************/
    /* Convert resbuf->ResVal to a string                                   */
    /************************************************************************/
    OdString rightString(OD_T("???"));
    switch (OdDxfCode::_getType(code))
```

```
{
  case OdDxfCode::Name:
  case OdDxfCode::String:
  case OdDxfCode::LayerName:
    rightString = toString(resbuf->getString());
    break;

  case OdDxfCode::Bool:
    rightString = toString(resbuf->getBool());
    break;

  case OdDxfCode::Integer8:
    rightString = toString(resbuf->getInt8());
    break;

  case OdDxfCode::Integer16:
    rightString = toString(resbuf->getInt16());
    break;

  case OdDxfCode::Integer32:
    rightString = toString((int)resbuf->getInt32());
    break;

  case OdDxfCode::Double:
    rightString = toString(resbuf->getDouble());
    break;
```

```
case OdDxfCode::Angle:
    rightString = toDegreeString(resbuf->getDouble());
    break;

case OdDxfCode::Point:
    {
        rightString = toString(resbuf->getPoint3d());
    }
    break;

case OdDxfCode::BinaryChunk:
    rightString = OD_T("<Binary Data>");
    break;

case OdDxfCode::ObjectId:
case OdDxfCode::SoftPointerId:
case OdDxfCode::HardPointerId:
case OdDxfCode::SoftOwnershipId:
case OdDxfCode::HardOwnershipId:
case OdDxfCode::Handle:
    rightString = toString(resbuf->getHandle());
    break;

case OdDxfCode::Unknown:
default:
    rightString = OD_T("Unknown");
    break;
```

```
      }
      writeLine(indent,toString(code),rightString);
   }
}

/*****************************************************************/
/* Dump the Xref the full path to the Osnap entity               */
/*****************************************************************/
void DbDumper::dumpXrefFullSubentPath(OdDbXrefFullSubentPath& subEntPath, int indent)
{
   writeLine(indent, OD_T("Subentity Index"), toString((int)subEntPath.subentId().index()));
   writeLine(indent, OD_T("Subentity Type"), toString(subEntPath.subentId().type()));
   for (OdUInt32 j = 0; j < subEntPath.objectIds().size(); j++)
   {
      OdDbEntityPtr pEnt = subEntPath.objectIds()[j].openObject();
      if (!pEnt.isNull())
      {
         writeLine(indent, toString(pEnt->isA()), toString(pEnt->getDbHandle()));
      }
   }
}

/*****************************************************************/
/* Dump Object Snap Point Reference for an Associative Dimension */
/*****************************************************************/
void DbDumper::dumpOsnapPointRef(OdDbOsnapPointRefPtr pRef,    int index, int indent)
{
```

```
    writeLine(indent++, toString(pRef->isA()), toString(index));
    writeLine(indent, OD_T("Osnap Mode"), toString(pRef->osnapType()));
    writeLine(indent, OD_T("Near Osnap"),    toString(pRef->nearPointParam()));
    writeLine(indent, OD_T("Osnap Point"), toString(pRef->point()));

    writeLine(indent, OD_T("Main Entity"));
    dumpXrefFullSubentPath(pRef->mainEntity(), indent+1);

    writeLine(indent, OD_T("Intersect Entity"));
    dumpXrefFullSubentPath(pRef->intersectEntity(), indent + 1);

    if (pRef->lastPointRef())
    {
      writeLine(indent, OD_T("Last Point Referenced"));
      dumpOsnapPointRef(pRef->lastPointRef(), index, indent + 1);
    }
    else
    {
      writeLine(indent, OD_T("Last Point Referenced"), OD_T("Null"));
    }
}

/*****************************************************************************/
/* Dump an Associative dimension                                             */
/*****************************************************************************/
void DbDumper::dumpDimAssoc(OdDbObjectPtr pObject, const int indent)
{
```

```
    OdDbDimAssocPtr pDimAssoc = pObject;
    writeLine(indent, OD_T("Associative"),
          toString ((OdDbDimAssoc::AssocFlags) pDimAssoc->assocFlag()));
    writeLine(indent, OD_T("TransSpatial"), toString(pDimAssoc->isTransSpatial()));
    writeLine(indent, OD_T("Rotated Type"), toString(pDimAssoc->rotatedDimType()));

    for (int i = 0; i < OdDbDimAssoc::kMaxPointRefs; i++)
    {
      OdDbOsnapPointRefPtr pRef = pDimAssoc->pointRef((OdDbDimAssoc::AssocFlags) (1 << i));
      if (!pRef.isNull())
      {
        dumpOsnapPointRef(pRef, i, indent);
      }
      else
      {
        break;
      }
    }
}

/*****************************************************************/
/* Dump the object.                                              */
/*                                                               */
/* Dictionary objects are recursively dumped.                    */
/* XRecord objects are dumped.                                   */
/* DimAssoc objects are dumped.                                  */
/*****************************************************************/
```

```
void DbDumper::dumpObject(
                                    OdDbObjectId id,
                                    OdString itemName,
                                    int indent)
{
  /******************************************************************/
  /* Get a SmartPointer to the object                               */
  /******************************************************************/
  OdDbObjectPtr pObject = id.safeOpenObject();

  /******************************************************************/
  /* Dump the item name and class name                              */
  /******************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
  {
    writeLine();
  }
  writeLine(indent++, itemName, toString(pObject->isA()));

  /******************************************************************/
  /* Dispatch                                                       */
  /******************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
  {
    /****************************************************************/
    /* Dump the dictionary                                          */
    /****************************************************************/
```

```
    OdDbDictionaryPtr pDic = pObject;

    /************************************************************************/
    /* Get a SmartPointer to a new DictionaryIterator                       */
    /************************************************************************/
    OdDbDictionaryIteratorPtr pIter = pDic->newIterator();

    /************************************************************************/
    /* Step through the Dictionary                                          */
    /************************************************************************/
    for (; !pIter->done(); pIter->next())
    {
      /************************************************************************/
      /* Dump the Dictionary object                                           */
      /************************************************************************/
      // Dump each item in the dictionary.
      dumpObject(pIter->objectId(), pIter->name(), indent);
    }
  }
  else if (pObject->isKindOf(OdDbXrecord::desc()))
  {
    /************************************************************************/
    /* Dump an Xrecord                                                      */
    /************************************************************************/
    OdDbXrecordPtr pXRec = pObject;
    dumpXdata(pXRec->rbChain(), indent);
  }
```

```
  else if (pObject->isKindOf(OdDbDimAssoc::desc()))
  {
    /*****************************************************************/
    /* Dump an Associative dimension                                 */
    /*****************************************************************/
    dumpDimAssoc(pObject, indent);
  }
}

/*****************************************************************/
/* Dump the database                                             */
/*****************************************************************/
void DbDumper::dump(OdDbDatabase* pDb, int indent)
{
  dumpHeader           (pDb, indent);
  dumpLayers           (pDb, indent);
  dumpLinetypes      (pDb, indent);
  dumpTextStyles    (pDb, indent);
  dumpDimStyles      (pDb, indent);
  dumpRegApps         (pDb, indent);
  dumpViewports      (pDb, indent);
  dumpViews             (pDb, indent);
  dumpMLineStyles (pDb, indent);
  dumpUCSTable        (pDb, indent);
  dumpObject(pDb->getNamedObjectsDictionaryId(), OD_T("Named Objects Dictionary"), indent);
  dumpBlocks(pDb, indent);
}
```

这个例子中，智能指针的使用消除了OdDbObjectId::safeOpenObject()调用返回的指针显式关闭和删除的麻烦。其他表中的入口可以通过相同的方式得到。

访问块入口

块(OdDbBlockTableRecord)中包含实体对象，所以一个块必须在访问之前就打开，下面的代码遍历整个绘图块，并对每个实体调用dumpEntity()函数。

```
void DbDumper::dumpBlocks(OdDbDatabase* pDb, int indent)
{
  /************************************************************************/
  /* Get a SmartPointer to the BlockTable                                 */
  /************************************************************************/
  OdDbBlockTablePtr pTable = pDb->getBlockTableId().safeOpenObject();

  /************************************************************************/
  /* Dump the Description                                                 */
  /************************************************************************/
  writeLine();
  writeLine(indent++, toString(pTable->desc()));

  /************************************************************************/
  /* Get a SmartPointer to a new SymbolTableIterator                      */
  /************************************************************************/
  OdDbSymbolTableIteratorPtr pBlkIter = pTable->newIterator();

  /************************************************************************/
  /* Step through the BlockTable                                          */
  /************************************************************************/
  for (pBlkIter->start(); ! pBlkIter->done(); pBlkIter->step())
```

```
  {
    /******************************************************************/
    /* Open the BlockTableRecord for Reading                          */
    /******************************************************************/
    OdDbBlockTableRecordPtr pBlock = pBlkIter->getRecordId().safeOpenObject();

    /******************************************************************/
    /* Dump the BlockTableRecord                                      */
    /******************************************************************/
    writeLine();
    writeLine(indent, toString(pBlock->desc()));
    writeLine(indent+1, OD_T("Name"), toString(pBlock->getName()));
    writeLine(indent+1, OD_T("Anonymous"), toString(pBlock->isAnonymous()));
    writeLine(indent+1, OD_T("Comments"), toString(pBlock->comments()));
    writeLine(indent+1, OD_T("Origin"), toString(pBlock->origin()));
    writeLine(indent+1, OD_T("Block Insert Units"), toString(pBlock->blockInsertUnits()));
    writeLine(indent+1, OD_T("Block Scaling"), toString(pBlock->blockScaling()));
    writeLine(indent+1, OD_T("Explodable"), toString(pBlock->explodable()));

    OdGeExtents3d extents;
    if (eOk == pBlock->getGeomExtents(extents)) {
      writeLine(indent+1, OD_T("Min Extents"), toString(extents.minPoint()));
      writeLine(indent+1, OD_T("Max Extents"), toString(extents.maxPoint()));
    }
```

```
    writeLine(indent+1, OD_T("Layout"), toString(pBlock->isLayout()));

    writeLine(indent+1, OD_T("Has Attribute Definitions"), toString(pBlock->hasAttributeDefinitions()));

    writeLine(indent+1, OD_T("Xref Status"), toString(pBlock->xrefStatus()));

    if (pBlock->xrefStatus()!=OdDb::kXrfNotAnXref)

    {

      writeLine(indent+1, OD_T("Xref Path"), toString(pBlock->pathName()));

      writeLine(indent+1, OD_T("From Xref Attach"), toString(pBlock->isFromExternalReference()));

      writeLine(indent+1, OD_T("From Xref Overlay"), toString(pBlock->isFromOverlayReference()));

      writeLine(indent+1, OD_T("Xref Unloaded"), toString(pBlock->isUnloaded()));

    }

    /*****************************************************************/

    /* Get a SmartPointer to a new ObjectIterator                    */

    /*****************************************************************/

    OdDbObjectIteratorPtr pEntIter = pBlock->newIterator();


    /*****************************************************************/

    /* Step through the BlockTableRecord                             */

    /*****************************************************************/

    for (; !pEntIter->done(); pEntIter->step())

    {

     /*****************************************************************/

     /* Dump the Entity                                               */

     /*****************************************************************/

     dumpEntity(pEntIter->objectId(), indent+1);
```

```
        }
    }
}

/*****************************************************************/
/* Dump Xdata                                                    */
/*****************************************************************/
void DbDumper::dumpXdata(OdResBuf* resbuf, int indent)
{
    if (!resbuf)
        return;

    writeLine(indent++, OD_T("Xdata:"));

    /*****************************************************************/
    /* Step through the ResBuf chain                                 */
    /*****************************************************************/
    for (; resbuf != 0; resbuf = resbuf->next())
    {
        int code = resbuf->restype();

        /*****************************************************************/
        /* Convert resbuf->ResVal to a string                            */
        /*****************************************************************/
        OdString rightString(OD_T("???"));
        switch (OdDxfCode::_getType(code))
        {
```

```
case OdDxfCode::Name:
case OdDxfCode::String:
case OdDxfCode::LayerName:
    rightString = toString(resbuf->getString());
    break;

case OdDxfCode::Bool:
    rightString = toString(resbuf->getBool());
    break;

case OdDxfCode::Integer8:
    rightString = toString(resbuf->getInt8());
    break;

case OdDxfCode::Integer16:
    rightString = toString(resbuf->getInt16());
    break;

case OdDxfCode::Integer32:
    rightString = toString((int)resbuf->getInt32());
    break;

case OdDxfCode::Double:
    rightString = toString(resbuf->getDouble());
    break;

case OdDxfCode::Angle:
```

```
        rightString = toDegreeString(resbuf->getDouble());
        break;

    case OdDxfCode::Point:
        {
            rightString = toString(resbuf->getPoint3d());
        }
        break;

    case OdDxfCode::BinaryChunk:
        rightString = OD_T("<Binary Data>");
        break;

    case OdDxfCode::ObjectId:
    case OdDxfCode::SoftPointerId:
    case OdDxfCode::HardPointerId:
    case OdDxfCode::SoftOwnershipId:
    case OdDxfCode::HardOwnershipId:
    case OdDxfCode::Handle:
        rightString = toString(resbuf->getHandle());
        break;

    case OdDxfCode::Unknown:
    default:
        rightString = OD_T("Unknown");
        break;
}
```

```
      writeLine(indent,toString(code),rightString);
    }
}

/*****************************************************************/
/* Dump the Xref the full path to the Osnap entity               */
/*****************************************************************/
void DbDumper::dumpXrefFullSubentPath(OdDbXrefFullSubentPath& subEntPath, int indent)
{
   writeLine(indent, OD_T("Subentity Index"), toString((int)subEntPath.subentId().index()));
   writeLine(indent, OD_T("Subentity Type"), toString(subEntPath.subentId().type()));
   for (OdUInt32 j = 0; j < subEntPath.objectIds().size(); j++)
   {
      OdDbEntityPtr pEnt = subEntPath.objectIds()[j].openObject();
      if (!pEnt.isNull())
      {
         writeLine(indent, toString(pEnt->isA()), toString(pEnt->getDbHandle()));
      }
   }
}

/*****************************************************************/
/* Dump Object Snap Point Reference for an Associative Dimension */
/*****************************************************************/
void DbDumper::dumpOsnapPointRef(OdDbOsnapPointRefPtr pRef,    int index, int indent)
{
   writeLine(indent++, toString(pRef->isA()), toString(index));
```

```
    writeLine(indent, OD_T("Osnap Mode"), toString(pRef->osnapType()));
    writeLine(indent, OD_T("Near Osnap"),    toString(pRef->nearPointParam()));
    writeLine(indent, OD_T("Osnap Point"), toString(pRef->point()));

    writeLine(indent, OD_T("Main Entity"));
    dumpXrefFullSubentPath(pRef->mainEntity(), indent+1);

    writeLine(indent, OD_T("Intersect Entity"));
    dumpXrefFullSubentPath(pRef->intersectEntity(), indent + 1);

    if (pRef->lastPointRef())
    {
      writeLine(indent, OD_T("Last Point Referenced"));
      dumpOsnapPointRef(pRef->lastPointRef(), index, indent + 1);
    }
    else
    {
      writeLine(indent, OD_T("Last Point Referenced"), OD_T("Null"));
    }
}

/*****************************************************************/
/* Dump an Associative dimension                                 */
/*****************************************************************/
void DbDumper::dumpDimAssoc(OdDbObjectPtr pObject, const int indent)
{
    OdDbDimAssocPtr pDimAssoc = pObject;
```

```
    writeLine(indent, OD_T("Associative"),
          toString ((OdDbDimAssoc::AssocFlags) pDimAssoc->assocFlag()));
    writeLine(indent, OD_T("TransSpatial"), toString(pDimAssoc->isTransSpatial()));
    writeLine(indent, OD_T("Rotated Type"), toString(pDimAssoc->rotatedDimType()));

    for (int i = 0; i < OdDbDimAssoc::kMaxPointRefs; i++)
    {
      OdDbOsnapPointRefPtr pRef = pDimAssoc->pointRef((OdDbDimAssoc::AssocFlags) (1 << i));
      if (!pRef.isNull())
      {
        dumpOsnapPointRef(pRef, i, indent);
      }
      else
      {
        break;
      }
    }
}

/******************************************************************************/
/* Dump the object.                                                           */
/*                                                                            */
/* Dictionary objects are recursively dumped.                                 */
/* XRecord objects are dumped.                                                */
/* DimAssoc objects are dumped.                                               */
/******************************************************************************/
void DbDumper::dumpObject(
```

```
                                    OdDbObjectId id,
                                    OdString itemName,
                                    int indent)
{
  /******************************************************************/
  /* Get a SmartPointer to the object                               */
  /******************************************************************/
  OdDbObjectPtr pObject = id.safeOpenObject();

  /******************************************************************/
  /* Dump the item name and class name                              */
  /******************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
  {
    writeLine();
  }
  writeLine(indent++, itemName, toString(pObject->isA()));

  /******************************************************************/
  /* Dispatch                                                       */
  /******************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
  {
    /****************************************************************/
    /* Dump the dictionary                                          */
    /****************************************************************/
    OdDbDictionaryPtr pDic = pObject;
```

```
    /*****************************************************************/
    /* Get a SmartPointer to a new DictionaryIterator                */
    /*****************************************************************/
    OdDbDictionaryIteratorPtr pIter = pDic->newIterator();

    /*****************************************************************/
    /* Step through the Dictionary                                   */
    /*****************************************************************/
    for (; !pIter->done(); pIter->next())
    {
      /*****************************************************************/
      /* Dump the Dictionary object                                    */
      /*****************************************************************/
      // Dump each item in the dictionary.
      dumpObject(pIter->objectId(), pIter->name(), indent);
    }
  }
  else if (pObject->isKindOf(OdDbXrecord::desc()))
  {
    /*****************************************************************/
    /* Dump an Xrecord                                               */
    /*****************************************************************/
    OdDbXrecordPtr pXRec = pObject;
    dumpXdata(pXRec->rbChain(), indent);
  }
  else if (pObject->isKindOf(OdDbDimAssoc::desc()))
```

```
    {
      /*****************************************************************/
      /* Dump an Associative dimension                                 */
      /*****************************************************************/
      dumpDimAssoc(pObject, indent);
    }
}

/*****************************************************************/
/* Dump the database                                             */
/*****************************************************************/
void DbDumper::dump(OdDbDatabase* pDb, int indent)
{
  dumpHeader          (pDb, indent);
  dumpLayers          (pDb, indent);
  dumpLinetypes       (pDb, indent);
  dumpTextStyles      (pDb, indent);
  dumpDimStyles       (pDb, indent);
  dumpRegApps         (pDb, indent);
  dumpViewports       (pDb, indent);
  dumpViews           (pDb, indent);
  dumpMLineStyles (pDb, indent);
  dumpUCSTable        (pDb, indent);
  dumpObject(pDb->getNamedObjectsDictionaryId(), OD_T("Named Objects Dictionary"), indent);
  dumpBlocks(pDb, indent);
}
```

访问命名对象字典中的对象

主要字典对象是访问数据库中对象的起点，下面的例子打印出主要字典对象的层次结构，并且递归找到的每个子字典对象(并非遍历整个对

象层次结构)。

```
void DbDumper::dumpObject(
                                    OdDbObjectId id,
                                    OdString itemName,
                                    int indent)
{
  /*****************************************************************/
  /* Get a SmartPointer to the object                              */
  /*****************************************************************/
  OdDbObjectPtr pObject = id.safeOpenObject();

  /*****************************************************************/
  /* Dump the item name and class name                             */
  /*****************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
  {
    writeLine();
  }
  writeLine(indent++, itemName, toString(pObject->isA()));

  /*****************************************************************/
  /* Dispatch                                                      */
  /*****************************************************************/
  if (pObject->isKindOf(OdDbDictionary::desc()))
```

```
{
  /************************************************************************/
  /* Dump the dictionary                                                  */
  /************************************************************************/
  OdDbDictionaryPtr pDic = pObject;

  /************************************************************************/
  /* Get a SmartPointer to a new DictionaryIterator                       */
  /************************************************************************/
  OdDbDictionaryIteratorPtr pIter = pDic->newIterator();

  /************************************************************************/
  /* Step through the Dictionary                                          */
  /************************************************************************/
  for (; !pIter->done(); pIter->next())
  {
    /************************************************************************/
    /* Dump the Dictionary object                                           */
    /************************************************************************/
    // Dump each item in the dictionary.
    dumpObject(pIter->objectId(), pIter->name(), indent);
  }
}
else if (pObject->isKindOf(OdDbXrecord::desc()))
{
  /************************************************************************/
  /* Dump an Xrecord                                                      */
```

```
      /*****************************************************************/
      OdDbXrecordPtr pXRec = pObject;
      dumpXdata(pXRec->rbChain(), indent);
   }
   else if (pObject->isKindOf(OdDbDimAssoc::desc()))
   {
      /*****************************************************************/
      /* Dump an Associative dimension                                 */
      /*****************************************************************/
      dumpDimAssoc(pObject, indent);
   }
}

/*****************************************************************************/
/* Dump the database                                                         */
/*****************************************************************************/
void DbDumper::dump(OdDbDatabase* pDb, int indent)
{
   dumpHeader          (pDb, indent);
   dumpLayers          (pDb, indent);
   dumpLinetypes       (pDb, indent);
   dumpTextStyles      (pDb, indent);
   dumpDimStyles       (pDb, indent);
   dumpRegApps         (pDb, indent);
   dumpViewports       (pDb, indent);
   dumpViews           (pDb, indent);
   dumpMLineStyles (pDb, indent);
```

```
    dumpUCSTable        (pDb, indent);
    dumpObject(pDb->getNamedObjectsDictionaryId(), OD_T("Named Objects Dictionary"), indent);
    dumpBlocks(pDb, indent);
}
```

以上对象转储可以由主要字典启动：

```
dumpObject(pDb->getNamedObjectsDictionaryId(),
                "Named Objects Dictionary",
                pDb,
                os,
                "    ");
```

使用OdDbDictionary::getAt()函数，字典中的对象可以根据他的名字得到。一下代码段返回字典名为“ACAD_LAYOUT”的入口（如果存

在的话）。

```
OdDbDictionaryPtr pDic =
    pDb->getNamedObjectsDictionaryId().openObject(OdDb::kForRead);
if (pDic)
{
    OdDbDictionaryPtr pLayoutDic =
        pDic->getAt("ACAD_LAYOUT", OdDb::kForRead);
    if (pLayoutDic)
    {
        // Do something with the layout dictionary
    }
}
```

注意使用OdDbDatabase::getLayoutDictionaryId()可以直接访问数据库中图层字典ID。

创建矢量化程序

矢量化程序示例

下面介绍一个简单的矢量化例子，代码在Example/OdVectorizEx目录中。这个程序创建必须的客户端矢量化类，并用他们转储2D矢量化输出（或者是3D矢量化输出）到一个text文件中。

创建一个自定义矢量化设备对象

下面ExGsSimpleDevice类是最小设备类的一个例子，OdGiConveyorGeometry重载（polylineOut，circleProc等）接受并处理矢量化框架生成的几何图形。

```
class ExGsSimpleDevice :
  public OdGsBaseVectorizeDevice
{
  OdGiConveyorGeometryDumperPtr m_pDestGeometry;
  OdGiDumperPtr                 m_pDumper;

public:
  enum DeviceType
  {
    k2dDevice,
    k3dDevice
  };

  ExGsSimpleDevice();

  /******************************************************************/
  /* Set the target data stream and the type.                       */
```

```
/******************************************************************/
static OdGsDevicePtr createObject(DeviceType type);

/******************************************************************/
/* Called by the vectorization framework to update                */
/* the GUI window for this Device object                          */
/*                                                                */
/* pUpdatedRect specifies a rectangle to receive the region updated */
/* by this function.                                              */
/*                                                                */
/* The override should call the parent version of this function,  */
/* OdGsBaseVectorizeDevice::update().                             */
/******************************************************************/
void update(OdGsDCRect* pUpdatedRect);

/******************************************************************/
/* Creates a new OdGsView object, and associates it with this Device */
/* object.                                                        */
/*                                                                */
/* pInfo is a pointer to the Client View Information for this Device */
/* object.                                                        */
/*                                                                */
/* bEnableLayerVisibilityPerView specifies that layer visibility  */
/* per viewport is to be supported.                               */
/******************************************************************/
OdGsViewPtr createView(const OdGsClientViewInfo* pInfo = 0,
                       bool bEnableLayerVisibilityPerView = false);
```

```
/************************************************************************/
/* Returns the geometry receiver associated with this object.           */
/************************************************************************/
OdGiConveyorGeometry* destGeometry();

/************************************************************************/
/* Returns the geometry dumper associated with this object.             */
/************************************************************************/
OdGiDumper* dumper();

/************************************************************************/
/* Called by each associated view object to set the current RGB draw    */
/* color.                                                               */
/************************************************************************/
void draw_color(ODCOLORREF color);

/************************************************************************/
/* Called by each associated view object to set the current ACI draw    */
/* color.                                                               */
/************************************************************************/
void draw_color_index(OdUInt16 colorIndex);

/************************************************************************/
/* Called by each associated view object to set the current draw        */
/* lineweight and pixel width                                           */
```

```
  /************************************************************************/
  void draw_lineWidth(OdDb::LineWeight weight, int pixelLineWidth);

  /************************************************************************/
  /* Called by each associated view object to set the current Fill Mode   */
  /************************************************************************/
  void draw_fill_mode(OdGiFillType fillMode);


  void setupSimplifier(const OdGiDeviation* pDeviation);

private:
  DeviceType                    m_type;
}; // end ExGsSimpleDevice
```

客户程序必须在他们的设备类中重载OdGsDevice::createView函数，如下面的例子：

```
OdGsViewPtr ExGsSimpleDevice::createView(const OdGsClientViewInfo* pInfo,
                                         bool bEnableLayerVisibilityPerView)
{
  /************************************************************************/
  /* Create a View                                                        */
  /************************************************************************/
  OdGsViewPtr pView = ExSimpleView::createObject();
  ExSimpleView* pMyView = static_cast<ExSimpleView*>(pView.get());

  /************************************************************************/
  /* This call is required by DD 1.13+                                    */
```

```
  /******************************************************************/
  pMyView->init(this, pInfo, bEnableLayerVisibilityPerView);

  /******************************************************************/
  /* Directs the output geometry for the view to the                */
  /* destination geometry object                                    */
  /******************************************************************/
  pMyView->output().setDestGeometry(*m_pDestGeometry);

  return (OdGsView*)pMyView;
}

/******************************************************************/
/* Returns the geometry receiver associated with this object.     */
/******************************************************************/
OdGiConveyorGeometry* ExGsSimpleDevice::destGeometry()
{
  return m_pDestGeometry;
}

/******************************************************************/
/* Returns the geometry dumper associated with this object.       */
/******************************************************************/
OdGiDumper* ExGsSimpleDevice::dumper()
{
  return m_pDumper;
}
```

```
void ExSimpleView::ownerDrawDc(const OdGePoint3d& origin,
                               const OdGeVector3d& u,
                               const OdGeVector3d& v,
                               const OdGiSelfGdiDrawable* /*pDrawable*/,
                               bool dcAligned,
                               bool allowClipping)
{
  // ownerDrawDc is not conveyor primitive. It means we must take all rendering processings
  // (transforms) by ourselves
  OdGeMatrix3d eyeToOutput = eyeToOutputTransform();
  OdGePoint3d originXformed(eyeToOutput * origin);
  OdGeVector3d uXformed(eyeToOutput * u), vXformed(eyeToOutput * v);

  OdGiDumper* pDumper = device()->dumper();

  pDumper->output(OD_T("ownerDrawDc"));

  // It's shown only in 2d mode.
  if(mode() == OdGsView::k2DOptimized)
  {
    pDumper->pushIndent();
    pDumper->output(OD_T("origin xformed"), toString(originXformed));
    pDumper->output(OD_T("u xformed"),      toString(uXformed));
    pDumper->output(OD_T("v xformed"),      toString(vXformed));
    pDumper->output(OD_T("dcAligned"),      toString(dcAligned));
    pDumper->output(OD_T("allowClipping"),  toString(allowClipping));
```

```
      pDumper->popIndent();
    }
}

/************************************************************************/
/* Called by the vectorization framework to update                      */
/* the GUI window for this Device object                                */
/*                                                                      */
/* pUpdatedRect specifies a rectangle to receive the region updated     */
/* by this function.                                                    */
/*                                                                      */
/* The override should call the parent version of this function,        */
/* OdGsBaseVectorizeDevice::update().                                   */
/************************************************************************/
void ExGsSimpleDevice::update(OdGsDCRect* pUpdatedRect)
{
  OdGsBaseVectorizeDevice::update(pUpdatedRect);
}

/************************************************************************/
/* Called by each associated view object to set the current RGB draw    */
/* color.                                                               */
/************************************************************************/
void ExGsSimpleDevice::draw_color(ODCOLORREF color)
{
  dumper()->output(OD_T("draw_color"), toRGBString(color));
}
```

```
/******************************************************************************/
/* Called by each associated view object to set the current ACI draw        */
/* color.                                                                     */
/******************************************************************************/
void ExGsSimpleDevice::draw_color_index(OdUInt16 colorIndex)
{
  dumper()->output(OD_T("draw_color_index"), toString(colorIndex));
}

/******************************************************************************/
/* Called by each associated view object to set the current draw              */
/* lineweight and pixel width                                                 */
/******************************************************************************/
void ExGsSimpleDevice::draw_lineWidth(OdDb::LineWeight weight, int pixelLineWidth)
{
  dumper()->output(OD_T("draw_lineWidth"));
  dumper()->pushIndent();
  dumper()->output(OD_T("weight"), toString(weight));
  dumper()->output(OD_T("pixelLineWidth"), toString(pixelLineWidth));
  dumper()->popIndent();
}


/******************************************************************************/
/* Called by each associated view object to set the current Fill Mode      */
/******************************************************************************/
```

```
void ExGsSimpleDevice::draw_fill_mode(OdGiFillType fillMode)
{
   OdString strMode;
   switch (fillMode)
   {
   case kOdGiFillAlways:
      strMode = OD_T("FillAlways");
      break;
   case kOdGiFillNever:
      strMode = OD_T("FillNever");
   }
   dumper()->output(OD_T("draw_fill_mode"), strMode);
}

/**********************************************************************/
/* Called by the vectorization framework to give the                  */
/* client application a chance to terminate the current               */
/* vectorization process.    Returning true from this function will   */
/* stop the current vectorization operation.                          */
/**********************************************************************/

bool ExSimpleView::regenAbort() const
{
   // return true here to abort the vectorization process
   return false;
}
```

```
/************************************************************************/
/* Called by the vectorization framework to vectorize                   */
/* each entity in this view.    This override allows a client           */
/* application to examine each entity before it is vectorized.          */
/* The override should call the parent version of this function,        */
/* OdGsBaseVectorizeView::draw().                                       */
/************************************************************************/
void ExSimpleView::draw(const OdGiDrawable* pDrawable)
{
  OdDbObjectPtr pEnt = OdDbObject::cast(pDrawable);

  device()->dumper()->output(OD_T(""));
  if (pEnt.get())
  {
    device()->dumper()->output(OD_T("Start Drawing ") + toString(pEnt->objectId()),
toString(pEnt->objectId().getHandle()));
  }
  else
  {
    device()->dumper()->output(OD_T("Start Drawing") + toString(pEnt->objectId()),
toString(OD_T("non-DbResident")));
  }

  device()->dumper()->pushIndent();

  /************************************************************************/
  /* The parent class version of this function must be called.            */
  /************************************************************************/
```

```
  OdGsBaseVectorizeView::draw(pDrawable);


  device()->dumper()->popIndent();
  if (pEnt.get())
  {
    device()->dumper()->output(OD_T("End Drawing ") + toString(pEnt->objectId()),
toString(pEnt->objectId().getHandle()));
  }
  else
  {
    device()->dumper()->output(OD_T("End Drawing ") + toString(pEnt->objectId()),
toString(OD_T("non-DbResident")));
  }
}


/************************************************************************/
/* Flushes any queued graphics to the display device.                   */
/************************************************************************/
void ExSimpleView::update()
{
  /************************************************************************/
  /* If geometry receiver is a simplifier, we must re-set the draw        */
  /* context for it                                                       */
  /************************************************************************/

(static_cast<OdGiGeometrySimplifier*>(device()->destGeometry()))->setDrawContext(drawContext()
);


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

```
/* Comment these functions to get primitives in eye coordinates.           */
/*************************************************************************/
OdGeMatrix3d eye2Screen(eyeToScreenMatrix());
OdGeMatrix3d screen2Eye(eye2Screen.inverse());
setEyeToOutputTransform(eye2Screen);

// perform viewport clipping in eye coordinates:

/*************************************************************************/
/* Perform viewport clipping in eye coordinates.                         */
/*************************************************************************/
m_eyeClip.m_bClippingFront = isFrontClipped();
m_eyeClip.m_bClippingBack = isBackClipped();
m_eyeClip.m_dFrontClipZ = frontClip();
m_eyeClip.m_dBackClipZ = backClip();
m_eyeClip.m_vNormal = viewDir();
m_eyeClip.m_ptPoint = target();
m_eyeClip.m_Points.clear();

OdGeVector2d halfDiagonal(fieldWidth() / 2.0, fieldHeight() / 2.0);

if(m_nrcCounts.size() == 1) // polygons aren't supported yet
{
   // polygonal clipping
   int i;
   for(i = 0; i < m_nrcCounts[0]; i ++)
   {
```

```
            OdGePoint3d screenPt(m_nrcPoints[i].x, m_nrcPoints[i].y, 0.0);
            screenPt.transformBy(screen2Eye);
            m_eyeClip.m_Points.append(OdGePoint2d(screenPt.x, screenPt.y));
        }
    }
    else
    {
        // rectangular clipping
        m_eyeClip.m_Points.append(OdGePoint2d::kOrigin - halfDiagonal);
        m_eyeClip.m_Points.append(OdGePoint2d::kOrigin + halfDiagonal);
    }

    m_eyeClip.m_xToClipSpace = getWorldToEyeTransform();

    pushClipBoundary(&m_eyeClip);

    OdGsBaseVectorizeView::update();

    popClipBoundary();
} // end ExSimpleView::update()


/************************************************************************/
/* Notification function called by the vectorization framework          */
/* whenever the rendering attributes have changed.                      */
/*                                                                      */
/* Retrieves the current vectorization traits (color                    */
```

```
/* lineweight, etc.) and sets them in this device.                    */
/*************************************************************************/
void ExSimpleView::onTraitsModified()
{
  bool b = isEntityTraitsDataChanged();
  OdGsBaseVectorizeView::onTraitsModified();
  const OdGiSubEntityTraitsData& currTraits = effectiveTraits();
  if(currTraits.trueColor().isByColor())
  {
    device()->draw_color(ODTOCOLORREF(currTraits.trueColor()));
  }
  else
  {
    device()->draw_color_index(currTraits.color());
  }

  OdDb::LineWeight lw = currTraits.lineWeight();
  device()->draw_lineWidth(lw, lineweightToPixels(lw));
  device()->draw_fill_mode(currTraits.fillType());

} // end ExSimpleView::onTraitsModified()


void ExSimpleView::beginViewVectorization()
{
  OdGsBaseVectorizeView::beginViewVectorization();
  device()->setupSimplifier( &m_pModelToEyeProc->eyeDeviation() );
```

```
}

void ExGsSimpleDevice::setupSimplifier(const OdGiDeviation* pDeviation)
{
  m_pDestGeometry->setDeviation(pDeviation);
}

//=========================================================================
```

这个函数应该创建并返回一个客户提供的自定义视图类。这个例子也设置为视图生成几何的输出对象。

创建一个自定义矢量化视图对象

```
class ExSimpleView : public OdGsBaseVectorizeView
{
  OdGiClipBoundary          m_eyeClip;
protected:

  /************************************************************************/
  /* Returns the ExGsSimpleDevice instance that owns this view.           */
  /************************************************************************/
  virtual void beginViewVectorization();

  ExGsSimpleDevice* device()
  {
    return (ExGsSimpleDevice*)OdGsBaseVectorizeView::device();
  }
```

```
  ExSimpleView()
  {
    m_eyeClip.m_bDrawBoundary = false;
  }

  friend class ExGsSimpleDevice;

  /******************************************************************/
  /* PseudoConstructor                                              */
  /******************************************************************/
  static OdGsViewPtr createObject()
  {
    return OdRxObjectImpl<ExSimpleView,OdGsView>::createObject();
  }

  /******************************************************************/
  /* To know when to set traits                                     */
  /******************************************************************/
  void setEntityTraits();

public:
  /******************************************************************/
  /* Called by the vectorization framework to vectorize             */
  /* each entity in this view.    This override allows a client     */
  /* application to examine each entity before it is vectorized.    */
  /* The override should call the parent version of this function,  */
```

```
/* OdGsBaseVectorizeView::draw().                                          */
/*************************************************************************/
void draw(const OdGiDrawable*);

/*************************************************************************/
/* Called by the vectorization framework to give the                       */
/* client application a chance to terminate the current                    */
/* vectorization process.    Returning true from this function will        */
/* stop the current vectorization operation.                               */
/*************************************************************************/
bool regenAbort() const;

/*************************************************************************/
/* Returns the recommended maximum deviation of the current                */
/* vectorization.                                                          */
/*************************************************************************/
double deviation(const OdGiDeviationType, const OdGePoint3d&) const
{
   // to force vectorization framework to use some reasonable value.
   return 0.000001;
}

/*************************************************************************/
/* Flushes any queued graphics to the display device.                      */
/*************************************************************************/
void update();
```

```
  TD_USING(OdGsBaseVectorizeView::update);

  /*********************************************************************/
  /* Notification function called by the vectorization framework       */
  /* whenever the rendering attributes have changed.                   */
  /*                                                                   */
  /* Retrieves the current vectorization traits (color                 */
  /* lineweight, etc.) and sets them in this device.                   */
  /*********************************************************************/
  void onTraitsModified();


  void ownerDrawDc(
    const OdGePoint3d& origin,
    const OdGeVector3d& u,
    const OdGeVector3d& v,
    const OdGiSelfGdiDrawable* pDrawable,
    bool bDcAligned,
    bool bAllowClipping);
}; // end ExSimpleView
```

矢量化之前先检查实体

ExSimpleView::draw()重载允许一个客户矢量化程序在每个视图中的实体矢量化之前访问每一个实体。这和定义在老的Open Design C视图工具库中的examinEntity回调相似。

正常完成前终止矢量化处理

重载ExSimpleView::regenAbort()允许客户矢量化程序在正常终止之前打断矢量化处理。Teigha for .dwg文件系统框架在整个过程中重复调用这个函数。如果客户重载在任何点返回true，那么矢量化过程将被终止。

矢量化过程中返回2D或者3D向量

矢量化框架调用ExSimpleView::update()函数来矢量化与特定视图关联的实体。

```
void ExSimpleView::update()
{
  /*****************************************************************/
  /* If geometry receiver is a simplifier, we must re-set the draw */
  /* context for it                                                */
  /*****************************************************************/

(static_cast<OdGiGeometrySimplifier*>(device()->destGeometry()))->setDrawContext(drawContext()
);

  /*****************************************************************/
  /* Comment these functions to get primitives in eye coordinates. */
  /*****************************************************************/
  OdGeMatrix3d eye2Screen(eyeToScreenMatrix());
  OdGeMatrix3d screen2Eye(eye2Screen.inverse());
  setEyeToOutputTransform(eye2Screen);

  // perform viewport clipping in eye coordinates:

  /*****************************************************************/
  /* Perform viewport clipping in eye coordinates.                 */
  /*****************************************************************/
  m_eyeClip.m_bClippingFront = isFrontClipped();
  m_eyeClip.m_bClippingBack = isBackClipped();
  m_eyeClip.m_dFrontClipZ = frontClip();
  m_eyeClip.m_dBackClipZ = backClip();
```

```
m_eyeClip.m_vNormal = viewDir();
m_eyeClip.m_ptPoint = target();
m_eyeClip.m_Points.clear();

OdGeVector2d halfDiagonal(fieldWidth() / 2.0, fieldHeight() / 2.0);

if(m_nrcCounts.size() == 1) // polygons aren't supported yet
{
   // polygonal clipping
   int i;
   for(i = 0; i < m_nrcCounts[0]; i ++)
   {
      OdGePoint3d screenPt(m_nrcPoints[i].x, m_nrcPoints[i].y, 0.0);
      screenPt.transformBy(screen2Eye);
      m_eyeClip.m_Points.append(OdGePoint2d(screenPt.x, screenPt.y));
   }
}
else
{
   // rectangular clipping
   m_eyeClip.m_Points.append(OdGePoint2d::kOrigin - halfDiagonal);
   m_eyeClip.m_Points.append(OdGePoint2d::kOrigin + halfDiagonal);
}

m_eyeClip.m_xToClipSpace = getWorldToEyeTransform();

pushClipBoundary(&m_eyeClip);
```

```
    OdGsBaseVectorizeView::update();


    popClipBoundary();
} // end ExSimpleView::update()
```