Contents

[第一章、建立一个最简单的MDL应用程序 3](#_Toc410238134)

[第二章、在MDL应用中创建元素 5](#_Toc410238135)

[第三章、在Visual Studio环境中生成并调试MDL应用 11](#_Toc410238136)

[第四章、给MDL应用添加命令 17](#_Toc410238137)

[第五章、用MstnTool类实现交互式工具 21](#_Toc410238138)

[第六章、给MDL应用添加用户界面 31](#_Toc410238139)

[第七章、响应MicroStation中的事件 36](#_Toc410238140)

**《一步步学习MDL》**

本系列博客将由浅入深地帮助您学习MDL。按照本博客的步骤，相信您能很快地掌握这一开发MicroStation应用程序的最主要方法。

**第零章、介绍与必备条件**

从MicroStation 4.0版开始就有MDL这一功能强大的开发手段了，那还要追溯到上世纪90年代初。那时的MDL是MicroStation Development Language（MicroStation开发语言）的简称，它是Bentley公司基于C语言的一种扩展语言，编译器和链接器均由Bentley公司提供。最终生成的程序扩展名为.MA，是MicroStation Application的缩写。我们把这样的开发方法叫做PureMDL，开发PureMDL您仅需要一个源代码编辑器，其它的工具（编译器、链接器以及调试器）都由Bentley提供了。

随着时间的推移，MDL也在与时俱进。今天的MDL已经完全可以基于C++来开发，因而我们可以采用微软的Visual Studio作为开发工具来生成本机代码的DLL，同时，为了保留MDL的一些特性（如命令表、MDL特有的资源等），仍然需要生成一个.MA文件。我们称这种新的MDL为NativeCode MDL，这里的MDL不再是MicroStation Development Language而是MicroStation Development Library（MicroStation开发库）。NativeCode MDL能使您采用面向对象的编程方式来写代码，还能使您直接调用任何VC++中可以调用的功能。大多数新的MicroStation开发功能（如XAttribute、点云、i-model等）都采用类的形式提供，这些新的功能也要求我们必须使用NativeCode MDL来开发应用。PureMDL、NativeCode再加上新的开发手段Addins三者的比照如下图所示。

**PureMDL Application (C)**

**NativeCode MDL Application (C/C++)**

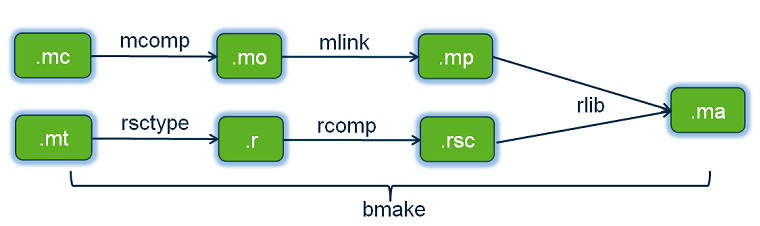
**Addins Application (C++/CLI, C#, VB.NET)**

**MicroStation (as a virtual machine)**

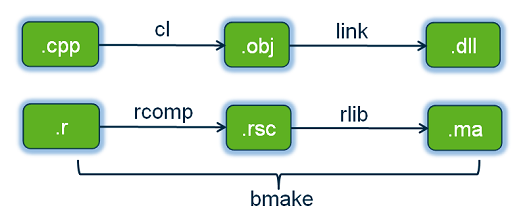
**Operating System (Windows 7, Windows Vista or Windows XP)**

**.NET Framework**

为了进一步分清楚PureMDL和NativeCode MDL，下图分别列出了两者的生成过程。其中的mcomp.exe、mlink.exe、rsctype.exe、rcomp.exe、rlib.exe都是由Bentley提供的执行程序，当您安装完MicroStation SDK后在…\MicroStation\mdl\bin目录下就能找到它们。而cl.exe和link.exe则是来自VS的编译器和连接器程序，在您的VS的…\VC\bin目录下能找到。从MicroStation SDK V8iSS2开始，所附带的例子都改为NativeCode MDL了，而以前版本的SDK中所附带的例子还是PureMDL的。



生成PureMDL应用程序的过程



生成NativeCode MDL应用程序的过程

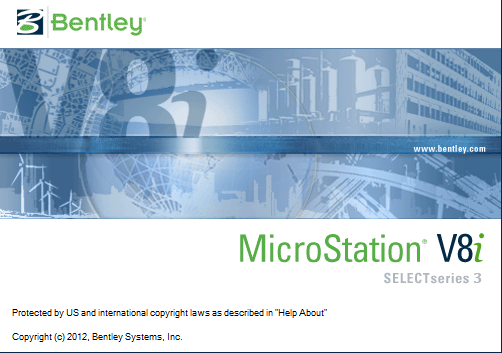
本系列博客中的代码都将以NativeCode MDL来写，所以请按下面步骤安装我们的开发环境：

1、安装Visual Studio 2005+SP1：

虽然大多数MicroStation的功能在VS2008和VS2010中也可使用，但有些特定的功能则必须在VS2005下才能正常工作，所以，强烈建议您安装VS2005及其补丁包1；

2、安装MicroStation V8iSS3：从SS3开始，可以联网自动安装预安装包了，如果您的网络条件较差，还是建议您先下载预安装包后在行安装；

3、安装MicroStation SDK V8iSS3。

如果您还没有VS2005，请与微软公司联系。如果您还没有MicroStation软件及其开发包，您需要加入Bentley开发商网络（BDN）以获取它们。详细情况请与我（yongan.fu@bentley.com）联系。

如果您的操作系统是64位的Windows 7，则MicroStation及其开发包的默认安装位置如下：

MicroStation程序文件：C:\Program Files (x86)\Bentley\MicroStation V8i (SELECTseries)

工作空间：C:\ProgramData\Bentley\MicroStation V8i (SELECTseries)\WorkSpace

SDK程序文件：C:\Program Files (x86)\Bentley\MicroStation V8i (SELECTseries)\mdl

SDK例子：“我的文档”下的MDLProjects V8i (SELECTseries 3)

安装完SDK后在C:\Program Files (x86)\Bentley\MicroStation V8i (SELECTseries)\Documentation目录下会有三个重要的开发帮助文档：

① MDLProgrammerGuide.chm：该文档是程序员指南，详细介绍了什么是MDL、如果建立并运行一个应用程序、MicroStation中的资源、MFC对话框、i-model开发以及如何使用向导来开发应用程序；

②MDLAPIFunctionReference.chm：该文档是MDL的C API参考，里面列出了上千个基于C的MDL函数及其详细说明；

③MicroStationAPI.chm：该文档是新的MDL C++ API参考，里面列出了数百个在NativeCode MDL中可用的类。

在C:\Program Files (x86)\Bentley\MicroStation V8i (SELECTseries)\MicroStation目录下有两个重要开发文档您可能会用到：

④readme\_microstationsdk.chm：MicroStation各版本中的MDL的变化；

⑤MicroStationVBA.chm：该文档在安装MicroStation后就存在了，不属于SDK安装的内容。其本意是为MicroStationVBA编程服务的，不过，如果您采用COM方式编程的话，有关MicroStationDGN对象模型的内容可从该文档中查到。这一部分内容可参考我的另一系列博客Learning MicroStation Addins Step By Step。

在NativeCode MDL程序中，您既可以调用C API也可以调用C++ API。目前这两种API并行使用，新功能基本上都是在C++ API中才有，而有些旧有的功能只有在C API中才有。还有一些功能两个里面都有。我们在此系列博客中将尽量使用C++ API来写代码。

# 第一章、建立一个最简单的MDL应用程序

本章将带领您从零开始一步步建立一个最简单的Hello World本机代码MDL应用程序，该程序能够在Mstn中装载运行。

1、在任一驱动器上建立文件夹\MDLSource\HelloWorld，我是在D:上建立的；

2、启动一个文本编辑器（当然可以启动VS2005用作编辑器），在其中键入如下内容并保存为文件D:\MDLSource\HelloWorld\HelloWorld.cpp。该文件中含有程序的入口点函数MdlMain，当应用程序被装载后会从该入口点开始执行。我们在该函数中调用了MDL C API函数mdlDialog\_dmsgsPrint用以弹出一个消息框显示Hello World字样。

/\*-------------------------------------------------------------+

| HelloWorld.cpp |

+-------------------------------------------------------------\*/

#include <MicroStationAPI.h>

#include <msdialog.fdf>

extern "C" DLLEXPORT int MdlMain (int argc, char \*argv[])

{

mdlDialog\_dmsgsPrint ("Hello World");

return 0;

}

3、在文本编辑器中键入如下内容并保存为文件D:\MDLSource\HelloWorld\HelloWorld.r。该文件中定义了DllMdlApp资源，该资源的作用是将MA（实际上不是MA文件名而是任务标识符）与指定的DLL文件关联起来。

#include <rscdefs.h>

#define DLLAPP\_PRIMARY 1

DllMdlApp DLLAPP\_PRIMARY =

{

"HELLOWORLD", "HelloWorld" // taskid, dllName

}

4、将如下内容复制并粘贴成一个叫做HelloWorld.mke的文件保存到D:\MDLSource\HelloWorld下。该文件是生成项目的控制文件。在项目生成过程中，bmake读取该文件的内容然后调用相应的编译器和链接器从源文件生成最终的MA和DLL。上一章的图片显示了这一项目生成过程。

#--------------------------------------------------------------------------------------

# $Source: HelloWorld.mke

#--------------------------------------------------------------------------------------

appName = HelloWorld

appObjects = $(o)$(appName)$(oext)

appRscs = $(o)$(appName).rsc

baseDir = $(\_MakeFilePath)

%include mdl.mki

dirToSearch = $(MSMDE)/mdl/MicroStationAPI

%include cincapnd.mki

#----------------------------------------------------------------------

# Create needed output directories if they don't exist

#----------------------------------------------------------------------

always:

~mkdir $(o)

~mkdir $(rscObjects)

~mkdir $(reqdObjs)

#----------------------------------------------------------------------

# Define macros for files included in our link and resource merge

#----------------------------------------------------------------------

DLM\_NO\_SIGN = 1

DLM\_OBJECT\_DEST = $(o)

DLM\_NAME = $(appName)

DLM\_OBJECT\_FILES = $(appObjects)

DLM\_NO\_DLS = 1

DLM\_NO\_DEF = 1

DLM\_NOENTRY = 1

DLM\_DEST = $(mdlapps)

DLM\_NO\_DELAYLOAD = 1

DLM\_NO\_NTBSADDR = 1

DLM\_LIBRARY\_FILES = $(mdlLibs)BentleyDgn.lib \

$(mdlLibs)toolsubs.lib \

$(mdlLibs)ditemlib.lib \

$(mdlLibs)mdllib.lib

#--------------------------------------------

# Create command table and header file

#--------------------------------------------

#$(baseDir)$(appName)cmd.h : $(baseDir)$(appName)cmd.r

#$(o)$(appName)cmd.rsc : $(baseDir)$(appName)cmd.r

#-----------------------------------------------------------------------

# Generate resource files

#-----------------------------------------------------------------------

$(o)$(appName).rsc : $(baseDir)$(appName).r

#----------------------------------------------------------------------

# Generate MA

#----------------------------------------------------------------------

$(mdlapps)$(appName).ma : $(appRscs)

$(msg)

> $(o)make.opt

-o$@

$(appRscs)

<

$(RLibCmd) @$(o)make.opt

~time

#-----------------------------------------------------------------------------------------

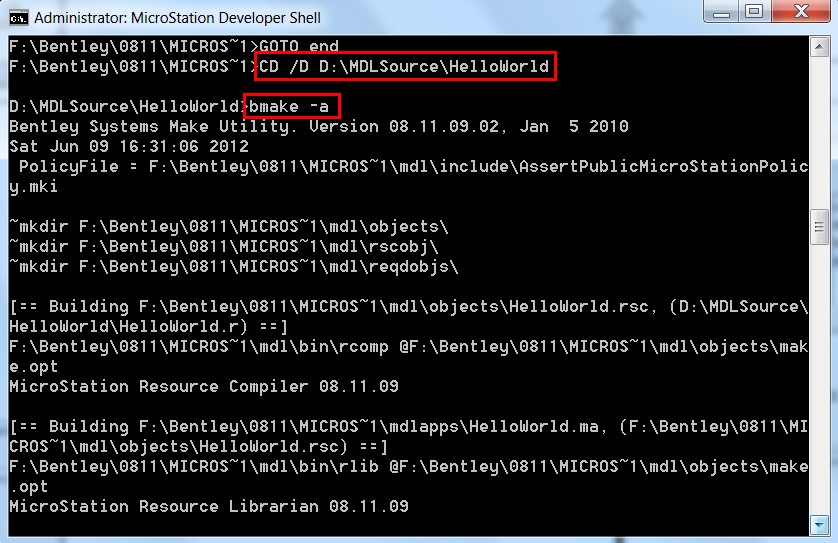
# Builds any necessary CODE modules and link them to DLL

#-----------------------------------------------------------------------------------------

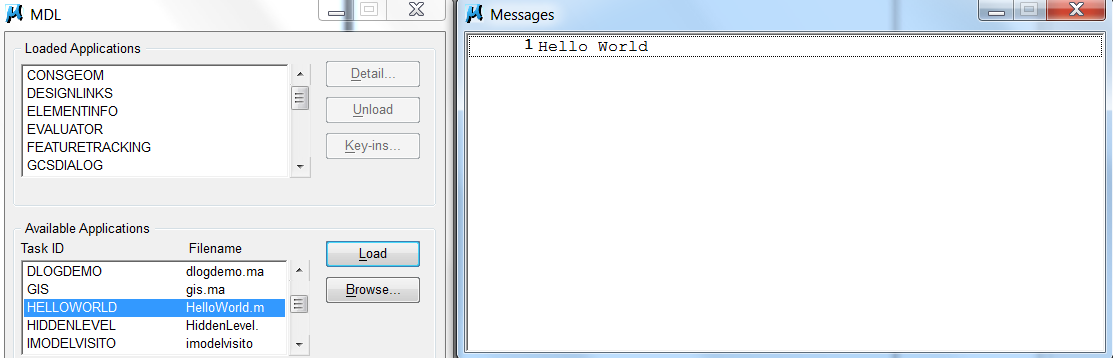
$(o)$(appName)$(oext) : $(baseDir)$(appName).cpp

%include dlmlink.mki

5、点击“开始 > 所有程序 > Bentley > MicroStation V8i (SELECTseries 3) SDK > MicroStation Developer Shell”启动MDL程序开发环境。在命令提示符后键入CD /D D:\MDLSource\HelloWorld并回车进入我们的项目所在目录，然后再键入bmake –a来生成HelloWorld.ma和HelloWorld.dll。这些生成的文件位于…\MicroStation\mdlapps目录下。

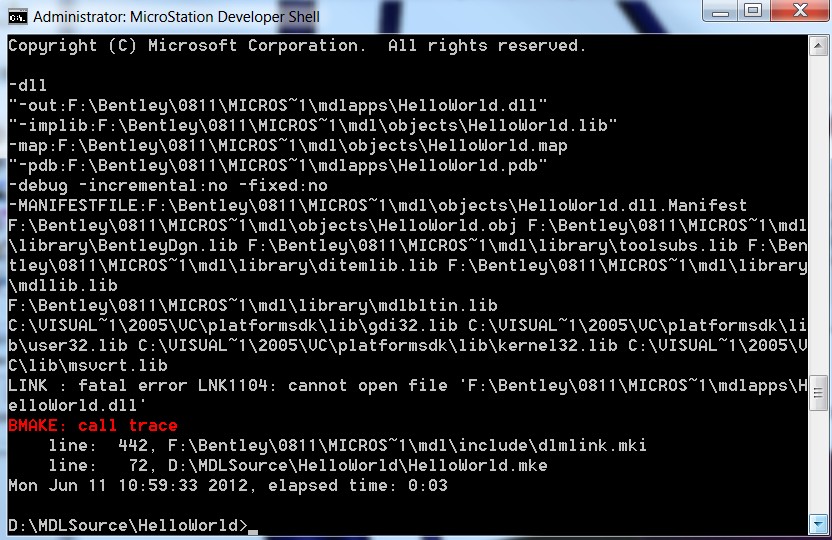


6、启动MicroStation，选菜单Utilities > MDL Applications打开MDL对话框，在该对话框下部的Available Applications列表框中找到并选中HELLOWORLD，然后点按钮Load装载我们的MDL应用程序。此时会看到一个写有Hello World字样的消息框弹出。如下图所示：



7、在您想要重新生成您的HelloWorld项目前需要首先卸载该应用，否则将很可能遇到如图所示的错误提示。卸载方法是在以上MDL对话框上部的Loaded Applications中选中HELLOWORLD，然后点击Unload按钮即可。

【技巧】：还可以通过键入命令来实现MDL应用程序的装载和卸载。选菜单Utilities > Key-in打开键入命令对话框，输入MDL LOAD HelloWorld来装载应用，输入MDL UNLOAD HelloWorld来卸载应用。

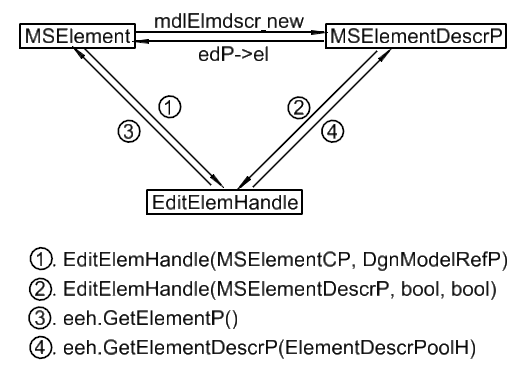


# 第二章、在MDL应用中创建元素

MicroStation有很多种类型的元素，我们可以将其分成图形元素和非图形元素。本章将讨论如何在MDL程序中创建图形元素。下表列出了MicroStation中支持的大多数图形元素：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素类型 | 图形表达 | 创建函数 | 是复杂元素吗 | 备注 |
| Line（线） |  | mdlLine\_create | 否 | 由两个点组成的一段直线。 |
| PointString  (点串) |  | mdlPointString\_create | 否 | 最多由2000多个点组成的离散点。源文档中的最多46个点有误，那是针对V7DGN而言的。  其实针对2D和3D V8DGN，分别为2728和2338 |
| LineString  （线串） |  | mdlLineString\_create | 否 | 最多由5000个点组成的多段直线。mdlElmdscr\_createFromVertices可以建立多于5000个点的复杂线串。 |
| Shape（形） |  | mdlShape\_create | 否 | 最多由5000个点组成的封闭直线段。mdlElmdscr\_createFromVertices可以建立多于5000个点的复杂形。mdlElmdscr\_createShapeWithHoles可建立中空的形。 |
| Arc（弧） |  | mdlArc\_create  mdlArc\_createByCenter  mdlArc\_createByDEllipse3d  mdlArc\_createByPoints | 否 |  |
| Ellipse（椭圆） |  | mdlEllipse\_create  mdlCircle\_createBy3Pts | 否 | 圆和椭圆是同一种元素类型（15） |
| ComplexChain  （复杂链）  ComplexShape  （复杂形） |  | mdlComplexChain\_createHeader | 是 | 该函数建立复杂头元素，随后要附加线、弧等元素构成最终的复杂链或复杂形。 |
| Curve（曲线） |  | mdlCurve\_create | 否 | 这是一类简单的曲线，曲线本身会经过您指定的点 |
| BsplineCurve  （B样条曲线） |  | mdlBspline\_createCurve  mdlBspline\_createInterpolationCurve | 是 | 可以先用mdlBspline\_createBsplineFromPointsAndOrder建立MSBsplineCurve结构。 |
| MultiLine（多线） |  | mdlMline\_create | 否 | 最多可由16条平行线组成的多段直线。 |
| Text（文本） | Abc | mdlText\_create  mdlText\_createWide | 否 |  |
| TextNode  （文本节点） | Line1  Line2 | mdlTextNode\_create/mdlTextMode\_createEx  mdlTextNode\_createWide  mdlTextNode\_createWithStrings  mdlTextNode\_createWithStringsWide | 是 | 多行文字或单行中具有多个文字样式的文字叫做文本节点 |
| Tag（标签） | Tag1 | mdlTag\_create | 否 | 与图形元素关联的一种文字 |
| Dimension（标注） |  | mdlDim\_create | 否 | 标注元素看起来样子虽然很复杂，但在MicroStation中却是一个简单元素。 |
| Cell（单元） |  | mdlCell\_create  mdlCell\_placeCell  mdlCell\_getElmDscr  mdlCell\_getElmDscrExtended | 是 | mdlCell\_create是不依赖单元库建立单元头的函数，mdlCell\_placeCell或mdlCell\_getElmDscr能从单元库中直接读取一个单元。 |
| SharedCell（共享单元） |  | mdlSharedCell\_create | 否 | 多个共享单元实例共用一个共享单元定义，因而对于有多个相同复杂单元的情况，使用共享单元能大大节省模型容量。 |
| Cone（圆锥体） |  | mdlCone\_create | 否 | 该函数可创建圆柱体和圆锥体 |
| Surface或Solid  （拉伸和旋转面或体） |  | mdlSurface\_project  mdlSurface\_revolve | 是 | 可通过设置edP->el.ehdr.type = SOLID\_ELM生成体 |
| SmartSolid  （智能实体） |  | mdlKISolid\_makeCuboid  mdlKISolid\_makePrism  mdlKISolid\_makeSphere  mdlKISolid\_makeCone  mdlKISolid\_makeTorus  mdlKISolid\_makeWedge  mdlKISolid\_sweepBodyWire | 是 | 智能实体借助ParaSolid内核来处理几何图形。而普通实体则不借助。因此，对于一些基本的三维体，既可以用普通实体来创建也可以用智能实体来创建。前者的执行效率比后者要高，但不能表达复杂的实体。 |
| BsplineSurface  （B样条曲面） |  | mdlBspline\_createSurface  mdlBspline\_crossSectionSurface  mdlBspline\_gordonSurface  mdlBspline\_coonsPatch  mdlBspline\_nSidedPatch  mdlBspline\_skinPatch  mdlBspline\_ruledSurface  mdlBspline\_surfaceOfProjection  mdlBspline\_surfaceOfRevolution  mdlBspline\_catmullRomSurface | 是 | 精确的曲面表达方式 |
| Mesh（网格曲面） |  | mdlMesh\_newCoordinateBlock  mdlMesh\_newGrid  mdlMesh\_newPolyface  mdlMesh\_newRegularPolyhedron | 是 | 网格曲面最适合用来表达大型地模数据，精度会比B样条曲面差 |

在写代码前还需要介绍三个重要的结构体：MSElement、MSElementDescr和EditElemHandle。前两个是C结构，最后一个是C++结构（C++的结构和类很相似）。当我们创建一个简单元素时，可以将其保存到MSElement结构中，这是一个具有固定大小（最大为256K字节）的结构。对于复杂元素，由于其大小无法事先确定，所以需要采用一个指针（即MSElementDescr \*）来表达。这两个结构体在MDL C API中大量使用。EditElemHandle是MDL C++ API中引入的一个新的类，它是表达一个被修改元素的最理想方式。有关EditElemHandle的详细描述，请阅读MicroStationAPI.chm中的EditElemHandle Struct Reference一节。这三个结构的相互转换关系如下图所示：



从本文、帮助文档或例子代码中您可能已经注意到MSElement和MSElementDescr有多种变形写法，其实际定义如下表所示：

|  |  |
| --- | --- |
| 变形写法 | 实际定义 |
| MSElementP | MSElement \* |
| MSElementCP | MSElement const \* |
| MSElementDescrP | MSElementDescr \* |
| MSElementDescrCP | MSElementDescr const\* |
| MSElementDescrH | MSElementDescr \*\* |

了解了以上一些基本知识后我们就开始动手来编写生成元素的代码。为了简化起见，所有代码都没有错误检查。

1. 编辑HelloWorld.cpp文件，最终结果如下：

/\*-------------------------------------------------------------+

| HelloWorld.cpp |

+-------------------------------------------------------------\*/

#include <MicroStationAPI.h>

#include <msdialog.fdf>

#include <mselmdsc.fdf>

#include <msbsplin.fdf>

USING\_NAMESPACE\_BENTLEY\_USTN\_ELEMENT

double g\_1mu;

void createALine (DPoint3d \*basePtP)

{

MSElement myLine;

DPoint3d pts[2];

pts[0] = pts[1] = \*basePtP;

pts[1].x += g\_1mu\*2; pts[1].y += g\_1mu;

mdlLine\_create (&myLine, NULL, pts);

EditElemHandle eeh (&myLine, ACTIVEMODEL);

eeh.AddToModel (ACTIVEMODEL);

}

void createAComplexShape (DPoint3d \*basePtP)

{

MSElement el;

MSElementDescrP edP = NULL;

DPoint3d pts[3], tmpPt;

mdlComplexChain\_createHeader (&el, 1, 0);

mdlElmdscr\_new (&edP, NULL, &el);

pts[0] = pts[1] = pts[2] = \*basePtP;

pts[1].x += g\_1mu\*0.3; pts[1].y += g\_1mu\*0.7;

pts[2].x += g\_1mu; pts[2].y += g\_1mu;

mdlArc\_createByPoints (&el, NULL, pts);

mdlElmdscr\_appendElement (edP, &el);

tmpPt = pts[0]; pts[0] = pts[2]; pts[2] = tmpPt;

pts[1].x = pts[0].x; pts[1].y = pts[2].y;

mdlLineString\_create (&el, NULL, pts, 3);

mdlElmdscr\_appendElement (edP, &el);

EditElemHandle eeh (edP, true, false);

eeh.AddToModel (ACTIVEMODEL);

}

void createAProjectedSolid (DPoint3d \*basePtP)

{

MSElement el;

MSElementDescrP sectionEdP = NULL, solidEdP = NULL;

DPoint3d pts[6], pt1, pt2;

pts[0] = \*basePtP;

pts[1].x = pts[0].x; pts[1].y = pts[0].y - g\_1mu/2; pts[1].z = pts[0].z;

pts[2].x = pts[1].x + g\_1mu/2; pts[2].y = pts[1].y; pts[2].z = pts[0].z;

pts[3].x = pts[2].x; pts[3].y = pts[2].y - g\_1mu/2; pts[3].z = pts[0].z;

pts[4].x = pts[3].x + g\_1mu/2; pts[4].y = pts[3].y; pts[4].z = pts[0].z;

pts[5].x = pts[4].x; pts[5].y = pts[0].y; pts[5].z = pts[0].z;

pt1 = pt2 = pts[0];

pt2.z += g\_1mu;

mdlShape\_create (&el, NULL, pts, 6, 0);

mdlElmdscr\_new (&sectionEdP, NULL, &el);

mdlSurface\_project (&solidEdP, sectionEdP, &pt1, &pt2, NULL);

mdlElmdscr\_freeAll (&sectionEdP);

solidEdP->el.ehdr.type = SOLID\_ELM;

EditElemHandle eeh (solidEdP, true, false);

eeh.AddToModel (ACTIVEMODEL);

}

void createABsplineSurface (DPoint3d \*basePtP)

{

MSElement el;

MSElementDescrP edP = NULL;

MSBsplineSurface bsSurface;

MSBsplineCurve bsCurves[4];

DPoint3d arcPts[4][3];

arcPts[0][0] = arcPts[0][1] = arcPts[0][2] = \*basePtP;

arcPts[0][1].x += g\_1mu/2; arcPts[0][1].z += g\_1mu/2;

arcPts[0][2].x += g\_1mu;

arcPts[1][0] = arcPts[1][1] = arcPts[1][2] = arcPts[0][2];

arcPts[1][1].y += g\_1mu/2; arcPts[1][1].z += g\_1mu/2;

arcPts[1][2].y += g\_1mu;

arcPts[2][0] = arcPts[2][1] = arcPts[2][2] = arcPts[1][2];

arcPts[2][1].x -= g\_1mu/2; arcPts[2][1].z += g\_1mu/2;

arcPts[2][2].x -= g\_1mu;

arcPts[3][0] = arcPts[3][1] = arcPts[3][2] = arcPts[2][2];

arcPts[3][1].y -= g\_1mu/2; arcPts[3][1].z += g\_1mu/2;

arcPts[3][2] = \*basePtP;

for (int i=0; i<4; i++)

{

mdlArc\_createByPoints (&el, NULL, arcPts[i]);

mdlElmdscr\_new (&edP, NULL, &el);

mdlBspline\_convertToCurve (&bsCurves[i], edP);

mdlElmdscr\_freeAll (&edP);

}

if (SUCCESS == mdlBspline\_coonsPatch (&bsSurface, bsCurves))

{

mdlBspline\_createSurface (&edP, NULL, &bsSurface);

EditElemHandle eeh (edP, true, false);

eeh.AddToModel (ACTIVEMODEL);

mdlBspline\_freeSurface (&bsSurface);

}

for (int i=0; i<4; i++)

mdlBspline\_freeCurve (&bsCurves[i]);

}

extern "C" DLLEXPORT int MdlMain (int argc, char \*argv[])

{

g\_1mu = mdlModelRef\_getUorPerMaster(ACTIVEMODEL);

DPoint3d basePt = {0,0,0};

createALine (&basePt);

basePt.x += g\_1mu\*1.7; basePt.y -= g\_1mu\*0.3; basePt.z -= g\_1mu\*0.6;

createAComplexShape (&basePt);

basePt.x += g\_1mu\*1.5; basePt.y -= g\_1mu\*0.3; basePt.z -= g\_1mu\*0.6;

createAProjectedSolid (&basePt);

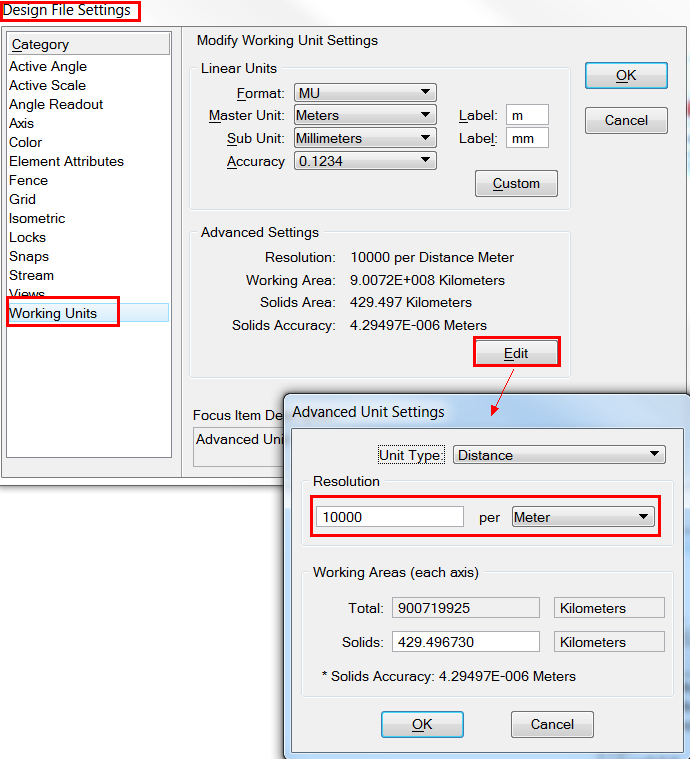
basePt.x += g\_1mu\*2.2; basePt.y -= g\_1mu\*1.7; basePt.z -= g\_1mu\*0.6;

createABsplineSurface (&basePt);

return 0;

}

* 1. 这段代码主要由四个函数组成，分别演示了如何创建一条线、一个复杂形、一个实体和一个B样条曲面。
  2. 在代码的开头包含了两个新的函数定义文件mselmdsc.fdf和msbsplin.fdf，它们分别包含了mdlElmdscr\_xxx和mdlBspline\_xxx函数的原型。
  3. 由于EditElemHandle类位于命名空间Bentley::Ustn::Element下，所以使用了宏定义USING\_NAMESPACE\_BENTLEY\_USTN\_ELEMENT。注意，这一行不需要用分号结束，因为宏定义中已经含有分号了。
  4. 由于MDL中默认的单位是UOR（Unit Of Resolution）而不是MicroStation中的主单位，所以我们定义了一个全程变量g\_1mu来代表一个主单位。在MdlMain函数中将其赋值为mdlModelRef\_getUorPerMaster(ACTIVEMODEL)。有关主单位、子单位和分辨率单位的关系，请参考下图（选菜单Settings > Design File可打开这个对话框）：



* 1. createALine函数建立一条线。mdlLine\_create函数根据两个点在MSElement结构体中生成这条线，然后据此建立一个EditElemHandle的实例，最后调用AddToModel方法加入到当前模型中。
  2. createAComplexShape函数建立一个由弧和线串组成的复杂形。首先建立一个复杂链头元素（mdlComplexChain\_createHeader的第二个参数1表示建立一个封闭形的头元素），紧接着将其转换为一个元素描述符，又分别建立弧元素和线串元素且将它们附加到这个元素描述符中。然后据此元素描述符建立一个EditElemHandle的实例，调用AddToModel方法将复杂形加入到当前模型中。注意，EditElemHandle eeh(edP, true,false)中的第二个参数true表示该元素句柄接管元素描述符edP。当eeh失效时会自动释放edP的，从而不再需要显式调用mdlElmdscr\_freeAll来释放edP了。
  3. createAProjectedSolid函数利用mdlSurface\_project函数根据一个断面来生成一个拉伸体。默认生成的是一个拉伸面，通过设置solidEdP->el.ehdr.type = SOLID\_ELM将拉伸面变成拉伸体。
  4. createABsplineSurface函数根据四个弧生成一个孔斯曲面。

1. 编辑HelloWorld.mke文件，修改DLM\_LIBRARY\_FILES定义如下：

DLM\_LIBRARY\_FILES = $(mdlLibs)BentleyDgn.lib \

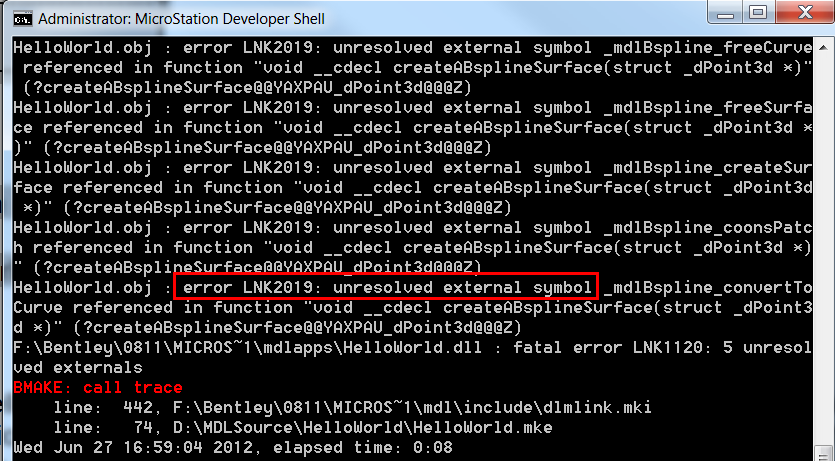
$(mdlLibs)toolsubs.lib \

$(mdlLibs)ditemlib.lib \

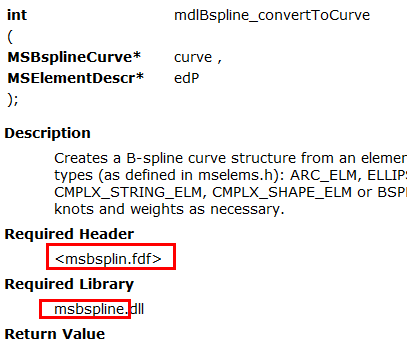
$(mdlLibs)mdllib.lib \

$(mdlLibs)msbspline.lib

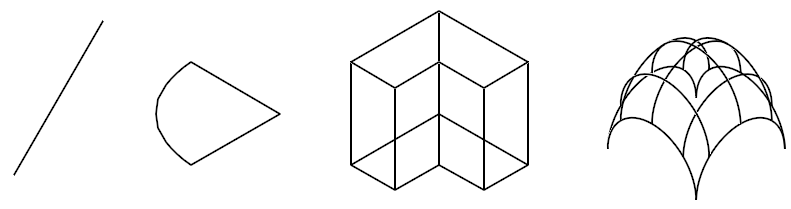
之所以要增加一个msbspline.lib是因为我们在代码中调用了mdlBspline类函数，而这些函数的内容位于msbspline.dll中。如果不增加这个库，执行bmake后会出现下图所示的链接错误：



从mdlBspline类函数的文档中可以知道要在.cpp中包含哪个头文件，在.mke中要指定哪个库。如下图所示：



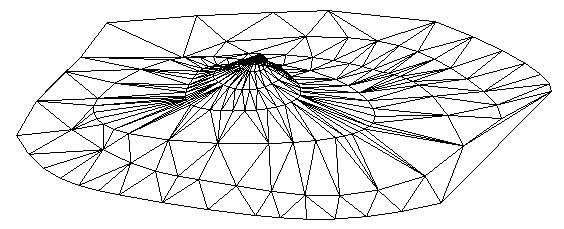
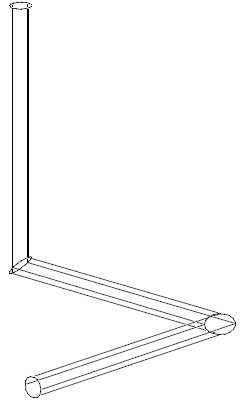
1. 在MicroStation Developer Shell中键入bmake –a来生成最终的HelloWorld.ma和HelloWorld.dll。
2. 启动MicroStation，打开或新建一个三维的模型，在键入命令域键入MDL LOAD HelloWorld并回车就能生成如下所示的图形。该图形是在ISO视图下看到的效果。



【技巧】：a. 在MicroStation Developer Shell黑窗口中可以通过键盘的上箭头键翻出前一个命令；

b. 在MicroStation的Keyin对话框中也可以通过键盘的上箭头键翻出前一个命令或直接双击以前键入的命令来执行；

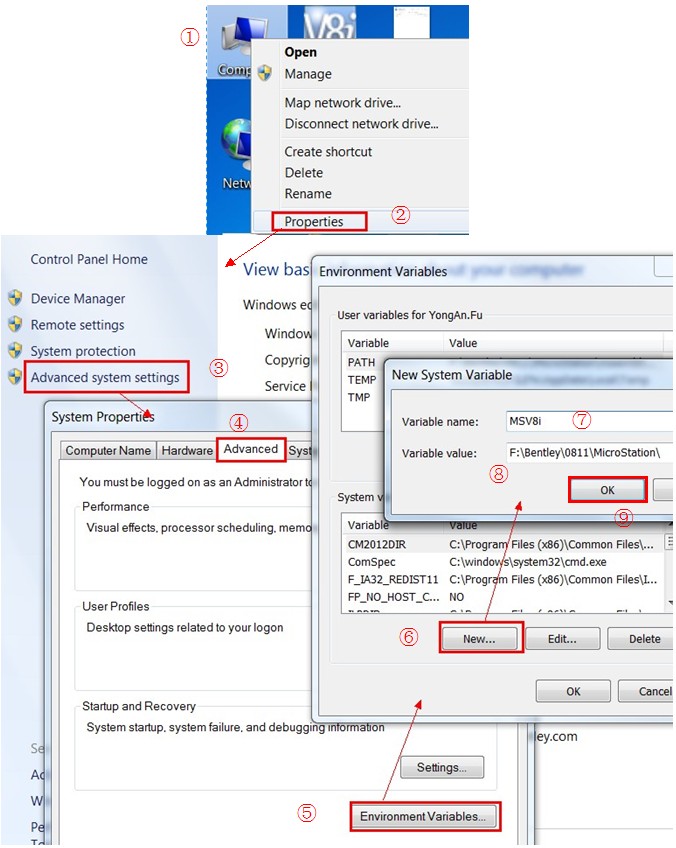
1. 有关建立网格曲面和智能实体的例子请见[Learning MicroStation Addins Step by Step[9]中的相关代码](http://communities.bentley.com/communities/other_communities/bdn_other_communities/b/bdn-blog/archive/2012/01/17/learning-microstation-addins-step-by-step-9.aspx#createMeshAndSolid)。将这些代码稍作修改即可生成如下所示的图形。不要犹豫，快自己动手吧！

# 第三章、在Visual Studio环境中生成并调试MDL应用

许多开发者多次问过我同样的一个问题：“如何在Visual Studio中直接生成程序？我不喜欢在那个黑窗口中键入bmake来生成。”这篇文章就带领您配置Visual Studio以达到这个目标。同时，还会演示如何在Visual Studio中调试我们的应用程序。

1. 在Windows操作系统级定义环境变量MSV8i指向您MicroStation V8iSS3的安装路径。如下图所示：



请注意，定义变量值时在路径最后加了一个路径分隔符。

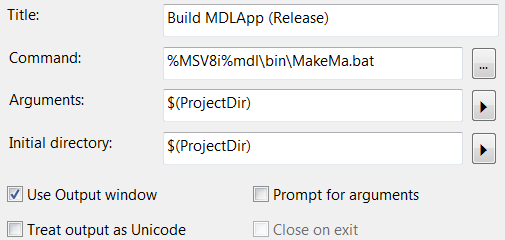
2. 在…\MicroStation\mdl\bin文件夹下创建一个批处理文件MakeMa.bat，其内容如下：

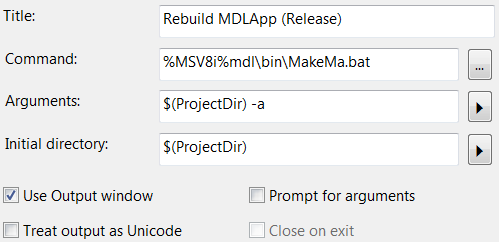
CALL "%MSV8i%mdl\bin\mstndevvars.bat" "%MSV8i%" "%MSV8i%"

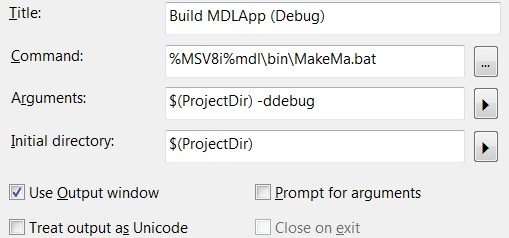
CD /D %1

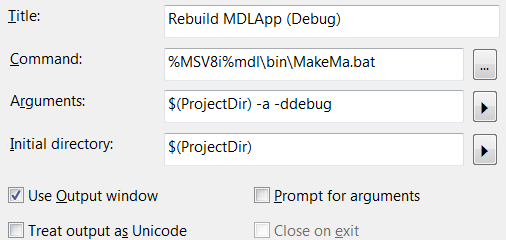
bmake %2 %3

3. 启动Visual Studio 2005，选菜单Tools > External Tools…定义如下四个外部工具。

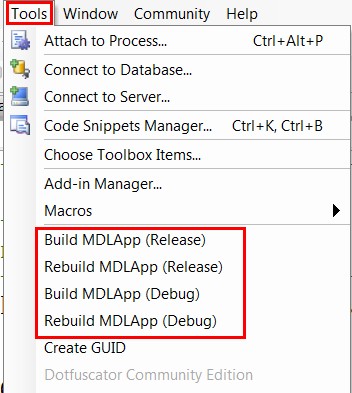




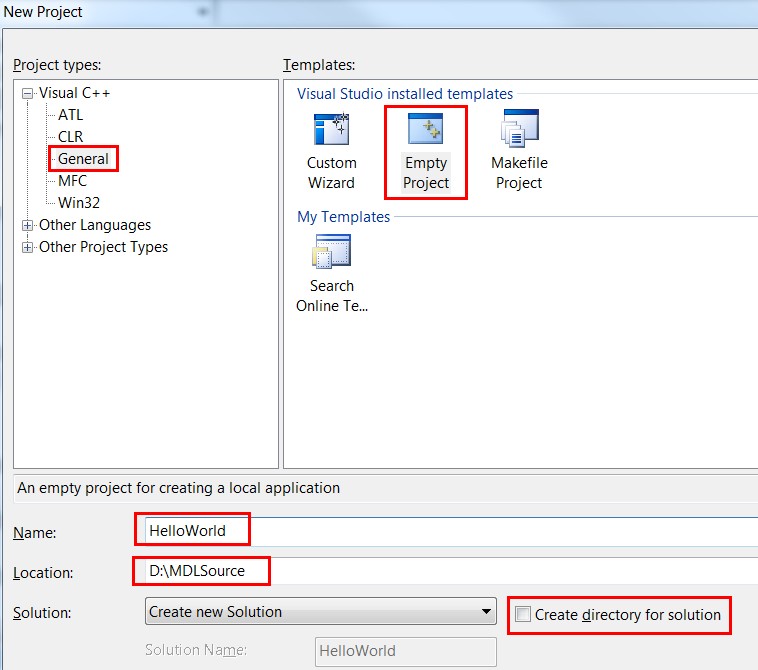




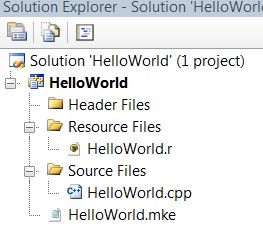
注意，在以上每个定义中都要勾选Use Output window选项，这样，在生成MDL应用程序过程中的所有提示（包括出错信息）就会在Visual Studio的输出窗口中显示出来。最终定义后的菜单如下：



4. 在Visual Studio中选菜单File > New > Project，然后按下图建立一个空项目。注意项目位置和项目名称的设置一定要与您已经建立的文件夹相匹配。同时不要钩选Create directory for solution项。

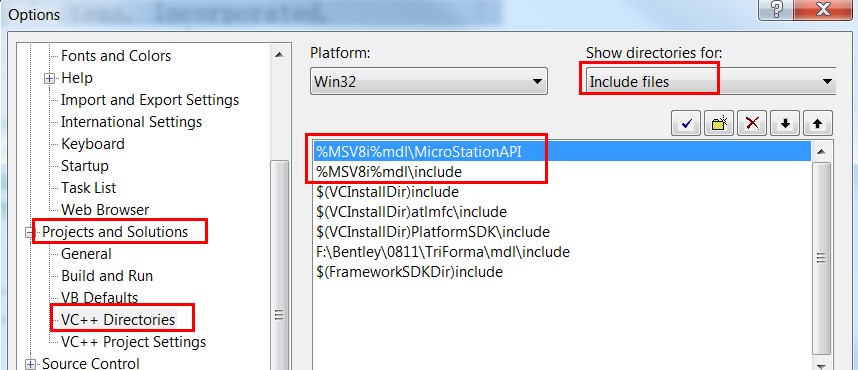


随后在该项目的解决方案浏览器中将HelloWorld.cpp添加到Source Files下，将HelloWorld.r添加到Resource Files下，将HelloWorld.mke添加到项目根下。最后的结果如下图所示。至此，我们完成了所有配置。注意，前三步骤只需要做一次，第四步需要对每个手工建立的项目做一次。下面就可以在Visual Studio中直接生成我们的MDL应用程序了。

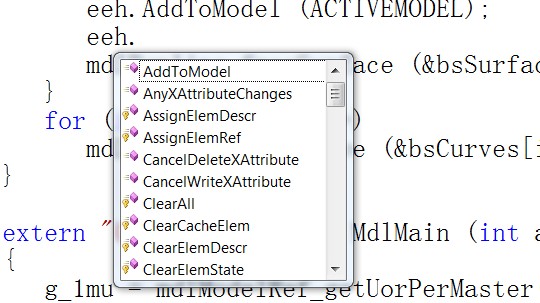


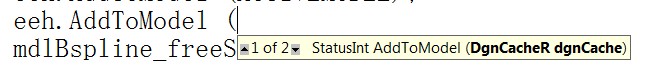
5. 请选择Tools下的Rebuild MDLApp (Release)，将会在输出窗口中看到生成程序的过程。如果您的源代码中有错误，则可通过双击输出窗口中的出错提示行而直接切换到源代码对应的行，这一点也是比用黑窗口要好得多。

6. 为了能进一步在Visual Studio中实现MDL函数的智能感知，需要我们选择菜单Tools > Options，然后按下图进行设置。



经过上面的设置后，当您在.cpp文件中输入代码时，会自动出现语法提示。如下图，当您在eeh后键入了点后，会弹出该对象的所有成员函数供您选择。当选择了AddToModel函数并输入了前括号后，又会出现函数的参数提示。这将大大提高我们的编程效率。





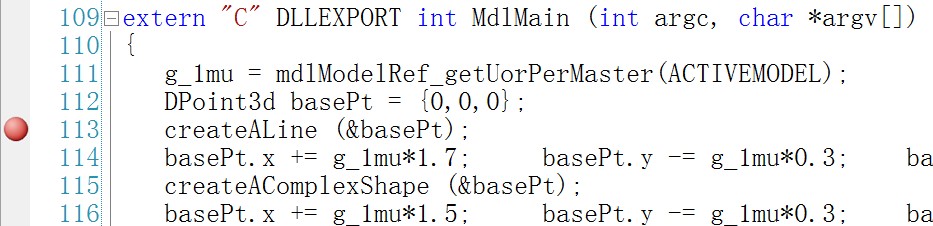
下面来介绍一下如何在Visual Studio中调试我们的MDL程序。

1. 启动MicroStation，如果您已经启动了MicroStation，则可忽略这一步。

2. 在Visual Studio中选菜单Tools > Rebuild MDLApp (Debug)以调试方式生成当前项目。

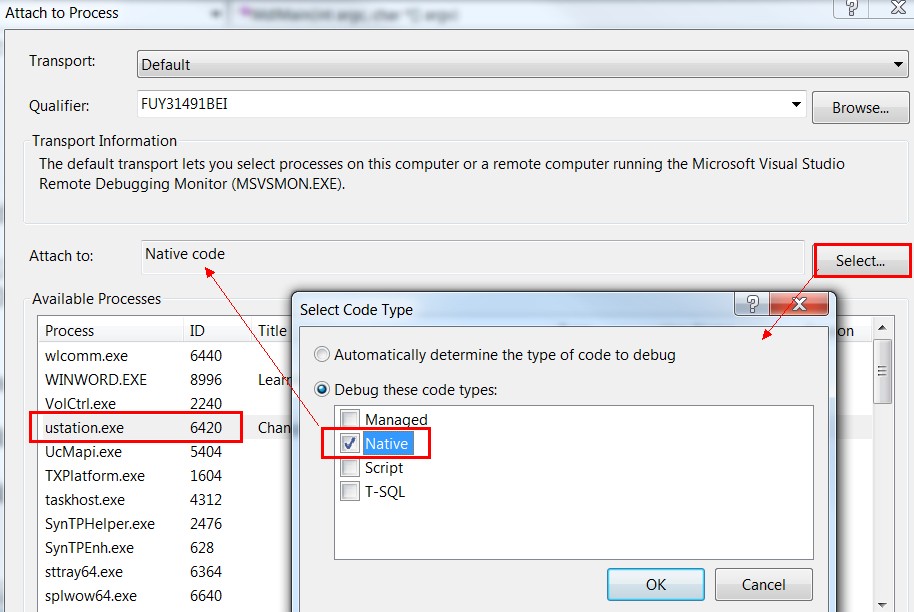
【注意】：由于以外部工具方式启动的生成命令不能像Visual Studio内置的生成命令那样会自动保存源文件，所以，在执行我们定义的这些外部工具命令前需要手工保存一下源代码。可以通过按Ctrl+S实现快速保存。

3. 在源代码的某行设置断点。方法是在该行最前列空白处点一下，此时该行最前列将会出现一个橘红色的圆钮。如下图所示：

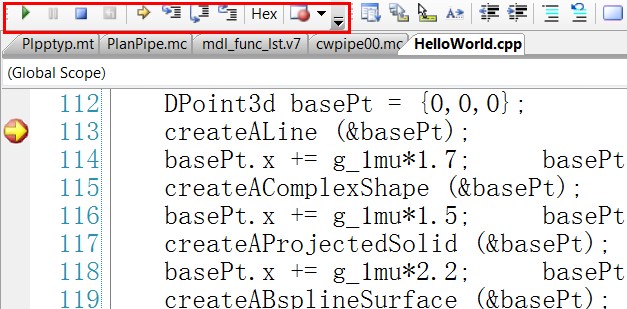


4. 选菜单Debug > Attach to Process…，在弹出的连接到进程对话框中的进程列表中找到并选中ustation.exe，确保Attach to:后是Native code，如果不是，可通过选择Select按钮来改变。然后点击Attach按钮。操作过程如下图所示：

【技巧】：可以用鼠标点击列表头Process使进程以降序排列，这样可以快速找到ustation.exe。



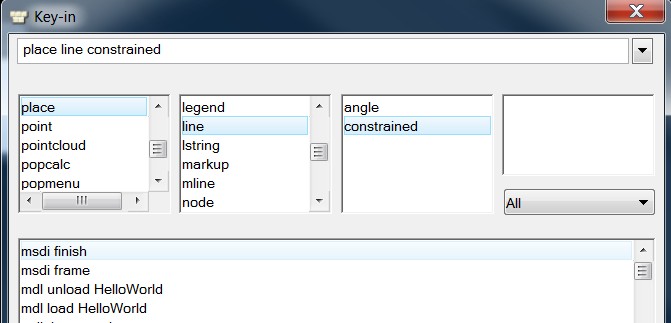
5. 切换到MicroStation，在键入域输入MDL LOAD HelloWorld，此时控制权会自动切换到Visual Studio并停在您设置的断点处。在调试工具栏上有许多工具可用，如进入型单步执行（Step Into）和越过型单步执行（Step Over）等等。如果您找不到这个工具栏，请选择Visual Studio菜单View > Toolbars > Debug。关于Visual Studio调试工具的详细用法请参见Visual Studio帮助文档。



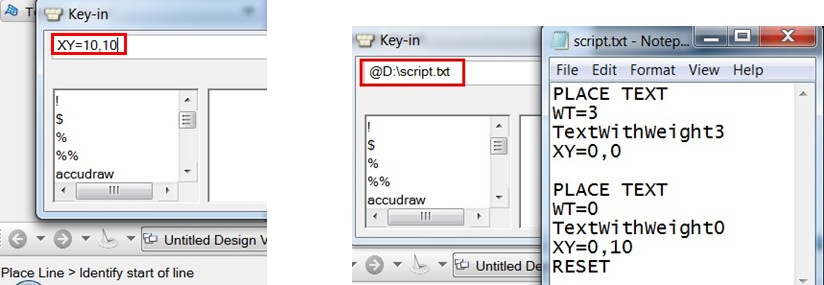
至此，所有的开发工作都可以在Visual Studio中完成了。包括编辑源代码、编译和链接程序以及调试程序。希望您喜欢这个完美的集成开发环境。

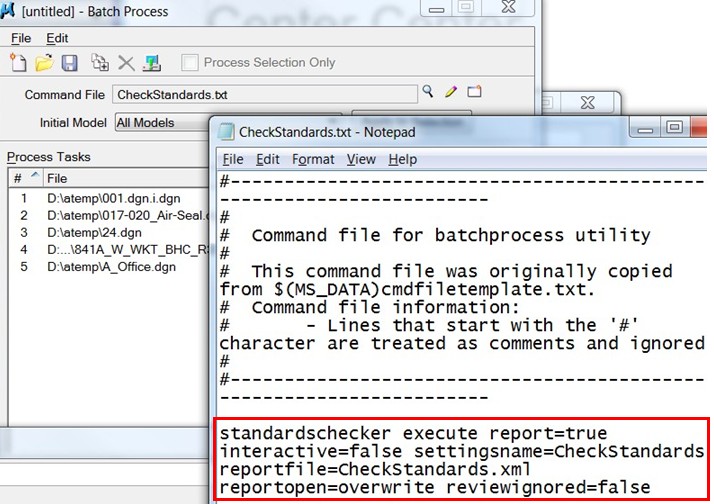
# 第四章、给MDL应用添加命令

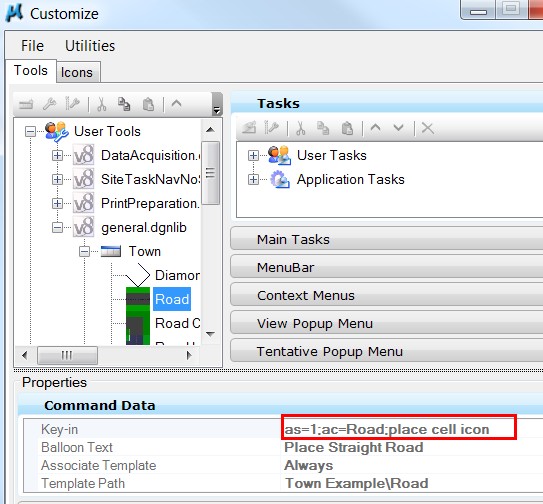
您可能已经注意到了MicroStation中存在着许多键入命令。选菜单Utilities > Key-in可打开键入命令对话框，在这里可浏览并执行当前可用的所有键入命令，如下图所示。虽然该对话框中只列出了四栏，但实际上每个键入命令最多可由5个单词组成。



这些键入命令有许多用途。比如在交互操作中可随时键入一个命令来实现操作的一个步骤，比如可以写一个由键入命令组成的脚本文件在MicroStation中直接执行，比如执行批处理功能时需要指定一个命令文件，再比如每个菜单项或工具栏中的每个图标工具，它们后台都关联着键入命令。可以这样说，键入命令是MicroStation图形用户界面与后台执行代码的重要纽带。以上列举的键入命令的四种用途分别如下图所示：

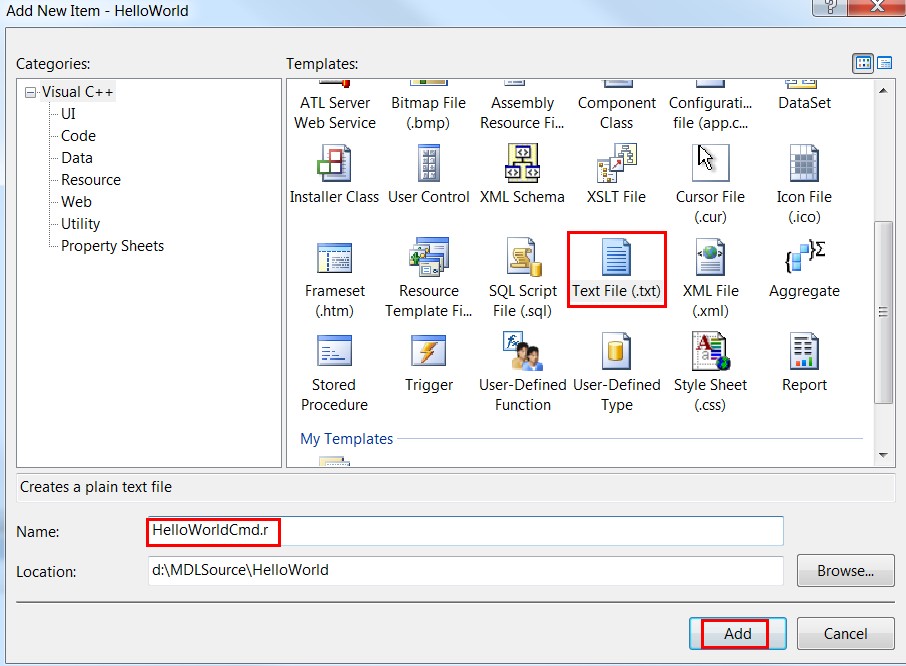






下面我们就一步步地在现有HelloWorld项目的基础上对其增加命令功能。

1. 在Visual Studio中打开您的HelloWorld项目，在解决方案浏览器中右击Resource Files文件夹并在弹出的菜单中选择Add New Item，在出现的窗体中找到并选中Text File (.txt)并在名称栏中输入HelloWorldCmd.r后点击Add按钮为当前工程添加一个空的.r文件。如下图所示：



2. 将如下命令表内容复制并粘贴到您的HelloWorldCmd.r文件中。保存该文件。

#include <rscdefs.h>

#include <cmdclass.h>

/\*----------------------------------------------------------------------+

| Local Defines |

+----------------------------------------------------------------------\*/

#define CT\_NONE 0

#define CT\_MAIN 1

#define CT\_SUB 2

#define CT\_CREATE 3

/\*----------------------------------------------------------------------+

| HelloWorld commands |

+----------------------------------------------------------------------\*/

/\*----------------------------------------------------------------------------------------------------+

| Table <tableId> = |

| { |

| { <no.>, <subTableId>, <cmdClass>, <options>, "<cmdWord>" }; |

| }; |

+---------------------------------------------------------------------------------------------------\*/

Table CT\_MAIN =

{

{ 1, CT\_SUB, PLACEMENT, REQ, "HelloWorld" },

};

Table CT\_SUB =

{

{ 1, CT\_CREATE, INHERIT, REQ, "Create" },

};

Table CT\_CREATE =

{

{ 1, CT\_NONE, INHERIT, NONE, "Line" },

{ 2, CT\_NONE, INHERIT, NONE, "ComplexShape" },

{ 3, CT\_NONE, INHERIT, NONE, "ProjectedSolid" },

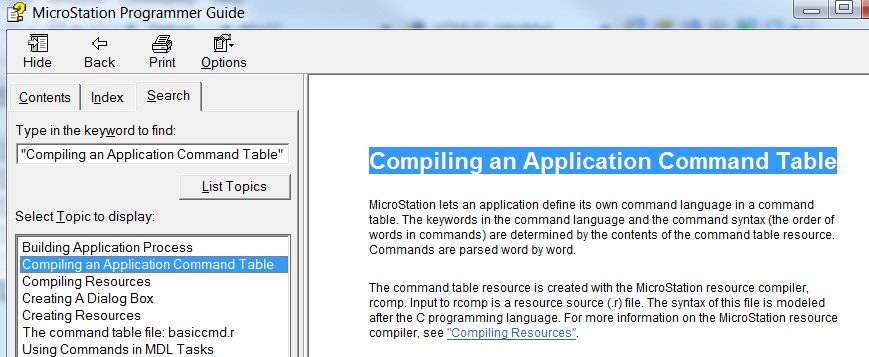
{ 4, CT\_NONE, INHERIT, NONE, "BsplineSurface" },

};

以上命令表文件定义了如下6个键入命令，它们分别有对应的命令号。

|  |  |  |
| --- | --- | --- |
| **键 入** | **命名号** | **命令号的值** |
| HelloWorld | CMD\_HELLOWORLD | 0x01000000 |
| HelloWorld Create | CMD\_HELLOWORLD\_CREATE | 0x01010000 |
| HelloWorld Create Line | CMD\_HELLOWORLD\_CREATE\_LINE | 0x01010100 |
| HelloWorld Create ComplexShape | CMD\_HELLOWORLD\_CREATE\_COMPLEXSHAPE | 0x01010200 |
| HelloWorld Create ProjectedSolid | CMD\_HELLOWORLD\_CREATE\_PROJECTEDSOLID | 0x01010300 |
| HelloWorld Create BsplineSurface | CMD\_HELLOWORLD\_CREATE\_BSPLINESURFACE | 0x01010400 |

有关命令表的语法和各项的含义，请参见MDLProgrammerGuide帮助中的Compiling an Application Command Table 一节。



3. 在Visual Studio中双击HelloWorld.mke打开制作文件并进行如下两处改动：

a. 将第8行的appRscs = $(o)$(appName).rsc改为

appRscs = $(o)$(appName).rsc $(o)$(appName)cmd.rsc

b. 将第48和50行前的注释符#删除。

$(baseDir)$(appName)cmd.h : $(baseDir)$(appName)cmd.r

$(o)$(appName)cmd.rsc : $(baseDir)$(appName)cmd.r

【注】：以上代码中含有两个很重要的空行，不能随手删掉它们。空行表示按默认规则来生成目标文件。在编译过程中bmake将会到mdl.mki文件中找到.r.h和.r.rsc规则去编译并生成HelloWorldCmd.h和HelloWorldCmd.rsc。HelloWorldCmd.h文件会在HelloWorld.cpp中引用，HelloWorldCmd.rsc被加入到appRscs宏定义中，从而最终被打包到HelloWorld.ma中。

4. 下面要对HelloWorld.cpp进行多处改动：

a. 包含HelloWorldCmd.h:

#include <MicroStationAPI.h>

#include <msdialog.fdf>

#include <mselmdsc.fdf>

#include <msbsplin.fdf>

#include "HelloWorldCmd.h"

b. 在MdlMain函数删除原来的一些函数调用，增加打开资源文件、装载命令表以及注册命令号等动作如下：

extern "C" DLLEXPORT int MdlMain (int argc, char \*argv[])

{

g\_1mu = mdlModelRef\_getUorPerMaster(ACTIVEMODEL);

RscFileHandle rscFileH;

mdlResource\_openFile (&rscFileH, NULL, RSC\_READ);

mdlParse\_loadCommandTable (NULL);

Private MdlCommandNumber commandNumbers [] =

{

{createALine, CMD\_HELLOWORLD\_CREATE\_LINE},

{createAComplexShape, CMD\_HELLOWORLD\_CREATE\_COMPLEXSHAPE},

{createAProjectedSolid, CMD\_HELLOWORLD\_CREATE\_PROJECTEDSOLID},

{createABsplineSurface, CMD\_HELLOWORLD\_CREATE\_BSPLINESURFACE},

0

};

mdlSystem\_registerCommandNumbers (commandNumbers);

return 0;

}

以上代码中的mdlResource\_openFile会打开资源文件HelloWorld.ma，然后mdlParse\_loadCommandTable从中装载命令表资源。

动态数组commandNumbers中定义了命令处理函数和每个命令号的对应关系，然后由mdlSystem\_registerCommandNumbers注

册。我们没有像大多MDL例子中那样同时还定义一套命令名，然后用mdlSystem\_registerCommandNames来注册，因为命令

名的使用远没有键入命令来得方便。只有在您没有定义命令表的情况下才有必要使用命令名。命令名的使用样例可参见SDK

中的cellexp.cpp、gis.cpp和toolExample.cpp。

c. 对createALine、createAComplexShape、createAProjectedSolid和createABsplineSurface进行一些改造以满足作为命令处理函数的要求。同时，将basePt的赋值内置在每个函数中。修改后的代码如下（省略掉了未改动部分）：

void createALine (char \*unparsed)

{

MSElement myLine;

DPoint3d basePt={0,0,0}, pts[2];

pts[0] = pts[1] = basePt;

……

}

void createAComplexShape (char \*unparsed)

{

MSElement el;

MSElementDescrP edP = NULL;

DPoint3d basePt, pts[3], tmpPt;

basePt.x = 1.7\*g\_1mu; basePt.y = -0.3\*g\_1mu; basePt.z = -0.6\*g\_1mu;

mdlComplexChain\_createHeader (&el, 1, 0);

mdlElmdscr\_new (&edP, NULL, &el);

pts[0] = pts[1] = pts[2] = basePt;

……

}

void createAProjectedSolid (char \*unparsed)

{

MSElement el;

MSElementDescrP sectionEdP = NULL, solidEdP = NULL;

DPoint3d basePt, pts[6], pt1, pt2;

basePt.x = 3.2\*g\_1mu; basePt.y = -0.6\*g\_1mu; basePt.z = -1.2\*g\_1mu;

pts[0] = basePt;

……

}

void createABsplineSurface (char \*unparsed)

{

MSElement el;

MSElementDescrP edP = NULL;

MSBsplineSurface bsSurface;

MSBsplineCurve bsCurves[4];

DPoint3d basePt, arcPts[4][3];

basePt.x = 5.4\*g\_1mu; basePt.y = -2.3\*g\_1mu; basePt.z = -1.8\*g\_1mu;

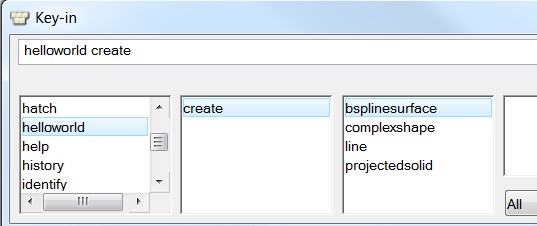
arcPts[0][0] = arcPts[0][1] = arcPts[0][2] = basePt;

……

}

5. 在Visual Studio中选菜单Tools > Rebuild MDL App(Debug)生成项目，然后切换到MicroStation装载HelloWorld。

6. 选MicroStation主菜单Utilities > Key-in打开键入对话框，输入HelloWorld后会看到如下图所示的命令键入树，这与我们在HelloWorldCmd.r中定义的完全对应。此时输入其中的一个命令并回车就能绘出相应的一部分图形。

****

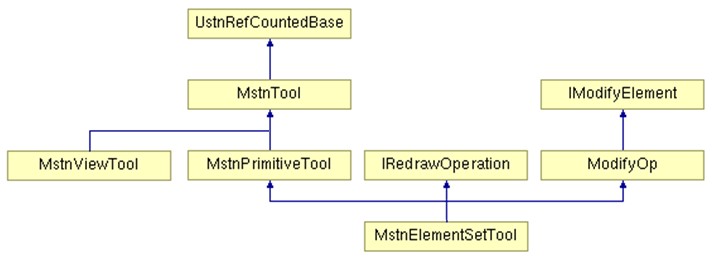
# 第五章、用MstnTool类实现交互式工具

上一章通过键入命令可实现放置各种图形，但这些图形的位置都是固定的。那么，能否在输入命令后让图形跟着光标走，定位后在该点放置图形呢？答案无疑是肯定的。

在面向对象的MicroStation C++API推出之前，MDL C API中有mdlState\_xxx类函数可以实现与MicroStation的交互。在新的C++API中，提供了功能强大的MstnTool类来实现类似的功能。新旧API对照如下：

|  |  |  |
| --- | --- | --- |
| **MDL C API** | **MicroStation C++ API** | **Example** |
| mdlState\_startViewCommand | MstnViewTool |  |
| mdlState\_startPrimitive | MstnPrimitiveTool |  |
| mdlState\_startModifyCommand  mdlState\_startFenceCommand | MstnElementSetTool |  |

新的C++ API都是以类的形式提供的，上表中重要的三个类都派生于MstnTool基类，在派生树中还涉及到其他一些重要的类和接口。如下图所示：



这些类中含有丰富的的成员函数，有些是设置一些参数的，如AllowSelection；有些是启动一个动作的，如BeginComplexDynamics；还有一些是处理用户事件的，如OnDataButton。限于篇幅，我们不在此列出这些类中的所有成员函数了，请您参考MicroStationAPI.chm文档了解它们的详细内容。

要实现一个交互式工具，我们需要定义一个派生于MstnViewTool、MstnPrimitiveTool或MstnElementSetTool的类。然后在命令处理函数中创建该类的一个实例并通过调用InstallTool()函数启动工具。

下面我们一步步将HelloWorld Create BsplineSurface命令改造成一个具有交互能力的工具。

1. 在Visual Studio中打开HelloWorld项目，将以下代码插入到HelloWorld.cpp文件的头部并保存。

USING\_NAMESPACE\_BENTLEY

USING\_NAMESPACE\_BENTLEY\_USTN

USING\_NAMESPACE\_BENTLEY\_USTN\_ELEMENT

double g\_1mu;

struct PlaceBsSurfaceTool: MstnPrimitiveTool

{

PlaceBsSurfaceTool (int cmdNumber, int toolName, int toolPrompt) :

MstnPrimitiveTool (cmdNumber, toolName, toolPrompt) {}

virtual bool CreateBsSurface (EditElemHandleR eeh, DPoint3dCP basePt)

{

MSElement el;

MSElementDescrP edP = NULL;

MSBsplineSurface bsSurface;

MSBsplineCurve bsCurves[4];

DPoint3d arcPts[4][3];

arcPts[0][0] = arcPts[0][1] = arcPts[0][2] = \*basePt;

arcPts[0][1].x += g\_1mu/2; arcPts[0][1].z += g\_1mu/2;

arcPts[0][2].x += g\_1mu;

arcPts[1][0] = arcPts[1][1] = arcPts[1][2] = arcPts[0][2];

arcPts[1][1].y += g\_1mu/2; arcPts[1][1].z += g\_1mu/2;

arcPts[1][2].y += g\_1mu;

arcPts[2][0] = arcPts[2][1] = arcPts[2][2] = arcPts[1][2];

arcPts[2][1].x -= g\_1mu/2; arcPts[2][1].z += g\_1mu/2;

arcPts[2][2].x -= g\_1mu;

arcPts[3][0] = arcPts[3][1] = arcPts[3][2] = arcPts[2][2];

arcPts[3][1].y -= g\_1mu/2; arcPts[3][1].z += g\_1mu/2;

arcPts[3][2] = \*basePt;

for (int i=0; i<4; i++)

{

mdlArc\_createByPoints (&el, NULL, arcPts[i]);

mdlElmdscr\_new (&edP, NULL, &el);

mdlBspline\_convertToCurve (&bsCurves[i], edP);

mdlElmdscr\_freeAll (&edP);

}

if (SUCCESS == mdlBspline\_coonsPatch (&bsSurface, bsCurves))

{

mdlBspline\_createSurface (&edP, NULL, &bsSurface);

edP->h.dgnModelRef = ACTIVEMODEL; // This line is IMPORTANT !

eeh.SetElemDescr (edP, true, false);

mdlBspline\_freeSurface (&bsSurface);

}

for (int i=0; i<4; i++)

mdlBspline\_freeCurve (&bsCurves[i]);

return eeh.IsValid();

}

virtual void OnPostInstall () override

{

BeginComplexDynamics ();

mdlAccuSnap\_enableSnap (true);

\_\_super::OnPostInstall();

}

virtual void OnRestartCommand () override

{

PlaceBsSurfaceTool \*pTool = new PlaceBsSurfaceTool (GetCmdNumber(), GetCmdName(), GetCmdPrompt());

pTool->InstallTool();

}

virtual void OnComplexDynamics (MstnButtonEventCP ev) override

{

EditElemHandle eeh;

if (!CreateBsSurface (eeh, ev->GetPoint()))

return;

RedrawElems redrawElems;

redrawElems.SetDynamicsViews (ev->GetViewport ());

redrawElems.SetDrawMode (DRAW\_MODE\_TempDraw);

redrawElems.SetDrawPurpose (DRAW\_PURPOSE\_Dynamics);

redrawElems.DoRedraw (eeh);

}

virtual bool OnDataButton (MstnButtonEventCP ev) override

{

EditElemHandle eeh;

if (CreateBsSurface (eeh, ev->GetPoint()))

eeh.AddToModel (ACTIVEMODEL);

OnReinitialize ();

return true;

}

virtual bool OnResetButton (MstnButtonEventCP ev) override

{

EndComplexDynamics ();

ExitTool ();

return true;

}

};

下面我们逐点解释以上代码的含义：

1. 增加了对两个新的命名空间的引用，一个是Bentley，一个是Bentley::Ustn;
2. 由于我们要实现一个放置工具，所以定义了一个派生于MstnPrimitiveTool的类PlaceBsSurfaceTool；
3. 构造函数PlaceBsSurfaceTool具有三个参数：命令号、工具名和工具提示。您可能会疑惑，为何工具名和工具提示也是整数型参数，其实它们仅仅是工具名标识和工具提示标识而并非具体的字符串，具体字符串的内容会保存在资源文件的消息列表中。本章随后的内容还会详细解释其中的细节；
4. CreateBsSurface是类PlaceBsSurfaceTool特有的成员函数，它几乎是前一章createABsplineSurface函数的翻版，但没有直接将生成的元素句柄eeh添加到模型中而是作为参数返回；
5. 重写了继承来的OnPostInstall函数。该函数会在工具加载后执行。在这个函数中调用了成员函数BeginComplexDynamics启动动态，还调用了全局函数mdlAccuSnap\_enableSnap启动精确捕捉；
6. OnRestartCommand是一个必须要重写的纯虚函数。当在外部事件（如Undo）打断了当前工具时会调用该函数；
7. 调用了BeginComplexDynamics后，当光标在视图区移动时，OnComplexDynamics函数会被反复调用。参数ev返回了当前光标位置等信息，我们利用该位置作为基点来创建B样条曲面eeh，然后利用RedrawElems类在视图区以临时元素方式重绘该元素句柄；
8. 当用户在视图区点鼠标的数据键（默认为左键）后会调用到OnDataButton函数，在该函数中才真正将生成的B样条曲面eeh添加到模型中。我们还调用了Reinitialize函数来重新初始化工具对象；
9. 当用户在视图区点鼠标的拒绝键（默认为右键）后会调用到OnResetButton函数，在该函数中停止动态并退出本工具。

2. 修改createABsplineSurface函数为如下内容：

void createABsplineSurface (char \*unparsed)

{

PlaceBsSurfaceTool \*pTool = new PlaceBsSurfaceTool (0, 0, 0);

pTool->InstallTool();

}

3. 选菜单Tools > Build MDLApp(Debug)编译HelloWorld；

4.启动MicroStation并进入一个三维模型，装载HelloWorld应用后键入HelloWorld Create BsplineSurface并回车启动命令，此时能看到屏幕上一个三维的B样条曲面跟随着光标在移动，当您在视图中点鼠标左键接受时，就会在您指定的点放置一个B样条曲面；

至此，我们已经实现了交互的主要功能。可能您会注意到，在工具工作过程中状态栏没有任何提示信息，工具设置对话框的标题栏显示的只是键入命令本身而不是一个更直观的命令，这说明我们的工具还不够完美。下面让我们继续改造HelloWorld使其完全像MicroStation的一个工具那样工作。

5. 回到Visual Studio中，在Header Files下新建一个HelloWorld.h头文件，然后在其中输入如下内容并保存。

#define STRINGLISTID\_Commands 1

#define STRINGLISTID\_Prompts 2

#define CMDNAME\_PlaceBsSurfaceTool 1

#define PROMPT\_PlaceBsSurfaceTool 1

6. 在HelloWorld.r中输入以下内容并保存：

MessageList STRINGLISTID\_Commands =

{

{

{ CMDNAME\_PlaceBsSurfaceTool, "Place a Bspline Surface" },

}

};

MessageList STRINGLISTID\_Prompts =

{

{

{ PROMPT\_PlaceBsSurfaceTool, "Identify a point" },

}

};

此处定义的内容就是1c中提及的工具名标识和工具名字符串，以及工具提示标识和工具提示字符串。

【注】：为了简化项目，我们直接将这些内容放入了HelloWorld.r中。当您读SDK附带的例子时，您会注意到这些内容是单独定义在一个xxxmsg.r资源文件中的，该资源文件是与语言相关的且位于一个叫做english的文件夹下。这样便于项目的国际化。比如，当您想生成项目的法语版本时，您只需要将english文件夹复制成一个平级的french文件夹，然后将french文件下的所有文件中的英文字符串改成法文字符串，最后在mke文件中将langSpec = $(baseDir)english/定义改成langSpec = $(baseDir)french/并重新编译项目即可。

7. 在HelloWorldCmd.r文件的头部包含HelloWorld.h：

#include "HelloWorld.h"

然后对CT\_CREATE表的第4行修改如下：

{ 4, CT\_NONE, INHERIT, CMDSTR(CMDNAME\_PlaceBsSurfaceTool), "BsplineSurface" },

8. 对HelloWorld.cpp做三处改动：

a.在该文件头部包含HelloWorld.h：

#include "HelloWorld.h"

b. 修改createABsplineSurface函数内容如下：

void createABsplineSurface (char \*unparsed)

{

PlaceBsSurfaceTool \*pTool = new PlaceBsSurfaceTool (

CMD\_HELLOWORLD\_CREATE\_BSPLINESURFACE,

CMDNAME\_PlaceBsSurfaceTool,

PROMPT\_PlaceBsSurfaceTool);

pTool->InstallTool();

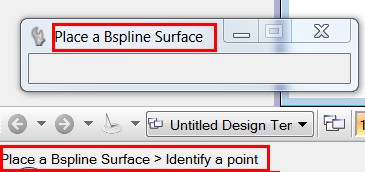
}

c. 在MdlMain函数中return 0前增加如下一行代码：

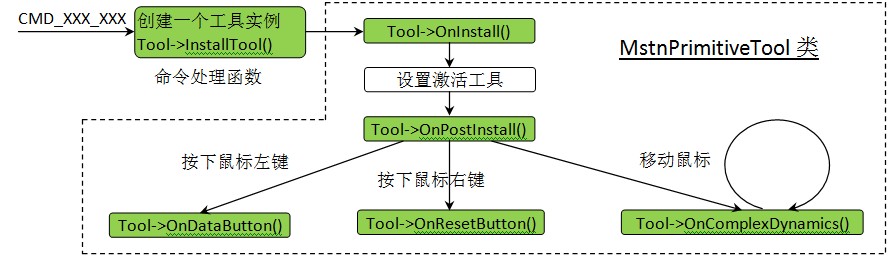
mdlState\_registerStringIds (STRINGLISTID\_Commands, STRINGLISTID\_Prompts);

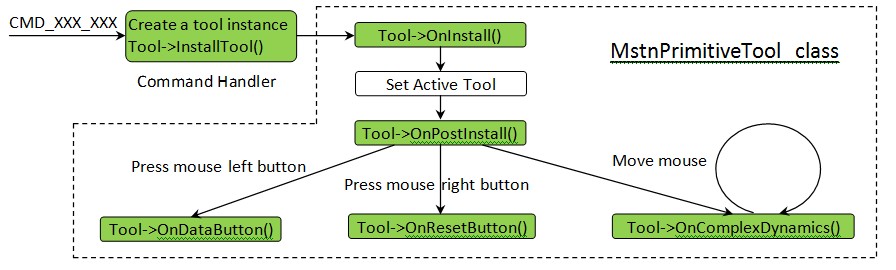
该函数在系统中注册了两个信息列表STRINGLISTID\_Commands和STRINGLISTID\_Prompts，一个供工具名（或称命令名）使用，一个供工具提示使用。这样，在new PlaceBsSurfaceTool时的第二个参数CMDNAME\_PlaceBsSurfaceTool和第三个参数PROMPT\_PlaceBsSurfaceTool将分别取自这两个消息列表。同样，在HelloWorldCmd.r中的CMDSTR后的参数CMDNAME\_PlaceBsSurfaceTool也是取自STRINGLISTID\_Commands。

9.在MicroStation中卸载HelloWorld，在Visual Studio中保存所有修改的文件，重新编译HelloWorld并进入MicroStation测试，您将看到状态栏有工具名和工具提示，工具设置对话框标题也改换成了我们的工具名而不再是原来的键入命令了。

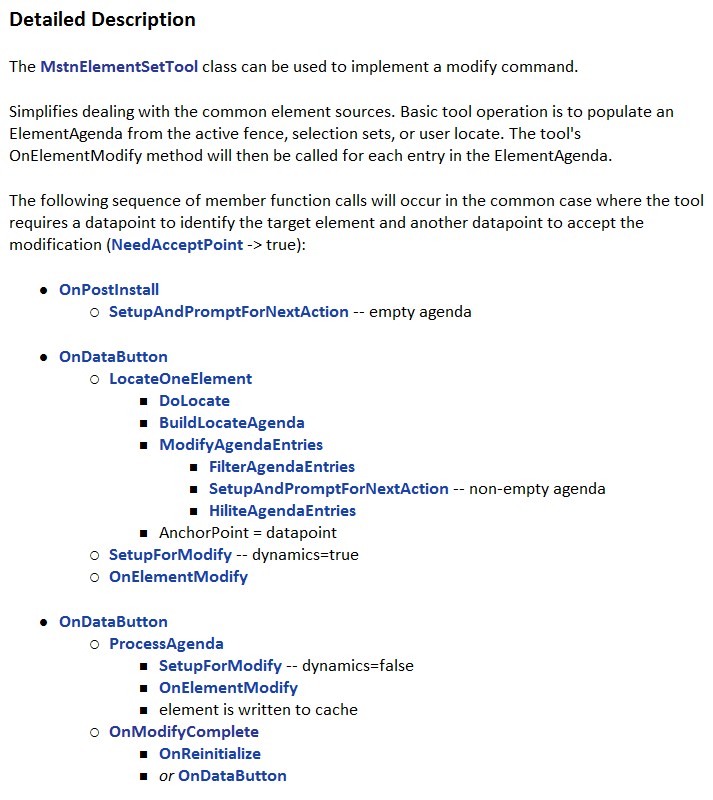


通过以上工具的实现，我们可以总结出一个简单的MstnPrimitiveTool工具工作示意图如下：





介绍完了MstnPrimitiveTool后我们再来介绍MstnElementSetTool。MstnElementSetTool类要比MstnPrimitiveTool类复杂得多，在MicroStationAPI帮助文档中有如下说明：



我们还是以一个例子来说明MstnElementSetTool的一个用法。该例子的功能是让用户选择多个元素，然后将选取元素时的一系列拾取点(HitPoint)连接成一条线串。在这个例子中还演示了如何使用派生于IElementState的对象。ElementState可用来表达附加于元素句柄上的一些额外数据。在本例中就是将HitPath作为这种额外数据保存在每个元素句柄上的。下面就一步步来改造我们的HelloWorld项目：

1. 在HelloWorld.cpp中增加如下内容：

//=================== ExampleLocateInfo class ======================

struct ExampleLocateInfo : RefCounted <IElementState>

{

ExampleLocateInfo () {m\_hitPath = NULL;}

~ExampleLocateInfo () {mdlDisplayPath\_release ((DisplayPathP)m\_hitPath);}

void SetHitPathCP (HitPathCP hitPath)

{

mdlDisplayPath\_release ((DisplayPathP)m\_hitPath);

m\_hitPath = hitPath;

mdlDisplayPath\_addRef ((DisplayPathP)m\_hitPath);

}

HitPathCP GetHitPathCP () {return m\_hitPath;}

private:

HitPathCP m\_hitPath;

}; // End of class ExampleLocateInfo

1. 以上代码定义了一个派生于IElementState接口的类ExampleLocateInfo；
2. 该类含有一个私有成员m\_hitPath，它 是指向HitPath的指针；
3. 构造函数中将该成员赋为空，析构函数中释放该成员；
4. 定义了取得和设置该成员的两个函数GetHitPathCP和SetHitPathCP。

2. 在HelloWorld.cpp中增加如下内容：

//=================== ConnectHitPointsTool class ======================

struct ConnectHitPointsTool : MstnElementSetTool

{

virtual bool OnInstall () override

{

if (!\_\_super::OnInstall ())

return false;

SetCmdNumber (CMD\_HELLOWORLD\_CONNECT\_HITPOINTS);

SetCmdName (CMDNAME\_ConnectHitPointsTool, PROMPT\_ConnectToolFirst);

return true;

}

virtual UsesSelection AllowSelection () override

{

return USES\_SS\_None;

}

virtual bool DoGroups () override

{

return false;

}

virtual bool WantDynamics () override

{

return false;

}

virtual bool NeedAcceptPoint () override

{

return true;

}

virtual void SetupAndPromptForNextAction () override

{

int count = GetElemAgendaP ()->GetCount ();

\_\_super::SetLocateCursor (count < 2 ? true : false, -1);

mdlOutput\_rscPrintf (MSG\_PROMPT, NULL, STRINGLISTID\_Prompts,

0==count ? PROMPT\_ConnectToolFirst :

(1==count ? PROMPT\_ConnectToolNext : PROMPT\_ConnectToolComplete));

}

virtual bool WantAdditionalLocate (MstnButtonEventCP ev) override

{

if (NULL == ev)

return true;

return (GetElemAgendaP ()->GetCount () < 2 || ev->IsControlKey ());

}

virtual bool OnModifierKeyTransition (bool wentDown, int key) override

{

if (CTRLKEY != key)

return false;

if (GetElemAgendaP ()->GetCount () < 2)

return false;

if (wentDown)

{

\_\_super::SetLocateCursor (true, -1);

mdlOutput\_rscPrintf (MSG\_PROMPT, NULL, STRINGLISTID\_Prompts, PROMPT\_ConnectToolNext);

}else{

\_\_super::SetLocateCursor (false, -1);

mdlOutput\_rscPrintf (MSG\_PROMPT, NULL, STRINGLISTID\_Prompts, PROMPT\_ConnectToolComplete);

}

return true;

}

virtual StatusInt OnElementModify (EditElemHandleR elHandle) override

{

return ERROR;

}

virtual EditElemHandleP BuildLocateAgenda (HitPathCP path, MstnButtonEventCP ev) override

{

EditElemHandleP eh = \_\_super::BuildLocateAgenda (path, ev);

if (eh)

{

ExampleLocateInfo\* locateInfo = new ExampleLocateInfo;

locateInfo->SetHitPathCP (path);

eh->SetIElementState(locateInfo);

}

return eh;

}

virtual bool OnModifyComplete (MstnButtonEventCP ev) override

{

if (GetElemAgendaP ()->GetCount () < 2)

return false;

EmbeddedDPoint3dArray\* pointsP = jmdlEmbeddedDPoint3dArray\_grab ();

EditElemHandleP curr = GetElemAgendaP ()->GetFirstP ();

EditElemHandleP end = curr + GetElemAgendaP ()->GetCount ();

for (; curr < end ; curr++)

{

ExampleLocateInfo\* info = dynamic\_cast <ExampleLocateInfo\*> (curr->GetIElementState ());

if (NULL != info)

jmdlEmbeddedDPoint3dArray\_addDPoint3d (pointsP, mdlHitPath\_getHitPoint (info->GetHitPathCP ()));

}

MSElement el;

MSElementDescrP edP = NULL;

mdlLineString\_create (&el, NULL, jmdlEmbeddedDPoint3dArray\_getPtr (pointsP, 0),

jmdlEmbeddedDPoint3dArray\_getCount (pointsP));

mdlElmdscr\_new (&edP, NULL, &el);

EditElemHandle eeh(edP, true, false);

eeh.AddToModel (ACTIVEMODEL);

return \_\_super::OnModifyComplete (ev);

}

virtual void OnRestartCommand () override

{

ConnectHitPointsTool \*newTool = new ConnectHitPointsTool ();

newTool->InstallTool();

}

}; //End of class ConnectHitPointsTool

这段代码的详细解释如下：

1. 定义一个派生于MstnElementSetTool的类ConnectHitPointsTool；
2. 在OnInstall事件处理函数中调用SetCmdNumber和SetCmdName设置本工具对应的命令号、工具名和工具提示；
3. 通过重写虚函数AllowSelection、DoGroups、WantDynamics和NeedAcceptPoint来设置本工具不支持选择集、不支持图形组和命名组（指当您选择一个元素时，不把该元素所在的图形组或命名组中的其他元素也加入到ElementAgenda中来）、不支持动态显示和需要接受更多的点；
4. SetupAndPromptForNextAction会在OnPostInstall和OnDataButton中被调用到，在该函数中根据不同的情况设置合适的工具提示。对于本例，有三个工具提示PROMPT\_ConnectToolFirst、PROMPT\_ConnectToolNext和PROMPT\_ConnectToolComplete会在不同的情况下被用到；
5. WantAdditionalLocate和OnModifierKeyTransition相配合用来支持在选取元素时按下Ctrl键来选择更多的元素；
6. 如果需要对每个选中的元素进行修改，可以在OnElementModify事件中处理，但本例是在选取完所有元素后才进行处理，所以，此处返回ERROR；
7. 由于我们希望记录每个元素的HitPath，所以，在截获BuildLocateAgenda后通过调用SetIElementState对每个ElemHandle附加一个ExampleLocateInfo指针；
8. 当选取了二个或二个以上的元素后，不按Ctrl键的情况下按下鼠标左键就会结束该工具，此时会调用到OnModifyComplete，在该函数中取得本工具关联的ElementAgenda，然后再取得其中每个EditElemHandle关联的HitPath，进一步取得该HitPath对应的HitPoint，将这些HitPath构造成一个线串元素添加到当前模型中；
9. 在OnRestartCommand函数中重新启动当前工具。

3. 在HelloWorld.cpp中增加一个新的命令处理函数如下：

void connectHitPoints (char \*unparsed)

{

ConnectHitPointsTool \*pTool = new ConnectHitPointsTool ();

pTool->InstallTool();

}

在命令号动态数组commandNumbers[]中增加如下一行：

{connectHitPoints, CMD\_HELLOWORLD\_CONNECT\_HITPOINTS},

4. 在HelloWorldCmd.r中增加一个新的命令HELLOWORLD CONNECT HITPOINTS。

……

#define CT\_CONNECT 4

…….

Table CT\_SUB =

{

{ 1, CT\_CREATE, INHERIT, NONE, "CREATE" },

{ 2, CT\_CONNECT, INHERIT, NONE, "CONNECT" },

};

……

Table CT\_CONNECT =

{

{ 1, CT\_NONE, INHERIT, CMDSTR(CMDNAME\_ConnectHitPointsTool), "HitPoints" },

};

5. 在HelloWorld.h中增加几个常量定义如下：

#define CMDNAME\_PlaceBsSurfaceTool 1

#define CMDNAME\_ConnectHitPointsTool 2

#define PROMPT\_PlaceBsSurfaceTool 1

#define PROMPT\_ConnectToolFirst 2

#define PROMPT\_ConnectToolNext 3

#define PROMPT\_ConnectToolComplete 4

6. 在HelloWorld.r中增加有关这个新命令的工具名和工具提示如下：

MessageList STRINGLISTID\_Commands =

{

{

{ CMDNAME\_PlaceBsSurfaceTool, "Place a Bspline Surface" },

{ CMDNAME\_ConnectHitPointsTool, "Connect Hit Points" },

}

};

MessageList STRINGLISTID\_Prompts =

{

{

{ PROMPT\_PlaceBsSurfaceTool, "Identify a point" },

{ PROMPT\_ConnectToolFirst, "Identify First Element" },

{ PROMPT\_ConnectToolNext, "Identity Next Element or RESET to reject" },

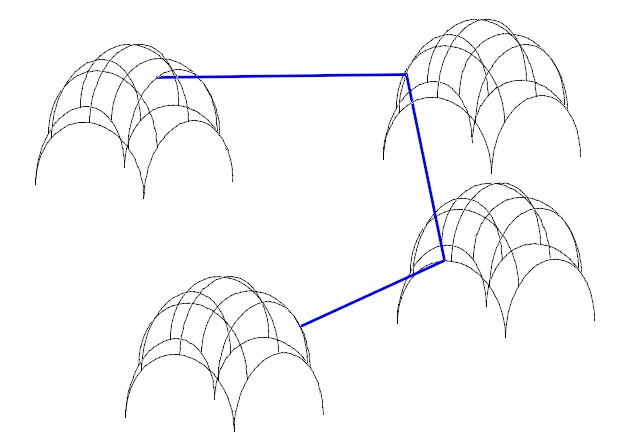
{ PROMPT\_ConnectToolComplete, "Accept/Reject (Ctrl + Data to identify additional elements)" },

}

};

7. 在Visual Studio中保存所有被修改的文件后选择Tools > Rebuild MDLApp (Debug)对项目进行重新编译。不要忘记在MicroStation中要先卸载HelloWorld应用；

8. 在MicroStation装载HelloWorld，然后键入HelloWorld Connect HitPoints并回车执行新的命令，选取几个元素后就能生成用HitPoints连成的线串。如下图所示：

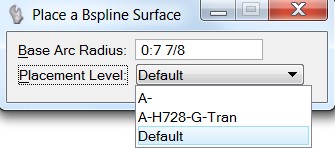


MstnElementSetTool还有许多其他用法，我们将会开辟专题来详细讲解。

完整的项目源代码可从本页的文件下载区下载。

# 第六章、给MDL应用添加用户界面

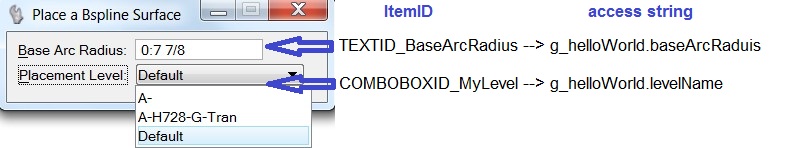
我们在本系列的[前一章](http://communities.bentley.com/communities/other_communities/bdn_other_communities/b/bdn-blog/archive/2012/11/30/mdl-5.aspx)实现了动态放置B样条曲线的功能。现在要给这个工具添加如下图形用户界面(GUI)：



增加这个界面后能帮助我们在放置曲面时指定曲面的大小和其所在的层。

在nativeCode（本机代码）应用程序中，可以通过MDL资源（.r文件）或MFC资源（.rc文件）来添加图形用户界面。前者是Bentley应用程序所特有的，后者是微软的实现方式。虽然MDL资源不支持可视化设计，但它是通过定义数据而不用写程序代码来支持用户交互的，这一点上反而比较简单。再者，您可以将MicroStation中已有的对话条目（即俗称的控件）集成到自己的对话框中，这也是MDL资源的另一大好处。总之，MDL资源在写nativeCode程序时占有极其重要的地位，Bentley的内部代码绝大多数都是通过MDL资源来写的界面，没有一个是用MFC资源写出来的（有一些是用最新的WinForm写出来的）。有关MDL资源的详细信息请参阅MicroStation程序员指南中的“Dialog Box Manager Overview（对话框管理器概述）”、“Standard Dialog Box Items（标准对话框条目）”、“Dialog Box Manager Hook Functions（对话框管理器钩函数）”和“Dialog Box Style Guidelines（设计对话框样式的指导原则）”。

在一篇文章中描述MDL资源的每一个细节几乎是不可能的，这个任务也不是本系列文章的要点。所以，在本例中我们将集中于以上两个对话条目。为了实现这个界面，我们要做一些设计方面的考虑：需要给每个对话框和对话条目赋一个标识（ID）以便在资源和代码中更容易地使用和引用；还需要给大多数对话条目指定“存取串（access string）”以便快速而直接地访问它们所引用的值。在本例中，所定义的条目标识和存取串如下图所示：



1. 在HelloWorld.h中定义如下内容：

#define TEXTID\_BaseArcRadius 1

#define COMBOBOXID\_MyLevel 1

#define HOOKITEMID\_MyLevelCombo 1

typedef struct helloworldinfo

{

double baseArcRadius;

char levelName[512];

}HelloWorldInfo;

* 对于同一种资源，各项的ID值必须唯一。不同资源类型的项的ID值不必唯一；
* 组合框条目COMBOBOXID\_MyLevel用来显示激活设计文件中的当前有效的层。这不是组合框条目的标准动作，因此需要为其添加一个钩函数，这就是为什么还要定义HOOKITEMID\_MyLevelCombo的原因；
* 通常情况下，所有的存取串都定义在一个结构体中并在.mt和.cpp文件中发布它。

2. 在HelloWorld.r中添加如下内容：

#include <dlogbox.h>

#include <dlogids.h>

#include “HelloWorldCmd.h”

CmdItemListRsc CMD\_HELLOWORLD\_CREATE\_BSPLINESURFACE =

{{

{{10\*XC, YC/2, 15\*XC, 0}, Text, TEXTID\_BaseArcRadius, ON, 0, "", ""},

{{10\*XC, YC\*2, 20\*XC, 0}, ComboBox, COMBOBOXID\_MyLevel, ON, 0, "", ""},

}};

DItem\_TextRsc TEXTID\_BaseArcRadius =

{

NOCMD, LCMD, NOSYNONYM, NOHELP, LHELP, NOHOOK, NOARG,

15, "%w", "%w", "0.0", "", NOMASK, TEXT\_NOCONCAT, "~Base Arc Radius:", "g\_helloWorld.baseArcRadius"

};

DItem\_ComboBoxRsc COMBOBOXID\_MyLevel =

{

NOCMD, LCMD, NOSYNONYM, NOHELP, MHELP, HOOKITEMID\_MyLevelCombo, NOARG,

MAX\_LEVEL\_NAME\_LENGTH, "%s", "%s", "", "", NOMASK,

0, 18, 4, 0, 0, COMBOATTR\_READONLY | COMBOATTR\_FULLWIDTH | COMBOATTR\_SORT,

" ~Placement Level:", "g\_helloWorld.levelName",

{

{0, MAX\_LEVEL\_NAME\_LENGTH, ALIGN\_LEFT, ""},

}

};

* 在命令CMD\_HELLOWORLD\_CREATE\_BSPLINESURFACE对应的CmdItemListRsc中定义了两个条目列表。当一个命令被处理并成为当前命令时，MicroStation会查找该命令关联的CmdItemListRsc资源并将其内容显示在工具设置（Tool Settings）对话框中；
* 我们为Base Arc Radius（基础弧半径）定义了一个文字条目资源。其中的15是控制在该文字条目中最大可输入的字符个数。“%w”表示该文字条目中的值将以工作单位格式显示。“~Base Arc Radius:”中的波折号表示B字符会带有下划线——这就是我们通常说的助记符。当工具设置对话框聚焦的情况下按Alt+B组合键就能选中该文字条目。

“g\_helloWorld.baseArcRadius”是该文字条目的存取串。我们可通过该变量获取或设置这个文字条目的值。

* 同理，我们为MyLevel定义了组合框条目资源。HOOKITEMID\_MyLevelCombo是与.cpp文件中钩函数对应的钩条目ID。通过这个ID将资源中的条目与.cpp中的函数关联了起来。

3. 如下建立一个MicroStation类型文件HelloWorldTyp.mt：

#include "HelloWorld.h"

publishStructures (helloworldinfo);

有关类型文件的详细内容请参阅MicroStation程序员指南中的“Generating Resource Files from C Type Definitions（从C类型定义生成资源文件）”一节。

4. 如下修改HelloWorld.mke文件：

a. 将编译后的资源HelloWorldTyp.rsc添加到宏定义appRscs中：

appRscs = $(o)$(appName).rsc $(o)$(appName)cmd.rsc $(o)$(appName)typ.rsc

b. 从HelloWorldTyp.mt生成HelloWorldTyp.r，编译HelloWorldTyp.r到HelloWorldTyp.rsc：

$(o)$(appName)typ.r : $(baseDir)$(appName)typ.mt

$(o)$(appName)typ.rsc : $(o)$(appName)typ.r

5. 对HelloWorld.cpp做多出修改：

a. 包含如下几个函数定义文件(.fdf)：

#include <listmodel.fdf>

#include <miscilib.fdf>

#include <leveltable.fdf>

b. 声明一个全局变量：

HelloWorldInfo g\_helloWorld;

c. 修改类PlaceBsSurfaceTool中的成员函数CreateBsSurface以响应工具设置中的两个参数：

virtual bool CreateBsSurface (EditElemHandleR eeh, DPoint3dCP basePt)

{

MSElement el;

MSElementDescrP edP = NULL;

MSBsplineSurface bsSurface;

MSBsplineCurve bsCurves[4];

DPoint3d arcPts[4][3];

**double radius = g\_helloWorld.baseArcRadius;**

arcPts[0][0] = arcPts[0][1] = arcPts[0][2] = \*basePt;

**arcPts[0][1].x += radius; arcPts[0][1].z += radius;**

**arcPts[0][2].x += radius\*2;**

arcPts[1][0] = arcPts[1][1] = arcPts[1][2] = arcPts[0][2];

**arcPts[1][1].y += radius; arcPts[1][1].z += radius;**

**arcPts[1][2].y += radius\*2;**

arcPts[2][0] = arcPts[2][1] = arcPts[2][2] = arcPts[1][2];

**arcPts[2][1].x -= radius; arcPts[2][1].z += radius;**

**arcPts[2][2].x -= radius\*2;**

arcPts[3][0] = arcPts[3][1] = arcPts[3][2] = arcPts[2][2];

**arcPts[3][1].y -= radius; arcPts[3][1].z += radius;**

arcPts[3][2] = \*basePt;

for (int i=0; i<4; i++)

{

mdlArc\_createByPoints (&el, NULL, arcPts[i]);

mdlElmdscr\_new (&edP, NULL, &el);

mdlBspline\_convertToCurve (&bsCurves[i], edP);

mdlElmdscr\_freeAll (&edP);

}

if (SUCCESS == mdlBspline\_coonsPatch (&bsSurface, bsCurves))

{

mdlBspline\_createSurface (&edP, NULL, &bsSurface);

edP->h.dgnModelRef = ACTIVEMODEL; // This line is IMPORTANT !

**LevelID levelId = LEVEL\_NULL\_ID;**

**MSWChar wLevelName[MAX\_LEVEL\_NAME\_LENGTH+1];**

**mdlCnv\_convertMultibyteToUnicode (g\_helloWorld.levelName, -1, wLevelName,**

**MAX\_LEVEL\_NAME\_LENGTH);**

**mdlLevel\_getIdFromName (&levelId, ACTIVEMODEL, LEVEL\_NULL\_ID, wLevelName);**

**mdlElmdscr\_setProperties (edP, &levelId, NULL, NULL, NULL, NULL, NULL, NULL, NULL);**

eeh.SetElemDescr (edP, true, false);

mdlBspline\_freeSurface (&bsSurface);

}

for (int i=0; i<4; i++)

mdlBspline\_freeCurve (&bsCurves[i]);

return eeh.IsValid();

}

d. 增加一个条目钩函数，该函数遍历当前设计文件中有效的层并将它们赋给组合框条目MyLevel：

void myLevel\_comboHook (DialogItemMessage \*dimP)

{

RawItemHdr \*riP = dimP->dialogItemP->rawItemP;

dimP->msgUnderstood = TRUE;

switch (dimP->messageType)

{

case DITEM\_MESSAGE\_CREATE:

{

MSWChar wLevelName[MAX\_LEVEL\_NAME\_LENGTH+1];

char levelName[MAX\_LEVEL\_NAME\_LENGTH+1];

ListModel \*pListModel = mdlListModel\_create (1);

LevelIteratorP pLevelIterator = mdlLevelIterator\_create (ACTIVEMODEL);

for (LevelID levelId = mdlLevelIterator\_getFirst(pLevelIterator); LEVEL\_NULL\_ID != levelId;

levelId = mdlLevelIterator\_getNext (pLevelIterator))

{

if (!mdlLevel\_isValid (ACTIVEMODEL, levelId) || mdlLevel\_isHidden (ACTIVEMODEL, levelId) ||

mdlLevel\_isElementLocked (ACTIVEMODEL, levelId))

continue;

mdlLevel\_getName (wLevelName, MAX\_LEVEL\_NAME\_LENGTH, ACTIVEMODEL, levelId);

mdlCnv\_convertUnicodeToMultibyte (wLevelName, -1, levelName,

MAX\_LEVEL\_NAME\_LENGTH);

mdlListModel\_insertString (pListModel, levelName, -1);

}

mdlLevelIterator\_free (&pLevelIterator);

mdlDialog\_comboBoxSetListModelP (riP, pListModel);

break;

}

case DITEM\_MESSAGE\_DESTROY:

{

ListModel \*pListModel = mdlDialog\_comboBoxGetListModelP (riP);

mdlListModel\_destroy (pListModel,TRUE);

break;

}

default:

dimP->msgUnderstood = FALSE;

break;

}

}

e. 定义一个全局动态数组uHook[]，并在MdlMain函数中发布它：

DialogHookInfo uHooks[] =

{

{HOOKITEMID\_MyLevelCombo, (PFDialogHook)myLevel\_comboHook },

};

……

……

mdlDialog\_hookPublish (sizeof(uHooks)/sizeof(DialogHookInfo), uHooks);

f. 在MdlMain函数中发布helloWorld结构并初始化其值：

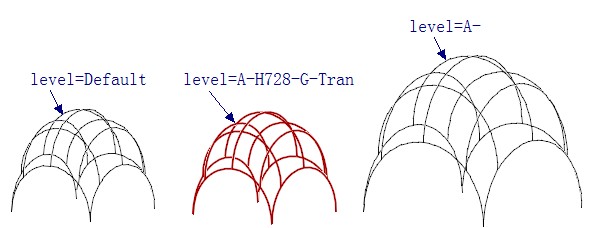
SymbolSet \*setP = mdlCExpression\_initializeSet (VISIBILITY\_DIALOG\_BOX, 0, 0);

mdlDialog\_publishComplexVariable (setP, "helloworldinfo", "g\_helloWorld", &g\_helloWorld);

g\_helloWorld.baseArcRadius = 2000;

strcpy\_s (g\_helloWorld.levelName, 512, "Default");

6. 在Visual Studio中保存所有文件，重生成该项目，然后切换到MicroStation来测试。当您装载HelloWorld应用并键入HELLOWORLD CREATE BSPLINESURFACE，您将会看到本章开头显示的第一个图片。下图是放置了三个曲面的例子，每个曲面都在不同的层上且具有不同的基本弧半径。



您或许注意到了，每当我们装载了HelloWorld应用后，g\_helloWorld总被初始化为一个固定的值，这并不是我们所期望的。我们期望该应用程序能保存并装载上次用户输入的值。要完成这个任务，请继续如下步骤：

7. 在HelloWorld.h中增加如下两行：

#define RTYPE\_HelloWorld 'Hello'

#define RSCID\_HelloWorld 1

8. 将HelloWorldInfo定义为一个资源类型并如下对其进行初始化：

resourceclass HelloWorldInfo RTYPE\_HelloWorld;

HelloWorldInfo RSCID\_HelloWorld = { 2000, "Default"};

9. 在HelloWorld.cpp中，我们在应用程序启动时从打开的资源文件中读取RSCID\_HelloWorld资源，然后为应用程序添加一个卸载钩函数，这个钩函数能在应用程序被卸载时将g\_helloWorld的当前值保存回到RSCID\_HelloWorld资源中。

a. 将局部变量rscFileH改为全局变量g\_rscFileH，因为我们需要在卸载函数中使用到它：

double g\_1mu;

HelloWorldInfo g\_helloWorld;

**RscFileHandle g\_rscFileH;**

b. 编写如下卸载函数：

Private int unloadFunction(int unloadType)

{

HelloWorldInfo \*helloRscP = NULL;

mdlResource\_closeFile (g\_rscFileH);

mdlResource\_openFile (&g\_rscFileH, NULL, RSC\_READWRITE);

helloRscP = (HelloWorldInfo \*)mdlResource\_load (g\_rscFileH, RTYPE\_HelloWorld, RSCID\_HelloWorld);

if (NULL == helloRscP)

mdlResource\_add (g\_rscFileH, RTYPE\_HelloWorld, RSCID\_HelloWorld, &g\_helloWorld, sizeof(HelloWorldInfo), NULL);

else {

\*helloRscP = g\_helloWorld;

mdlResource\_write (helloRscP);

mdlResource\_free (helloRscP);

}

mdlResource\_closeFile (g\_rscFileH);

return FALSE;

}

【注意】：为使我们的应用程序能在多个MicroStation中（指同一台电脑中启动了多份MicroStation）同时运行，资源在MdlMain中最开始以只读方式打开。当我们想在卸载函数中将g\_helloWorld的当前值写回到资源文件中时，需要首先关闭资源文件然后以读写方式重新打开它。

c. 在MdlMain函数中设置卸载函数：

mdlSystem\_setFunction (SYSTEM\_UNLOAD\_PROGRAM, unloadFunction);

d. 在MdlMain函数中装载资源RSCID\_HelloWorld到g\_helloWorld中：

HelloWorldInfo \*helloRscP = NULL;

if (g\_rscFileH)

helloRscP = (HelloWorldInfo \*)mdlResource\_load (g\_rscFileH, RTYPE\_HelloWorld, RSCID\_HelloWorld);

if (NULL != helloRscP)

{

g\_helloWorld = \*helloRscP;

mdlResource\_free (helloRscP);

}else{

g\_helloWorld.baseArcRadius = 2000;

strcpy\_s (g\_helloWorld.levelName, 512, "Default");

}

10. 重复第6步。修改Base Arc Radius和/或Placement Level条目的值，卸载应用，然后再重新装载它。你会发现上次输入的值被保留下来了。

至此为止，我们完成了本nativeCode MDL应用程序。本项目更新后的代码附下。

正如文章开头所提及的，我们不可能在一篇文章中包含MDL程序GUI设计的所有方面。一些重要的概念，如工具栏资源，没有在本文中涉及到。幸运的是，您可以使用MicroStation的定制功能将应用程序与MicroStation的任务系统集成。这能最大程度地减少写代码的工作量——创建一个DGN库文件（扩展名为.DGNLIB）并把定制内容放入其中。请参考MicroStation的帮助来学习这个非常简单而又非常强大的功能。

# 第七章、响应MicroStation中的事件

在MicroStation C API中，可以调用mdlXXX\_setFunction族中的某个函数（如mdlSystem\_setFunction、mdlInput\_setFunction、mdlState\_setFunction、mdlLocate\_setFunction、mdlView\_setFunction和mdlWindow\_setFunction等）来监控或钩住大多数事件。在MicroStation C++ API中，以上部分内容被封装到了不同的管理器（Manager）类或接口中。因而，您可以从管理器类或接口出发，通过添加一个处理器（Handler）或监控器（Monitor）来响应这些事件。下表是MicroStation C++ API中一些主要事件处理的列表。

|  |  |  |  |
| --- | --- | --- | --- |
| 类或接口 | 添加/移除处理器或监控器 | 处理器或监控器 | 描 述 |
| MSDocumentManager | AddDocumentMonitor  DropDocumentMonitor | MSDocumentMonitor | 监控文件的创建 |
| ISessionMgr | AddSessionMonitor  DropSessionMonitor | SessionMonitor | 监控会话期的动作，如打开主文件、激活一个模型、打开或关闭写锁定等 |
| ITxnManager | AddTxnMonitor  DropTxnMonitor | TxnMonitor | 监控撤销/恢复以及XAttribute的变化 |
| IVewManager | AddViewDecoration  DropViewDecoration | IViewDecoration | 在视口中以覆盖方式绘制修饰性图形（decoration） |
| AddElementDecorator  DropElementDecorator | IViewDecoration | 在视口中绘制修饰性图形，会考虑到Z缓冲区 |
| AddViewMonitor  DropViewMonitor | IViewMonitor | 监控视图事件，如打开或关闭视图、进入或退出视图、视图改变和取得视图标题等 |
| AddViewTransientHandler  DropViewTransientHandler | IViewTransients | 在视口中绘制临时（transient）图形 |
| INamedViewManager | AddEventHandler  RemoveEventHandler | INamedViewEvent  -Handler | 响应命名视图（NamedView）的创建、删除和修改事件 |
| AddDisplayElementEventHandler  None | INamedViewDisplay  -ElementEventHandler | 响应命名视图中元素显示的事件 |
| MstnEventManager | AddAccuSnapHandler  DropAccuSnapHandler | AccuSnapHandler | 响应精确捕捉（AccuSnap） |
| AddEditActionSource  DropEditActionSource | IEditActionSource | 响应一些编辑动作 |
| AddMouseWheelHandler  DropMouseWheelHandler | IMouseWheelHandler | 响应鼠标滚轮事件 |
| MstnGCS | SetEventHandler  RemoveEventHandler | IGeoCoordianteEvent  -Handler | 响应地理坐标系的修改和删除事件 |

使用以上事件处理的方法很简单——只需按如下三步做即可：

* 定义一个派生于基类处理器或监控器的类，然后在该类中实现您自己的动作；
* 声明一个该类的静态实例；
* 调用添加或移除处理器的函数来启动或终止事件处理。

作为一个例子，您可以用IViewTransient接口来给视图添加一个处理器。

基于本系列前面文章中用过的HelloWorld项目，按如下步骤来操作：

1. 在HelloWorldCmd.r中增加两个命令HELLOWORLD TRANSIENT ON和HELLOWORLD TRANSIENT OFF：

……

#define CT\_TRANSIENT 5

……

Table CT\_SUB =

{

……

{ 3, CT\_TRANSIENT, INHERIT, NONE, "TRANSIENT"},

};

……

Table CT\_TRANSIENT =

{

{ 1, CT\_NONE, INHERIT, NONE, "ON" },

{ 2, CT\_NONE, INHERIT, NONE, "OFF" },

};

2. 在HelloWorld.cpp中定义派生于IViewTransients的类MyViewTransients：

const int MAX\_PNT = 400000;

DPoint3d g\_pts[MAX\_PNT], g\_reducedPts[MAX\_PNT/100];

struct MyViewTransients : IViewTransients

{

void \_DrawTransients (ViewContextP context, bool isPreUpdate) override

{

if (isPreUpdate)

return; // Draw AFTER elements...

IViewDrawP output = context->GetIViewDraw ();

output->SetSymbology (0x0000ff00, 0, 4, 0); // opaque green (color is TBGR packed int), weight 4...

switch (context->GetDrawPurpose())

{

case DRAW\_PURPOSE\_UpdateDynamic:

case DRAW\_PURPOSE\_FitView: // Can support FitView for transient

output->DrawPointString3d (MAX\_PNT/100, g\_reducedPts, NULL);

break;

case DRAW\_PURPOSE\_Update:

output->DrawPointString3d (MAX\_PNT, g\_pts, NULL);

break;

}

}

};

* 在定义该类前需要先定义全局DPoint3d数组g\_pts和g\_reducedPts，它们分别代表一大堆点和对应的缩减数量后的点；
* 在MyViewTransients类中实现纯虚函数\_OnDrawTransients；
* 在调用DrawingPointString3d函数前需要从ViewContext参数获得指向IViewDraw的一个指针，然后设置要绘制的元素的线符；
* 通常情况下，用g\_pts来显示临时元素。但在用户的动态操作过程中，如旋转视图，将会用较少的点（即g\_reducedPts）来绘制该临时元素从而提高屏幕显示的性能。

3. 声明MyViewTransients的一个静态实例：

MyViewTransients s\_ViewTransient;

4. 实现两个命令处理函数：

void turnOnViewTransient (char \*unparsed)

{

double uor = mdlModelRef\_getUorPerMaster (ACTIVEMODEL);

for (int i=0; i<MAX\_PNT; i++)

{

g\_pts[i].x = uor \* rand(); g\_pts[i].y = uor \* rand(); g\_pts[i].z = uor \* rand();

}

int j = 0;

for (int i=0; i<MAX\_PNT; i+=100)

g\_reducedPts[j++] = g\_pts[i];

IViewManagerR viewMgr = IViewManager::GetManager();

viewMgr.AddViewTransientHandler (&s\_ViewTransient);

viewMgr.UpdateViews (NULL, false, DRAW\_MODE\_Normal, NULL, false);

}

void turnOffViewTransient (char \*unparsed)

{

IViewManagerR viewMgr = IViewManager::GetManager();

viewMgr.DropViewTransientHandler (&s\_ViewTransient);

viewMgr.UpdateViews (NULL, false, DRAW\_MODE\_Normal, NULL, false);

}

* 在turnOnViewTransient函数中，首先给g\_pts和g\_reducedPts赋了一些随机点；然后调用AddViewTransientHandler函数将s\_ViewTransient设为视图临时显示的处理器；最后，调用UpdateViews函数更新所有视图；
* 在turnOffViewTransient函数中，移去了该视图临时显示处理器并再次更新视图。

5. 在commandNumbers[]中将命令号和命令处理函数关联起来：

Private MdlCommandNumber commandNumbers [] =

{

……

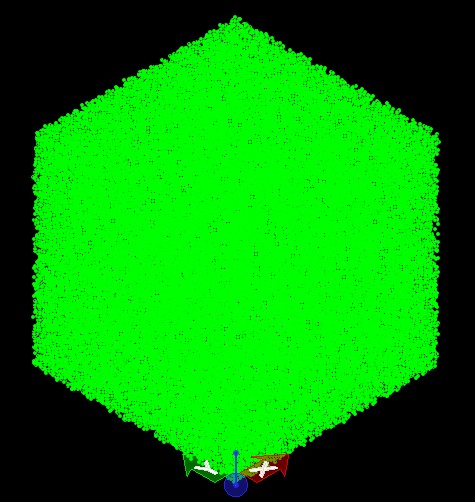
{turnOnViewTransient, CMD\_HELLOWORLD\_TRANSIENT\_ON},

{turnOffViewTransient, CMD\_HELLOWORLD\_TRANSIENT\_OFF},

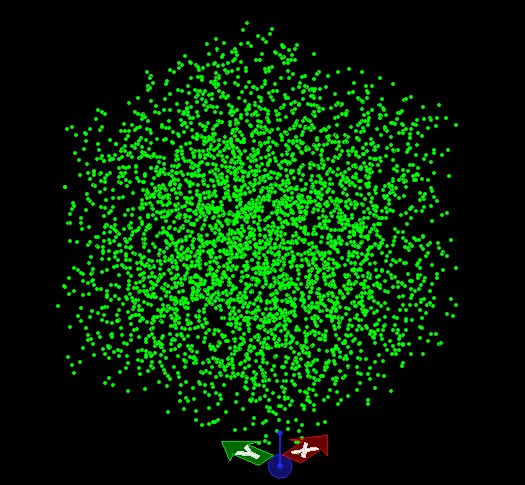
0

};

6. 在Visual Studio中保存所有文件，重新生成项目，切换到MicroStation中去测试。当装载HelloWorld并键入HELLOWORLD TRANSIENT ON后，您将能看到如下显示的画面。它包含了40万个点！



当旋转视图时，一个缩减版本的临时立方体如下图所示显示出来。当您停止了旋转，上图又重新显示出来。



我也附加了本项目更新后的源代码供大家参考。