第二十五章　ZEMAX 扩展

介绍

➡ 这个特性仅对 ZEMAX 的 EE 版本有效

ZEMAX 有一个功能非常强大的特性，它允许其他 Windows 程序建立一个与 ZEMAX 相连的信息传递链，并可以为这个程序从 ZEMAX 中获取有关镜头的数据。它的意思是一个其它程序可以使用 ZEMAX 来追迹光线通过镜头，然后将数据传输到这个程序中进行进一步的分析和计算。

有三种相互紧密联系的方法可以利用这一特性来扩展 ZEMAX 的这一性能：

第一，可以设计一个独立 Windows 程序来建立与 ZEMAX 相连的链，它被用来获取 ZEMAX 提供的有关镜头的数据—有代表性的是光线追迹数据和其他的光学数据。这个程序可以以它认为合适的方法来使用这些数据。

第二，一个独立程序可以做它自己的由用户“隐藏”的分析，然后在 ZEMAX 通常的图表和文本窗口中显示产生的数据。当使用这种模式时，这个程序被称为 ZEMAX 的一个“扩展程序”。扩展程序看起来和 ZEMAX 程序中的“扩展”菜单下的菜单选项一样。扩展程序必须放在目录\Extend 中，以便于 ZEMAX 来运行它们。

第三，独立程序可以用来计算 ZEMAX 可以优化的数据。在这种模式里，程序被称为“用户自定义操作数”或者 UDO。这些 UDO 必须放在目录\UDO 中，以便于 ZEMAX 来运行它们。

那些需要复杂的用户界面，带有许多输入、菜单、按钮、和复杂的输入和输出格式的应用程序比较适合于作为一个独立程序，它们不需要利用 ZEMAX 来显示计算结果。

那些可以进行复杂计算，但只需一个简单的对话框类型的用户界面的应用程序比较适合于结合 ZEMAX 来作为一个扩展程序。扩展程序有着和正规的 ZEMAX 分析程序十分相象的观察和操作优点。其执行特性，如可以在 ZEMAX 内观看扩展程序，更新，改变设置，被打

印，被复制到剪贴板上，或者被平移和缩放，和其他的 ZEMAX 分析窗口一样。

UDO 作为应用程序，可以计算那些必须由 ZEMAX 优化程序优化的数据。UDO 在“优化”一章中的“用户自定义”中被详细介绍。

编写得很好的应用程序甚至可以和一个独立程序或者扩展一样操作，如在范例一节中介绍的那样。

在应用程序和 ZEMAX 之间的信息传递可使用动态数据交换（DDE）来完成。DDE 是为了在程序间分享数据而在 Windows 操作系统内部定义的一个协议。两个程序可以建立一条 DDE 链，其中一个扮演“服务程序”，另一个扮演“客户程序”。客户程序通常向服务程序要求特殊的数据，而服务程序则将这些数据发送给客户程序。

ZEMAX 被作为服务程序，而其他任意一个 Windows 程序做一些正确的修改，可以被作为一个客户程序。这一章定义了连到 ZEMAX 的 DDE 界面，以便于用户可以编写软件程序来利用 ZEMAX 的光线追迹引擎。

编写扩展程序的要求

这一章和 DDE 特性的目的是帮助经验丰富的程序员编写可以与 ZEMAX 连接的编码。这一特性允许程序员编写第三种类型的程序，在与 ZEMAX 的联系中工作。第三种类型的程序可以将 ZEMAX 用作光线追迹引擎，而且提供了界面，图表，输入输出，和扩展计算，这些都作为分析的一个特殊类型被用户化。

使用这一特性需要一定的 Windows 和 C 语言编程的知识，至少要掌握有关信息传递，信息循环，存储器管理，和空间单元，操作，和指示的知识。包含了学习 DDE 编程所需的所有的信息的一本很好的书是由微软公司的 Charles Petzold 编写的《Programming Windows 95》。在这一章的正文中的范例和编码一节不是一个完整的程序，而是仅用来作为例子的编码片段。

虽然编写 Windows 应用程序稍微有些复杂，但比较好的是所有的困难编码都已经编好了，而且可以用来学习和利用。关于建立与

ZEMAX 相连的信息传递链的详细资料已被设计好，而且可以仅将样本编码做一些必要的修改后再“剪贴”成与特殊计算相关的编码。也由一个 C 语言源代码文件，称为 ZCLIENT，它通过将所有的信息压缩到函数包中来大大简化和 ZEMAX 之间的 DDE 信息传递。ZCLIENT 提供了少量的简单函数，它们可以被调用来从 ZEMAX 中获取数据，这不需要有关 DDE 编程的知识。ZCLIENT 将在这一章的后面部分进行介绍。ZEMAX 提供了有关客户程序和扩展程序的完整范例，包括源代码，将在这一章的最后说明。

扩展程序的能力在于它的价值，虽然它是合理的。使用 DDE 特性要求用户有一个相匹配的编译器或者发展工具，可以用来产生 32 位的 Windows 执行结果。这也是假设用户可以编写所要求的编码，最重要的是要确保编码是可靠的，没有错误的。为了使速度达到最大，ZEMAX 对由客户程序返回的数据不进行错误检查，因此，有错误的扩展程序将使 ZEMAX 计算失败，或者使 ZEMAX 进入一个无限循环。

由于这个原因，关于 DDE 或者扩展程序的执行的技术支持被严格限制来阐明提供的范例文件可以正确工作。如果你需要一个 ZEMAX 扩展程序，但没有愿望或能力自己来编写它们，请随意和 FSI 联系，提出开发一个自定义程序来满足你的要求。FSI 在开发这些类型的程序方面有相当丰富的经验，通常可以很快地以非常具有竞争性的价格编写扩展程序。

传递链的建立

为了建立与 ZEMAX 之间的 DDE 链，客户程序必须将一条包括有关应用程序名和主题名称的参考的信息传送到所有顶级窗口中。对于 ZEMAX 来说，应用程序名是“ZEMAX”，主题名称可以是任意一个非空的字符串。ZEMAX 不使用这个主题名称，只用应用程序名和“项目”。这项目给 ZEMAX 指出了哪些数据将被申请。ZEMAX 支持的各种项目将在后面说明。

特殊代码的例子可参见许多的样本 DDE 客户代码。

一旦建立了 DDE 链，服务程序可以执行任意基于光线追迹的分析，而且数据被返回到客户程序中。

传递链的终止

当所有的计算都完成了，DDE 链将由客户程序终止，客户程序将向 ZEMAX 服务程序发送一条终止信息。这条信息将仅仅终止 ZEMAX 的服务程序部分，而不是 ZEMAX 本身。

从 ZEMAX 中获取数据

ZEMAX 支持许多 DDE 下的许多功能。每个函数都被取了一个名称，被成为“项目”，通过 WM-DDE-REQUEST 信息将这个项目传送到 ZEMAX 中。在项目名称中通过编码将 ZEMAX 需要的数据（例如追迹一条光线）传送到 ZEMAX 中。大多数项目有一个简单的名称，不需要进一步编码，但一些项目有编码数值添加到名称中。这个编码数值被添加到项目名称中，用逗号分开。

ZEMAX 将用 WM-DDE-DATA 信息对每个项目请求做出回应，这个信息中包含一个指向一块用来存放字符串的存储器的指示器。大多数数据将放在字符串中从 ZEMAX 传送到客户应用程序中，这相当于 CF-TEXT 格式。这个客户应用程序必须记录这个字符串，并释放字符串的记忆内容。

关于特殊编码的例子可参见许多的样本 DDE 客户程序编码。

数据项目

这里是被支持的数据项目：

CloseUDOData

这个项目用来关闭用户自定义操作数缓冲器，这允许 ZEMAX 优化程序继续下去。仅当执行一个用户自定义操作数或者 UDO 时，才使用这个项目名。UDO 在“优化”一章中详细说明。详细内容可参见那一章。CloseUDOData 的语法结构是：

CloseUDOData，缓冲器代码

也可参见 GetUDOSystem 和 SetUDOItem。

DeleteSurface

这个项目用来删除一个已存在的表面。其语法结构为：

DeleteSurface，表面编号

也可参见 InsertSurface。

FindLabel

这个项目查找一个有与指定表面关联的整数标记的表面。其语法结构为：

FindLabel，标记

得到的数据是带有相同整数标记的第一个表面的表面编号，或者如果没有表面有指定的标记，则得到的数据为-1。也可参见 SetLabel 和 GetLabel。

GetAddress

这个项目记录了环境设置中第 1，2，3，或者 4 行的地址。其语法结构为：

GetAddress，1

对于第 2，3，和 4 行有着类似的命令。如果用户选择了隐藏这个地址栏，则所有字符串都将得到简单的“ \r\n ”。

GetAperture

这个项目记录了表面口径数据。其语法结构为：

GetAperture，表面编号

得到的字符串有着如下的格式：

“ 类型，最小值，最大值，x 偏心，y 偏心 ”

这个项目得到的类型是一个整数代码；0 代表没有口径，1 代表环形口径，2 代表环形挡光，3 代表星形，4 代表长方形口径，5 代表长方形挡光，6 代表椭圆口径，7 代表椭圆挡光，8 代表用户自定义口径，9 代表用户自定义挡光，以及 10 代表浮动口径。这个最小值和最大值对于椭圆、长方形、和星形口径有着与环形口径不一样的

意义；详细内容可参见编辑菜单一章。

GetAspect

这个项目记录了显示的图表的外表比率和以当前镜头长度单位表示打印纸的宽度或高度。例如，如果当前的外表比率是 3*4，则得到的外表比率将为 0.75。当画等大的图表时，需要知道正确的外表比率。得到的数据格式是外表比率，宽度。如果外表比率大于 1，那么该图用的是高度，而不是宽度，得到的数据的格式为外表比率，高度。

其语法结构为：

GetAspect，文件名

这里的文件名是与被创建或更新的窗口相联系的临时文件的名称。如果不用临时文件名，则得到默认的外表比率和宽度（或高度）。

GetBuffer

项目 GetBuffer 用来从一个被更新的窗口中得到客户程序指定的数据。其语法结构为：

GetBuffer，n，tempfile

这里 n 是缓冲器编号，它必须是在 0 到 15 之间，包括这两个数；tempfile 是与被更新的窗口相联系的临时文件的名称。当 ZEMAX 调用客户程序时，这个临时文件名将被传送到这个客户程序中；详细内容可参见“ ZEMAX 如何调用客户程序 ”的说明。注意，每个窗口都可能有它自己的缓冲器数据，ZEMAX 使用这个文件名来识别窗口，为此要求得到缓冲器内容。也可参见 SetBuffer。

GetComment

无论是什么，这个项目都记录表面的注释。要得到注释数据的表面编号被添加到项目名中，例如，要得到表面 5 的注释，项目名应该是“ GetComment，5 ”。

GetConfig

这个项目摘录了当前的结构编号、结构的个数、和多重结构操作数的个数。得到的字符串的格式如下：

“当前结构编号，结构个数，多重结构操作数个数”。

也可参见项目 SetConfig。

GetDate

这个项目摘录了当前的日期和时间，这以用户在 ZEMAX 环境对话框中选择的格式表示。

GetExtra

这个项目摘录了特殊表面数据。对应于得到的数据的表面被添加到这个项目的名称的后面。例如，为了得到表面 8 的特殊数据，这个项目名应该是“GetExtra，8”。得到的字符串的格式如下：

“特殊数据 1，特殊数据 2，特殊数据 3，……”

这个项目得到对应的表面编号作为一个证实数据。其它项目是自我说明的。

GetField

这个项目摘录了视场数据。语法结构为

GetField，n

这里 n 为零或者视场编号。如果 n 为零，则得到的字符串有以下格式：

“类型，数量”

类型参数是一个整数，为 0、1、或者 2 中的一个，分别代表以度表示的角度、物高、或者近轴近轴象高。数量参数是当前定义的视场的个数。如果 n 不为零，而是对应于一个有效的视场编号，得到的字符串有以下格式：

“x 视场，y 视场，权重，vdx，vdy，vcx，vcy”

这 7 个值是表示不同的视场数据的指数格式的浮点数。也可参见 SetField。

GetFile

这个项目摘录了镜头的完整名称，包括驱动器名和路径名。如果要修改这个文件，则要十分小心，因为在任意时刻，ZEMAX 都可能

要从这个文件读取数据或者写数据到这个文件中。

GetFirst

这个项目摘录了关于镜头的第一次序数据。得到的数据的格式如下：

“焦距，pwfn，rwfn，pima，pmag”

在这个字符串中的值分别是有效焦距、近轴工作 F/#、实际工作 F/#、近轴象高、和近轴放大率。

GetGlass

这个项目摘录了与任意一个表面使用的玻璃有关的一些数据。语法结构为：

GetGlass，表面

如果指定的表面是无效的，不是由玻璃构成的，或者是梯度折射率表面，则得到的字符串是空的。这个数据对于那些在 FdC 波带外定义的玻璃来说可能是无意义的，得到的字符串的格式如下：

“名称，nd，vd，dpgf”

GetGlobalMatrix

这个项目得到要求的矩阵，将任意当前坐标系（如光线追迹）转化成空间坐标系。关于空间坐标矩阵的详细内容可参见在系统菜单一章中的“空间坐标参考表面”部分。

这个项目的语法结构为：

GetGlobalMatrix，表面

得到的数据字符串有如下格式：

“R11，R12，R13，R21，R22，R23，R31，R32，R33，X_O，Y_O，Z_O”

GetIndex

这个项目摘录了任意一个表面的折射率数据。语法结构为：

GetIndex，表面

如果指定的表面是无效的，或者是梯度折射率表面，则得到的字符串是空的。否则，得到的字符串的格式如下：

“ n1，n2，n3，…… ”

这里的折射率数值按顺序对应于每个定义波长的折射率。

GetLabel

这个项目重新得到与指定表面有关的整数标记。语法结构为：

GetLabel，表面

得到的数值是表面的标记。也可参见 SetLabel 和 FindLabel。

GetName

这个项目摘录了镜头的名称。得到的字符串是当前镜头的名称，与在通用数据对话框中输入的一样。

GetPath

这个项目摘录了指向 ZEMAX 被安装于其中的目录的完整路径，和指向镜头的默认目录的路径。得到的字符串由一个逗号分开。

GetPolState

这个项目摘录了由用户设置的默认偏振状态。这个数据的格式如下：

“ nIsPolarized，Ex，Ey，Phax，Phay ”

如果 nIsPolarized 是不为零的任意值，则默认偏振状态为无偏振。否则，使用 Ex、Ey、Phax、和 Phay 来定义偏振状态。Ex 和 Ey 应该各自被归一化成一个单位数量，尽管这不被要求。Phax 和 Phay 以度来表示。可以参见 SetPolState。

GetPupil

这个项目摘录了光瞳数据。得到的字符串的格式如下：

“ 类型，数值，ENPD，ENPP，EXPD，EXPP，变迹类型，变迹因子 ”

类型参数是一个象征系统孔径类型的整数，是 0 和 5 之间的一个数，

分别代表入瞳直径、象空间 F/#、物空间 NA、通过光栏尺寸浮动、近轴工作 F/#、或者物方锥形角。数值参数是系统的孔径数值，除非使用的是通过光栏尺寸浮动类型，在这种情况中，这个数值是光栏表面的半口径。接下来的 4 个数值是入瞳直径、入瞳位置、出瞳直径、和出瞳位置，这些都是以镜头长度单位表示。变迹类型是一个整数，它被设为 0 来代表均匀，1 代表高斯，2 代表正切。变迹因子是在通用数据对话框中显示的数值。

GetPolTrace

这个项目与 GetTrace 非常相似，它有一个追迹通过系统的偏振光线的附加功能。详细内容可参见 GetTrace 的资料。项目 GetPolTrace 的语法结构如下：

GetPolTrace，波长，模式，表面，hx，hy，px，py，Ex，Ey，Phax，Phay

自变量和 GetTrace 是一样的，除了那些附加的自变量 Ex、Ey、Phax、和 Phay。Ex 和 Ey 是在 x 和 y 方向上的归一化的电场幅度。等式 Ex*Ex+Ey*Ey 应该有一个为 1 的值（有一个重要的例外，这在下面说明），尽管任意一个值都可以接受。Phax 和 Phay 是相对相位，以角表示。

如果 Ex、Ey、Phax、和 Phay 都为零，而且仅在这种情况中，ZEMAX 假设一条“无偏振”的光线是需要的。一条无偏振光线的追迹实际上要求 ZEMAX 追迹两条正交的光线，对得到的传递强度求平均值。如果四个数值中的任意一个不为零，则要追迹一条偏振光线。

例如，为了在波长编号 2 的波长处追迹一条实际的无偏振边缘光线到象面，这个项目字符串为：

GetPolTrace，2，0，-1，0.0，0.0，0.0，1.0，0，0，0，0

对于偏振光线，数据信息将以下面的格式返回：

“error，intensity，Exr，Eyr，Ezr，Exi，Eyi，Ezi”

如果成功地追迹了光线，则整数 error 将为零，否则它将是一个正数或负数。如果为正数，则光线避过有 error 说明地表面。如果为

负数，则光线在由数值 error 的绝对值给出的表面上发生全反射（TIR）。在使用字符串的余下部分之前，要始终检查以确保光线数据是有效的。

intensity 是传递强度。它总被归一化成一个单位的输入电场强度。这个传递强度计算了表面、薄膜、和总体吸收效果，但是不考虑有无光线被拦住。Ex、Ey、和 Ez 是电场分量，带着的字母 r 和 i 代表实部和虚部。

对于无偏振光线，得到的数据信息很简单，为：

“ error，intensity ”

虽然 GetPolTrace 很容易用来编程和使用，但是使用 GetPolTrace 有一个重要的缺点：每次 DDE 调用，只能追迹一条光线。通过 DDE 来传输数据的操作与光线追迹相比较是很多的，因此如果要求追迹大量的光线，则执行过程可能会相应的减慢。关于追迹大量光线的信息可参见在这一章的其它地方的“ 大量光线的追迹 ”部分的说明。也可参见 GetPolTraceDirect。

GetPolTraceDirect

GetPolTraceDirect 为由 GetTraceDirect 所做的 ZEMAX 光线追迹引擎提供了同样直接的方法，同样也允许追迹偏振光线。关于 GetTrace 直接方法版本的重要细节可参见 GetTraceDirect。项目 GetPolTraceDirect 的语法结构为：

GetPolTraceDirect，波长，模式，开始表面，停止表面，x，y，z，l，m，n，Ex，Ey，Phax，Phay

参数 Ex、Ey、Phax、和 Phay 与在 GetPolTrace 中定义的一样。返回的数据信息的格式与在 GetPolTrace 中说明的一样。

GetRefresh

这个数据项目使 ZEMAX 将 LDE 中的镜头数据复制到服务程序中储存的副本中。然后更新这个镜头，这意味着 ZEMAX 将重新计算所有的光瞳位置、求解、和折射率数据。如果这个镜头可以被更新，则 ZEMAX 返回一个字符串“ 0 ”，否则返回“ -1 ”。如果 GetUpdate 返

回“ -1 ”，则不执行任何光线追迹。

所有后面的命令都将影响新复制的镜头数据或者在它的基础上被执行。旧的镜头数据不能被恢复，即使这被要求。也可参见 GetUpdate 和 PushLens。

GetSag

这个项目计算任意一个表面的矢高。这个项目名的格式为“ GetSag，表面，x，y ”，这里表面是表面编号，x 和 y 是表面上计算矢高处的坐标。得到的字符串的格式为“ 矢高，改变的矢高 ”。X、y 和矢高的值都是以镜头长度单位表示的。

GetSequence

这个项目得到在服务程序存储器中的镜头的序号，然后是在 LDE 中的镜头的序号，这两个有一个逗号分开。

GetSerial

这个项目得到 ZEMAX 密码锁的序列号。

GetSolve

这个项目得到关于任一表面上的任一求解的数据。它的语法结构为“ GetSolve，表面，代码 ”，这里代码是一个整数代码，指出了这个求解数据是用在哪个表面参数上的。得到的求解数据以下列格式表示，这与代码值有关。

GetSolve 代码	得到的数据的格式
0，曲率	求解类型，参数 1，参数 2
1，厚度	求解类型，参数 1，参数 2，参数 3
2，玻璃	求解类型，拾取表面编号
3，半口径	求解类型，拾取表面编号
4，圆锥系数	求解类型，拾取表面编号
5-12，参数 1-8	求解类型，拾取表面编号，补偿，比例因子

求解类型是一个整数代码，这些参数有着依赖于求解类型的意义；详细内容可参见“求解”一章。也可参见 SetSolve。

GetSurface

这个项目摘录了表面数据。要返回数据的表面编号被附加在项目名称后面，例如，为了得到表面 8 的数据，这个项目的名称应该为“GetSurface，8”。

得到的字符串的格式如下：

“类型，曲率，厚度，玻璃，半口径，圆锥系数，p1，p2，p3，p4，p5，p6，p7，p8”

这个项目得到一个 8 个字母的字符串的表面类型。注意，下一个数据项是曲率，而不是半径。其它项都是自我解释的。

也可参见 SetSurface。

GetSystem

这个项目摘录了系统数据。

得到的字符串的格式如下：

“表面数量，单位代码，光栏表面编号，非轴对称标记，光线定位类型，环境数据，温度，压力，空间参考表面”。

这个项目得到了表面的数量，单位代码（0、1、2、或 3 分别代表 mm、cm、in、或 M），光栏表面编号，用来指明系统是否是非轴对称系统（0 代表假，说明它是一个轴对称系统；或者如果系统是非轴对称的则为 1）的非轴对称标记，光线定位类型（0、1、2 分别代表无、近轴光线、实际光线），使用的环境数据标记（0 代表不是，1 代表是），当前温度和压力，以及空间坐标的参考表面编号。

也可参见 SetSystem 和 GetSystemAper。

GetSystemAper

这个项目摘录了系统的光圈数据。

得到的字符串的格式如下：

“ 类型，光栏表面编号，光圈值 ”

这个项目得到了系统的光圈类型（0、1、2、3、4、或 5 分别代表入瞳直径、象空间 F/#、物空间数值孔径、通过光栏尺寸浮动、近轴工作 F/#、或物方锥形角），光栏表面的编号，和系统的光圈值。如果光圈类型为通过光栏尺寸浮动，则光圈值为光栏表面的半口径。

也可参见 GetSystem 和 SetSystemAper。

GetTol

这个项目摘录了公差数据。它的语法结构为：

GetTol，n

这里 n 为 0 或者其它公差操作数编号。如果 n 为 0，得到的字符串的格式如下：

“ number ”

这里 number 为定义的公差操作数的数量。如果 n 不为 0，而是对应于一个有效的操作数的编号，则得到的字符串的格式如下：

“ 公差类型，int1，int2，min，max ”

GetTrace

这个项目要求客户程序提供附加数据。为了追迹一条光线，ZEMAX 需要知道相应的视场和光瞳坐标、波长、模式（实际，模式=0；或者近轴，模式=1），以及要追迹光线到其上的表面。所有这些数据都被编码，附加在项目名称的后面。这通过定义由逗号分开的不同参数来实现，如下：

GetTrace，波长，模式，表面编号，hx，hy，px，py

例如，要追迹在波长 3 处的实际主光线到表面 5，这个项目字符串为

GetTrace，3，0，5，0.0，1.0，0.0，0.0

虽然这看起来麻烦，但是它易于编程，是将数据送进 ZEMAX 的最简单的方法。ZEMAX 接受这个项目，辨别在这个名称开头的“ GetTrace ”，记下这个字符串的其余部分，然后分析它，以及追迹

光线。通常，仅仅需要在象面上的光线数据；通过将表面编号设置为-1，可产生在象面上的光线数据。注意，0 被用来代表物面。

返回的数据信息有如下格式：

“ error，vigcode，x，y，z，l，m，n，l2，m2，n2，intensity ”

如果光线追迹成功，则整数 error 为零，否则它将是一个正数或者负数。如果是正数，则光线将避过由 error 指出的表面，如果是负数，则光线在由数值 error 的绝对值给出的表面上发生全反射（TIR）。在使用这个字符串的余下部分之前，始终要检查以确保光线数据是有效的。

参数 vigcode 是指光线被拦光的第一个表面。除非在那个面上发生错误或者后来才到那个面，否则这条光线将继续被追迹通过被要求的表面。

x、y、和 z 的值是指在被要求的表面上坐标。

l、m、和 n 的值是指折射进入跟在被要求的表面以后的介质中以后光线的方向余弦。

l2、m2、和 n2 的值是指在被要求的表面上的表面截止点的法线的方向余弦。

intensity 是光线的相对强度，包括定义的任一光瞳或者表面的照度分布。

虽然 GetTrace 易于编程和使用，但是使用 GetTrace 有一个重要的缺点：每次 DDE 调用只能追迹一条光线。通过 DDE 传输数据的操作与光线追迹相比要多得多，所以如果要求追迹大量的光线，则执行将相应减慢。关于大量光线的追迹的信息，可参见这一章中其它地方的“ 大量光线的追迹 ”部分的说明。也可参见 GetTraceDirect。

GetTraceDirect

GetTraceDirect 为 ZEMAX 的光线追迹引擎提供了一个比较直接的方法。通常，光线是由归一化的视场和光瞳坐标 hx、hy、px、和 py。ZEMAX 获得这些归一化坐标，计算物体坐标（x，y，和 z）和

到入瞳目标点（l，m，和 n）的方向余弦（l，m，和 n，分别代表 x、y、和 z 方向余弦）。

然而，有时直接规定 x、y、z、l、m、和 n 更适合于追迹光线。直接规定对于光学系统中的任意地方的光线有定义起始表面方面的额外的适合性。这个适合性表现在一个特殊的客户程序上，这个程序用来严格确保光线起始坐标是有效的。

与 GetTrace 一样，这个项目要求客户程序提供附加数据。为了追迹光线，ZEMAX 必须知道 x、y、z、l、m、n、波长、模式（实际光线，模式=0；或者近轴光线，模式=1）以及光线追迹的起始和终止表面。所有这些数据都要被编码，附加在项目名称的后面。这是通过定义由逗号隔开的不同参数来实现的，如下：

GetTraceDirect，波长，模式，起始表面，终止表面，x，y，z，l，m，n

返回的数据信息的有如下格式：

“ error，vigcode，x，y，z，l，m，n，l2，m2，n2，intensity ”

这里的参数和 GetTrace 的参数是完全一样的，除了 intensity 之外。Intensity 是光线的相对传递强度，但排除任何定义的光瞳照度分布在外。注意，GetTrace 包括光瞳照度分布，而 GetTraceDirect 没有。但两者都包含了表面照度分布。

虽然 GetTraceDirect 易于编程和使用，但是使用 GetTraceDirect 有一个重要的缺点：每次 DDE 调用只能追迹一条光线。通过 DDE 传输数据的操作与光线追迹相比要多得多，所以如果要求追迹大量的光线，则执行将相应减慢。关于大量光线的追迹的信息，可参见这一章中其它地方的‘ 大量光线的追迹 ”部分的说明。也可参见 GetTrace。

GetUDOSystem

这个项目被用从优化程序存储器中载入一个特定镜头，送到 ZEMAX 的服务程序存储器中。使用这个项目名的唯一一次机会是在执行一个用户自定义操作数或者 UDO 时。UDO 在“ 优化 ”一章已详细说明了。其详细内容可参见那一章。GetUDOSystem 的语法结

构为：

GetUDOSystem，缓冲器代码

这里的缓冲器代码是一个在命令行中的被传送到 UDO 中的整数。也参见 SetUDOData。

GetUpdate

这个数据项目使 ZEMAX 去更新一个镜头，这意味着 ZEMAX 将重新所有的光瞳位置、求解、和折射率数据。如果镜头可以被更新，则 ZEMAX 返回字符串“ 0 ”，否则，它将返回字符串“ -1 ”。如果 GetUpdate 得到“ -1 ”，则将不执行任何光线追迹。可以参见 GetRefresh。

GetVersion

这个项目得到 ZEMAX 的版本号。

GetWave

这个项目摘录了波长数据。其语法结构为：

GetWave，n

这里 n 为零或者波长编号。如果 n 为零，则得到的字符串的格式如下：

“ 主波长，数量 ”

参数主波长是一个整数，它指出了哪个波长是主波长。参数数量是指当前定义的波长的数量。如果 n 不为零，而是对应于一个有效的波长编号，则得到的字符串的格式如下：

“ 波长，权重 ”

这两个数值都是指数格式的浮点数，它们对应于指定波长的数值和权重。也可参见 SetWave。

InsertSurface

这个项目插入了一个新的表面。其语法结构为：

InsertSurface，表面

新的表面将被放在由参数表面指定的位置上。也可参见

SetSurface 关于新表面的数据定义的内容，以及项目 DeleteSurface。

LoadFile

将一个 ZEMAX 文件载到服务程序中。注意，文件的载入不会改变在 LDE 中显示的数据；服务程序有这个镜头数据的一个独立副本。要载入的文件名被附加在 LoadFile 的项目名称之后，而且必须包括完整的路径名。例如："LoadFile，C:\ZEMAX\SAMPLES\COOKE.ZMX"。在更新这个新载入的镜头文件之后，这个项目得到的字符串和项目 GetUpdate 的是相同的。如果得到一个不为零的数值，则更新失败；如果得到-999，则这个文件不能被载入。也可参见 GetPath、SaveFile、和 PushLens。

LoadMerit

载入一个 ZEMAX 的.MF 或者.ZMX 文件，记录它的评价函数，并将它放到在服务程序载入的镜头中。注意，评价函数文件的载入不会改变在 LDE 中显示的数据；服务程序有这个镜头数据的一个独立副本。要载入的文件名被附加在 LoadMerit 的项目名称之后，而且必须包括完整的路径名。例如："LoadMerit，C:\ZEMAX\SAMPLES\MyMerit.MF"。得到的字符串的格式如下：

"数量，评价"

这里参数数量是在评价函数中的操作数的数量，参数评价是评价函数的数值。如果这个评价函数的数值为 9.00e+009，则这个评价函数不能被评估。

也可参见 Optimize。

MakeGraphicWindow

这个项目通知 ZEMAX 图形数据已被写到一个文件中了，现在可以以一个 ZEMAX 子窗口的形式来显示。这个项目的主要目的是在一个客户应用程序中执行一个用户自定义特性，这看起来和实际操作 ZEMAX 本来的特性一样。这个项目的字符串必须有以下格式：

MakeGraphicWindow，文件名，模块名，窗口标题，文本标记，设置数据

参数“文件名”是指向一个临时文件的完整路径名和文件名，这个临时文件中包含了这些图形数据。无论如何，这必须与在命令行自变量中的被传送给可执行客户程序的文件有相同的名称。参数“模块名”是产生图形数据的客户程序的完整路径名和执行名称。参数“窗口标题”是一个字符串，它定义了 ZEMAX 用来放在这个图形窗口的顶端条栏中的标题。

如果客户程序同时也可以产生这些数据的一个文本版本，则参数“文本标记”应为 1。由于当前数据是一个图形显示（如果这个项目是 MakeGraphicWindow，则它必须是），所以 ZEMAX 想知道“Text”菜单选项对用户是否有效，或者它是否被变灰。如果参数“文本标记”为零，则 ZEMAX 将使这个“Text”菜单选项变灰，将不会要求这个客户程序去产生这个数据的一个文本版本。

参数“设置数据”是一个由空格（不是逗号）分开的数值字符串，它被客户程序用来定义如何产生图形数据。这些数值仅由客户程序使用，但不能被 ZEMAX 使用。字符串“设置数据”包含了通常在一个 ZEMAX 的“设置”类型对话框中显示的选项和数据。如果需要的话，设置数据可以被用来重新产生这些数据。关于设置数据的详细内容可参见“ZEMAX 如何调用客户程序”的说明。

一个范例字符串如下：

MakeGraphicWindow , C:\TEMP\ZGF001.TMP , C:\ZEMAX\FEATURES\CLIENT.EXE, ClientWindow, 1, 0 1 2 12.55

这个项目表明 ZEMAX 应该打开一个图形窗口，显示储存在文件 C:\TEMP\ZGF001.TMP 中的数据，通过调用客户程序模块 C:\ZEMAX\FEATURES\CLIENT.EXE 来做一些更新或者设置改变。这个客户程序可以产生这个图形的一个文本版本，设置数据字符串（仅被客户程序使用）为“0 1 2 12.55”。

MakeTextWindow

这个项目通知 ZEMAX 文本数据已被写到一个文件中了，现在可以以一个 ZEMAX 子窗口的形式来显示。这个项目的主要目的是在一个客户应用程序中执行一个用户自定义特性，这看起来和实际操作

ZEMAX 本来的特性一样。这个项目的字符串必须有以下格式：

MakeTextWindow，文件名，模块名，窗口标题，设置数据

参数“ 文件名 ”是指向一个临时文件的完整路径名和文件名，这个临时文件中包含了这些文本数据。无论如何，这必须与在命令行自变量中的被传送给可执行客户程序的文件有相同的名称。参数“ 模块名 ”是产生文本数据的客户程序的完整路径名和执行名称。参数“ 窗口标题 ”是一个字符串，它定义了 ZEMAX 用来放在这个文本窗口的顶端条栏中的标题。

参数“ 设置数据 ”是一个由空格（不是逗号）分开的数值字符串，它被客户程序用来定义如何产生文本数据。这些数值仅由客户程序使用，但不能被 ZEMAX 使用。字符串“ 设置数据 ”包含了通常在一个 ZEMAX 的“ 设置 ”类型对话框中显示的选项和数据。如果需要的话，设置数据可以被用来重新产生这些数据。关于设置数据的详细内容可参见“ ZEMAX 如何调用客户程序 ”的说明。

一个范例字符串如下：

MakeTextWindow，C:\TEMP\ZGF002.TMP，C:\ZEMAX\FEATURES\CLIENT.EXE，ClientWindow，6 5 4 12.55

这个项目表明 ZEMAX 应该打开一个文本窗口，显示储存在文件 C:\TEMP\ZGF002.TMP 中的数据，通过调用客户程序模块 C:\ZEMAX\FEATURES\CLIENT.EXE 来做一些更新或者设置改变。设置数据字符串（仅被客户程序使用）为“ 6 5 4 12.55 ”。

NewLens

这个项目将清除当前的镜头。留下的“ 最小 ”镜头与在镜头数据编辑界面中选择“ 文件，新文件 ”时的镜头是相同的。不会给出保存这个已存在镜头的提示。

Optimize

Optimize 将调用 ZEMAX 阻尼最小二乘法优化。其语法结构为：

Optimize，n

这里 n 时运行的循环数。得到的数值是最终的评价函数。如果得到的评价函数值为 9.0E+009，优化失败，通常这是因为镜头或者评价函数不能被评估。如果 n 为零，则以自动模式运行优化。如果 n 为小于零的值，则 Optimize 得到当前的评价函数值，而不执行优化。

PushLens

PushLens 将获得当前在服务程序存储器中的镜头，并将它放到镜头数据编辑界面中。ZEMAX 主窗口将显示一个对话框，问用户是否允许接受由客户程序放入的镜头数据。如果这个在 LDE 中的镜头数据没有被保存，则将出现一个附加对话框，询问旧的镜头数据是否先被保存。在更新这个新放入的镜头文件之后，得到的字符串是项目 GetUpdate 得到的是一样的。如果得到一个不为零的数值，则更新失败；如果得到-999，则这个镜头不能被放入到 LDE 中。也可参见 GetPath、GetRefresh、LoadFile、和 SaveFile。

ReleaseWindow

当 ZEMAX 调用客户程序去更新或者改变由客户程序函数使用的设置时，在窗口中的菜单条变成灰色，防止同时要求多次更新或者改变设置。通常，在客户程序编码发送了 MakeTextWindow 和 MakeGraphicWindow 时，这个菜单选项才被再次激活。然而，如果在一个更新或者设置改变的过程中，不能计算新的数据，则必须释放这个窗口。ReleaseWindow 正好提供了这一个简单的目的。如果在改变设置时用户选择了“取消”，则客户程序编码将发送一个 ReleaseWindow 项目来释放菜单条的缩定。如果没有发送这个命令，则不能关闭这个窗口，这通常可以防止终止 ZEMAX。ReleaseWindow 项目仅仅使用一个自变量：临时文件的名称。其语法结构为：

ReleaseWindow，文件名称

如果没有窗口在使用这个文件名称，则得到的数值为零；如果这个文件被使用则得到的数值为一个正整数。

SaveFile

将当前在服务程序中的镜头保存为一个 ZEMAX 文件。被用来保

存的文件的名称被附加在 SaveFile 项目的名称的后面，而且必须包括这个文件的完整的路径名。例如：“ SaveFile，C:\ZEMAX\SAMPLES\COOKE.ZMX ”。在更新这个新被保存的镜头文件之后，得到的字符串与项目 GetUpdate 得到的是一样的。如果得到一个不为零的数值，则更新失败；如果得到-999，则不能保存这个文件。也可参见 GetPath、GetRefresh、LoadFile、和 PushLens。

SetBuffer

项目 SetBuffer 被用来储存窗口被创建或者更新的客户程序特定数据。缓冲器数据被用来存放用户选择的选项，而不是在 MakeTextWindow 和 MakeGraphicWindow 项目的命令行中使用设置数据。这个数据必须以一个字符串格式表示。其语法结构为：

SetBuffer，1，你想要的任意文本……

这里提供了 16 个缓冲器，从编号 0 到 15，每个都可以被设成使用 SetBuffer，0，……；SetBuffer，1，……；等等。跟在“ SetBuffer，n，”后面的文本仅仅是那些被存放的文本；它最大可以有 240 字符。

注意，直到送出 MakeTextWindow 或者 MakeGraphicWindow，否则缓冲器数据与任意特定的窗口无关。一旦 ZEMAX 接受到 MakeTextWindow 或者 MakeGraphicWindow 项目，则这个缓冲器数据将被复制到适当的窗口存储器中，然后可能通过使用 GetBuffer 从那个窗口重新得到这个数据。也可参见 GetBuffer。

SetCofig

这个项目转换当前的结构编号和更新系统。想要的结构被附加在这个项目名称的后面。例如，为了转换到结构 3，这个项目的名称为“ SetCofig，3 ”。

得到的字符串的格式如下：

“ 当前结构编号，结构数量，error ”

当前结构编号是新的结构编号，在 1 和结构数量值之间。通常，它将是在项目名称中要求的目标结构，只要它是一个有效的结构编号。Error 代码与由项目 GetUpdate 得到的是一样的，如果新的当前结构

是可被追迹的，则这个代码为零。也可参见 GetConfig。

SetField

这个项目设置了视场数据。其语法结构为：

SetField，0，类型，数量

或者

SetField，n，xf，yf，wgt，vdx，vdy，vcx，vcy

如果 n 的值为零，则视场类型和总的视场数量被设为一个新的整数值。如果 n 为一个有效的视场编号（在 1 和视场数量之间，包括这两个数），则视场 x 和 y 的值，视场权重，和渐晕因子全都被设置。也可参见 GetField。得到的数值与“ GetField，n ”得到的相同。

SetFloat

这个项目将所有没有表面口径的表面设为有一个浮动口径。浮动口径将挡住在半口径外面追迹的光线。其语法结构为：

SetFloat

得到的数值仅仅是“ OK ”。

SetLabel

这个项目与指定表面的一个整数标记有关。当在这个目标表面周围增加或删除表面时，ZEMAX 将保留这个标记。其语法结构为：

SetLabel，表面，标记

得到的数值是这个标记。也可参见 GetLabel 和 FindLabel。

SetPolState

这个项目设置了由用户设置的偏振状态。这个数据的格式如下：

SetPolState，nIsPolarized，Ex，Ey，Phax，Phay

可关于完整的说明可参见 GetPolState。

SetSolve

这个项目设置了求解数据，得到由于任意表面的新的求解的数

据。其语法结构为：

SetSolve，表面，代码，数据……

这里，代码的定义和“ Getsolve ”中说明的是一样，“ 数据…… ”是在 GetSove 中定义的同样数据。这些参数根据列出的代码顺序列出。得到的字符串与对新定义的求解数据用 GetSolve 得到的是一样的。

SetSurface

这个项目设置了表面数据。这个项目名称的格式如下：

SetSurface，表面，类型，曲率，厚度，玻璃，半口径，圆锥系数，p1，p2，p3，p4，p5，p6，p7，p8

这里的这些参数与在项目 GetSurface 中说明的是一样的。得到的字符串是新设置的表面数据，其格式与在项目 GetSurface 中说明的一样。也可参见 GetSurface。

SetSystem

这个项目设置了系统数据。其语法结构为：

SetSystem，单位代码，光栏表面，光线定位类型，环境数据，温度，压力，空间参考表面

得到的字符串的格式与在 GetSystem 中定义的是一样的。关于自变量的详细说明也可参见 GetSystem。

SetSystemAper

这个项目设置了系统的光圈数据。其语法结构为：

SetSystemAper，类型，光栏表面，光圈值

得到的字符串的格式与在 GetSystemAper 中定义的是一样的。关于自变量的详细说明也可参见 GetSystemAper。

SetUDOData

这个项目被用来由客户程序将计算好的数据传输给 ZEMAX 的优化程序。使用这个项目名称的唯一一次机会是执行一个用户自定义操作数，或者 UDO 时。UDO 在“ 优化 ”一章中已详细说明。详细内容可参见那一章。SetUDOData 的语法结构为：

SetUDOData，缓冲器代码，数据 0，数据 1，数据 2，……，数据 50

这里，缓冲器代码是在命令行中的传送给 UDO 的一个整数。当 ZEMAX 服务程序接受到这个 SetUDOData 时，它将这些数据放在缓冲器中，然后关闭缓冲器，不让其它数据输入。这个项目允许一次传送多个数据项给服务程序，然而，在输入行中有一个最大不超过 255 字符的限制。为了避免这个（Windows 的）限制，可使用项目 SetUDOItem 和 CloseUDOData。也可参见 GetUDOSystem。

SetUDOItem

这个项目被用来由客户程序仅仅将一个计算好的资料传输给 ZEMAX 的优化程序。使用这个项目名称的唯一一次机会是执行一个用户自定义操作数，或者 UDO 时。UDO 在“优化”一章中已详细说明。详细内容可参见那一章。SetUDOItem 的语法结构为：

SetUDOItem，缓冲器代码，数据编号，数据

这里，缓冲器代码是在命令行中的传送给 UDO 的一个整数，数据是被传输的数据项的编号。使用这个项目，服务器缓冲器可能一次就被一个数据项充满，这个数据项已远远超出关于数据项内容的 255 个字符的限制。然而，ZEMAX 不知道是否有更多的数据被送过来，所以缓冲器保持打开状态。在最后一个数据项被传送以后，在执行优化之前必须使用 CloseUDOData 关闭缓冲器。一个典型的定义包含下列项目信息：

SetUDOItem，缓冲器代码，0，数值 0

SetUDOItem，缓冲器代码，1，数值 1

SetUDOItem，缓冲器代码，2，数值 2

CloseUDOData，缓冲器代码

也可参见 GetUDOSystem。

SetWave

这个项目设置了波长数据。其语法结构为：

SetWave，0，主波长，数量

或者

SetWave，n，波长，权重

如果 n 的值为零，则主波长编号和总的波长数量被设成一个新的整数值。如果 n 是一个有效的波长编号（在 1 和波长数量之间，包括这两个数），则以微米表示的波长和波长的权重都被设置。也可参见 GetWave。得到的数据与“ GetWave，n ”得到的一样。

大量光线的追迹

如果只有少量的光线需要被追迹，则使用 DDE 项目是足够容易和快的，如 GetTrace 或 GetPolTrace。然而，如果需要追迹多于 100~500 左右条的光线，则使用数组光线追迹技巧要快得多。

与一次追迹一条光线不同，这建立了一个数组，它容纳了一个所有要被追迹的光线的列表，然后一次性将整个数组传送给 ZEMAX。然后 ZEMAX 将追迹所有光线，并将这整个数组传回给客户程序。

这个方法模仿了 GetTrace、GetTraceDirect、GetPolTrace、和 GetPolTraceDirect 的运转,但这个方法有一个附加功能，它可以一次性追迹任意多的光线，虽然它编写起来稍微有点复杂。

下面 2 个步骤是要求的：

1） 用要求的数据充满一个数组的结构

2） 将这个数据传输给 ZEMAX

步骤 1：将光线数据放入数组

将定义了要被追迹的光线的列表的数据放到一个数组结构中，其形式如下：

```
typedef struct
  {
  double x,y,z,l,m,n,opd,intensity;
```

```
int wave,error,vigcode,want_opd;
}DDERAYDATA;
```

在客户程序内部，将用一个说明来建立一个数组类型DDERAYDATA，这个说明如：

```
DDERAYDATA MyData[101];
```

这里支持4中光线追迹模式，从0到3编号，它们对应于GetTrace的操作（模式 0）、GetTraceDirect 的操作（模式 1）、GetPolTrace的操作（模式 2）、和 GetPolTraceDirect（模式 3）。每种模式要求在这个 DDERAYDATA 结构中放入一列稍微不同的数据。这个模式是由在数组位置 0 处的参数 opd 参数来设置的。

每条光线需要 1 个数组分量。如果要追迹“ n ”条光线，则从 1 到 n 的数组结构将被用来存放这 n 条光线的数据。数组位置零被保留给那些需要和数组一起被传送的标题数据。

模式 0：类似于 GetTrace

与 GetTrace 一样，光线是由相对的视场和光瞳坐标、波长、模式（实际光线模式=0，或者近轴光线模式=1）、以及要追迹光线到其上的表面来定义。在追迹光线之前，应修改数组中位置 1 到 n 的内容，使它们包含了每条光线的定义。视场、光瞳、和波长数据全都如下被存放在数组中：

```
MyData[i].x = hx;
MyData[i].y = hy;
MyData[i].z = px;
MyData[i].l = py;
MyData[i].m = 0.0;  /*对于输入忽略*/
MyData[i].n = 0.0;  /*对于输入忽略*/
MyData[i].opd= 0.0;  /*对于输入忽略*/
```

MyData[i].intensity = 1.0; /*初始强度*/

MyData[i].wave = 波长;

MyData[i].error = 0; /*最初必须为零*/

MyData[i].vigcode = 0; /*最初必须为零*/

MyData[i].want_opd = 0; /*如果不需要 OPD 数据则为零，如需要则为 1*/

注意，自变量 i 应该在 1 和光线数量 n 之间。数组的大小是随意的，然而，一次追迹太多的光线对于编程实践来说是差的，这是因为如果光线追迹被终止，则要接受一个用户中断是困难的。初始强度可以根据发生任意光瞳或表面照度分布来缩放。注意，模式 0 承认光瞳照度分布，而模式 1 却不承认。

数组位置 0 被留给那些应用于所有光线的其它数据。我们用在数组位置 0 中的数据来告诉 ZEMAX 使用的模式、光线的数量、和要追迹光线到其上的表面。这些数据如下被放入数组中：

MyData[i].x = 0.0; /*忽略*/

MyData[i].y = 0.0; /*忽略*/

MyData[i].z = 0.0; /*忽略*/

MyData[i].l = 0.0; /*忽略*/

MyData[i].m = 0.0; /*忽略*/

MyData[i].n = 0.0; /*忽略*/

MyData[i].opd= 0; /*设置为模式 0*/

MyData[i].intensity = 0.0; /*忽略*/

MyData[i].wave = 模式; /*0 代表实际光线，1 代表近轴光线*/

MyData[i].error = 光线数量; /*数组中光线的数量*/

MyData[i].vigcode = 0; /*最初必须为零*/

MyData[i].want_opd = 最后表面编号; /*用-1 代表象面，或者用其它任意的有效表面编号*/

光线的数量必须放在变量 error 中。注意，一个 DDERAYDATA 类型的数组的说明必须比光线的数量大 1，这是因为数组位置 0 要被留给上面说明的数据。所有在这个数组中的光线必须使用相同的模式被追迹到相同的表面上，但是它们可以有不同的波长以及视场和光瞳坐标。

如果变量 want_opd 为 0，则计算这条光线的通常的 x、y、z、l、m、和 n。如果 want_opd 为一不为零的数，则和这些通常数据一起将得到这条光线 OPD。

计算这条光线的 OPD 将花费正规的光线追迹之外的额外时间，而且仅仅在被要求如此做时，ZEMAX 才执行这个额外计算。对于客户程序计算 OPD 也比较复杂，这是因为 OPD 意味着光程差，这意味着必须追迹两条光线，而不是一条光线。为了计算一条任意光线的 OPD，ZEMAX 必须先追迹主光线，再追迹这条任意光线，然后将两条光线的相位相减，得到 OPD。这与反复地追迹这条相同的主光线（这是慢的）不同，通常只追迹一次主光线，然后从后面计算的每条光线中减去主光线的相位。ZEMAX 使用参数 want_opd 的符号来指定应该使用哪个计算。如果 want_opd 小于零（如-1），则主光线和指定光线都要求被追迹，OPD 是两者之间的相位差；如果 want_opd 大于零，则得到这条光线的“自然”光程，客户程序必须将被条光线都独立减去主光线的相位。如果从相同视场点出发追迹许多光线，则后面的方法要快得多，许多光学分析计算都是这种情况。

如果最后一个表面是象面，则可以计算 opd，否则，opd 的值将为零。

模式 1：类似于 GetTraceDirect

与 GetTraceDirect 一样，光线是由在任意起始面上的坐标 x、y、z、l、m、和 n，以及波长、模式（实际光线模式=0，或者近轴光线模式=1）、和要追迹光线到其上的表面来定义。在追迹光线之前，应修改数组中位置 1 到 n 的内容，使它们包含了每条光线的定义。视场、

光瞳、和波长数据全都被如下存放在数组中：

```
MyData[i].x = x;
MyData[i].y = y;
MyData[i].z = z;
MyData[i].l = l;
MyData[i].m = m;
MyData[i].n = n;
MyData[i].opd= 0.0;  /*对于输入忽略*/
MyData[i].intensity = 1.0;  /*初始强度*/
MyData[i].wave = 波长;
MyData[i].error = 0.0;  /*最初必须为零*/
MyData[i].vigcode = 0.0;  /*最初必须为零*/
MyData[i].want_opd = 0;  /*忽略*/
```

注意，自变量 i 是在 1 和光线的数量 n 之间。数组的大小是随意的，然而，一次追迹太多的光线对于编程实践来说是差的，这是因为如果光线追迹被终止，则要接受一个用户中断是困难的。初始强度可以根据发生任意光瞳或表面照度分布来缩放。注意，模式 0 承认光瞳照度分布，而模式 1 却不承认。

数组位置 0 被留给那些应用于所有光线的其它数据。我们用在数组位置 0 中的数据来告诉 ZEMAX 使用的模式、光线的数量、和要追迹光线到其上的表面。这些数据如下被放入数组中：

```
MyData[i].x = 0.0;  /*忽略*/
MyData[i].y = 0.0;  /*忽略*/
MyData[i].z = 0.0;  /*忽略*/
MyData[i].l = 0.0;  /*忽略*/
```

MyData[i].m = 0.0; /*忽略*/

MyData[i].n = 0.0; /*忽略*/

MyData[i].opd= 1; /*设置为模式 1*/

MyData[i].intensity = 0.0; /*忽略*/

MyData[i].wave = 模式; /*0 代表实际光线，1 代表近轴光线*/

MyData[i].error = 光线数量; /*数组中光线的数量*/

MyData[i].vigcode = 0; /*最初必须为零*/

MyData[i].want_opd = 最后表面编号; /*用-1 代表象面，或者用其它任意的有效表面编号*/

这种模式在其它方面是与模式 0 是一样的。但是在这种模式中不允许 OPD 的计算。

模式 2：类似于 GetPolTrace

这种模式与 GetPolTrace 非常相似。在追迹光线之前，应修改数组中位置 1 到 n 的内容，使它们包含了每条光线的定义。视场、光瞳、和波长数据全都被如下存放在数组中：

MyData[i].x = hx;

MyData[i].y = hy;

MyData[i].z = px;

MyData[i].l = py;

MyData[i].m = 0.0; /*忽略*/

MyData[i].n = 0.0; /*忽略*/

MyData[i].opd= 0.0; /*对于输入忽略*/

MyData[i].intensity = 1.0; /*初始强度*/

MyData[i].wave = 波长;

MyData[i].error = 0.0; /*最初必须为零*/

MyData[i].vigcode = 0.0; /*最初必须为零*/

MyData[i].want_opd = 0; /*忽略*/

注意，自变量 i 是在 1 和光线的数量 n 之间。数组的大小是随意的，然而，一次追迹太多的光线对于编程实践来说是差的，这是因为如果光线追迹被终止，则要接受一个用户中断是困难的。

参数 intensity 被用来得到这条光线的相对传递强度。

数组位置 0 被留给那些应用于所有光线的其它数据。我们用在数组位置 0 中的数据来告诉 ZEMAX 使用的模式、光线的数量、和要追迹光线到其上的表面。这些数据如下被放入数组中：

MyData[i].x = Ex; /*x 方向上的电场振幅*/

MyData[i].y = Ey; /*y 方向上的电场振幅*/

MyData[i].z = Phax; /*Ex 的相位，以度表示*/

MyData[i].l = Phay; /*Ey 的相位，以度表示*/

MyData[i].m = 0.0; /*忽略*/

MyData[i].n = 0.0; /*忽略*/

MyData[i].opd= 2; /*设置为模式 2*/

MyData[i].intensity = 0.0; /*忽略*/

MyData[i].wave = 模式; /*0 代表实际光线，1 代表近轴光线*/

MyData[i].error = 光线数量; /*数组中光线的数量*/

MyData[i].vigcode = 0; /*最初必须为零*/

MyData[i].want_opd = 最后表面编号; /*用-1 代表象面，或者用其它任意的有效表面编号*/

这种模式在其它方面是与模式 0 是一样的。

模式 3：类似于 GetPolTraceDirect

与 GetTracePolDirect 一样，光线是由在任意起始面上的坐标 x、y、z、l、m、和 n，以及波长、模式（实际光线模式=0，或者近轴光线模式=1）、和要追迹光线到其上的表面来定义。在追迹光线之前，应修改数组中位置 1 到 n 的内容，使它们包含了每条光线的定义。视场、光瞳、和波长数据全都被如下存放在数组中：

```
MyData[i].x = x;
MyData[i].y = y;
MyData[i].z = z;
MyData[i].l = l;
MyData[i].m = m;
MyData[i].n = n;
MyData[i].opd= 0.0;  /*忽略*/
MyData[i].intensity = 1.0;  /*初始强度*/
MyData[i].wave = 波长;
MyData[i].error = 0.0;  /*最初必须为零*/
MyData[i].vigcode = 0.0;  /*最初必须为零*/
MyData[i].want_opd = 0;  /*忽略*/
```

注意，自变量 i 是在 1 和光线的数量 n 之间。数组的大小是随意的，然而，一次追迹太多的光线对于编程实践来说是差的，这是因为如果光线追迹被终止，则要接受一个用户中断是困难的。

参数 intensity 被用来得到这条光线的相对传递强度。

数组位置 0 被留给那些应用于所有光线的其它数据。我们用在数组位置 0 中的数据来告诉 ZEMAX 使用的模式、光线的数量、和要追迹光线到其上的表面。这些数据如下被放入数组中：

```
MyData[i].x = Ex;  /*x 方向上的电场振幅*/
MyData[i].y = Ey;  /*y 方向上的电场振幅*/
```

MyData[i].z = Phax;　/*Ex 的相位，以度表示*/

MyData[i].l = Phay;　/*Ey 的相位，以度表示*/

MyData[i].m = 0.0;　/*忽略*/

MyData[i].n = 0.0;　/*忽略*/

MyData[i].opd= 3;　/*设置为模式 3*/

MyData[i].intensity = 0.0;　/*忽略*/

MyData[i].wave = 模式;　/*0 代表实际光线，1 代表近轴光线*/

MyData[i].error = 光线数量;　/*数组中光线的数量*/

MyData[i].vigcode = 起始面;　/*坐标开始的表面*/

MyData[i].want_opd = 最后表面编号;　/*用-1 代表象面，或者用其它任意的有效表面编号*/

这种模式在其它方面是与模式 1 是一样的。

步骤 2：传输数组给 ZEMAX

一旦所有的光线都被定义完成，数组必须被传输给 ZEMAX 服务程序。实现这一目标的简单的方法是使用 ZCLIENT PosArrayTraceMessage 函数：

PosArrayTraceMessage(szBuffer,RD);

变量 RD 是指向包含光线列表的 DDERAYDATA 数组的指针。ZEMAX 将追迹所有的光线，然后将这些数据传回到相同的数组中。有关这个代码例子的详细内容，可参见本章结尾的“一个客户程序例子”部分。

ZEMAX 如何调用客户程序

ZEMAX 在目录\Extend 中查找扩展执行程序。在这个目录中以普通的.EXE 扩展名结尾的每个程序都被认为是一个有效的 ZEMAX 扩展程序。这个可执行程序的名称被放在主菜单栏中的“扩展”下的下拉菜单中。

当从这个菜单中选中任意一个被列出的扩展程序名称时，ZEMAX 将一个以下的语法形式来执行这个扩展程序：

programname textflag optionsflag tempfile {settings data}

参数 programname 是这个可执行程序的完整路径名，例如，“ C:\ZEMAX\EXTENSIONS\ DDE_DEMO.EXE ”。

参数 textflag 初始总为 0。ZEMAX 对于扩展程序使用的约定和对于 ZEMAX 内部特性是一样的：如果这个特性同时产生一个图形输出格式和一个文本输出格式，则先显示这个图形版本。如果这个特性没有图形显示，则显示文本窗口。扩展程序必须产生一个图形文件，如果它可以而且被要求这样做的话。用户可以在新窗口中点击“ 文本 ”来查看文本版本；如果这种情况发生，则 ZEMAX 将把文本标记设为 1，再次调用这个扩展程序。如果扩展程序只产生文本，则它可以产生一个文本文件，通过传送项目 MakeTextWindow 来打开它。除非一个扩展程序特别规定它可以支持一个文本模式，否则 ZEMAX 将永远不会传送一个被设为 1 的 textflag；这就是为什么 MakeGraphicWindow 要返回一个标记来说明这个特性是否支持文本的原因，以及为什么 ZEMAX 总是要先寻找图形模式的原因。

如果用户不想看到 “ 设置 ” 框，则将 optionsflag 设为 0，或者，如果用户想看的话则设为 1。设置这个标记可用两种方法；用户在已打开的图形窗口中点击 “ 设置 ”，或者，用户从 ZEMAX 内部的环境选项中选择 “ 先显示选项 ”。在后面这种情况中，设置框可以要求在第一次计算之前被显示。如果设置了这个标记，则扩展必须使用一个对话框或者其它界面装置来提示用户设置选项。如果没有用户自定义选项，则将显示一个声明 “ 这个窗口没有选项 ” 的信息框。

参数 temfile 是包含由扩展产生的数据的临时文件的完整路径名和文件名。最初，这个文件不存在，需要扩展程序来建立它，将文本或图形数据写到这个文件中，然后按要求调用 MakeTextWindow 或 MakeGraphicWindow。如果用户只是简单地更新一个已存在的窗口，那么这个文件可能已经存在了，它将被覆盖。这个文件不能被打开或者读取，因为它的数据不可能是扩展程序最初放在那里的数据了。

ZEMAX 特别做了一些建立图形文件的扩展的翻译。没有其它文件可被用作输出，因为 ZEMAX 认为要被更新的窗口要保留它们的原始文件名。只有一个窗口可以通过一个到扩展程序的简单的调用来建立。

当 ZEMAX 从扩展程序中接受到一个 MakeTextWindow 或 MakeGraphicWindow 请求时，在基本单元名称中的被编码的自变量之一是一个空格，这个空格用来分开定义数值的扩展程序字符串。这个字符串应被用来包含代表扩展程序需要的设置的“设置数据”，例如表面编号、视场、波长、或者在窗口中要被计算的其它数据。当 ZEMAX 更新一个窗口时，它将调用这个扩展程序，而且提供这个设置字符串回到扩展程序。

重要的一点是，扩展不会自己试着将这个设置数据保持或储存在一个文件中。其原因是 ZEMAX 可能有同时打开的多个文本或图形窗口，它们全都使用相同的扩展编码来产生显示数据。例如，一个计算和显示单个表面的权重的扩展程序可以同时被几个窗口同时使用，每个窗口代表不同的表面。但是 ZEMAX“知道”哪个窗口使用哪个设置数据；因此 ZEMAX 将在相应的命令行中传输这个数据。在最初的扩展程序调用中，不提供设置数据字符串，所以扩展程序必须测定它自己的默认设置。

文本窗口的生成

由 ZEMAX 生成一个用来显示的文本窗口是十分简单的。首先，使用典型的 C 语言 fopen 函数打开要传输到扩展程序的文件名，带着的“wt”代表“写文本”模式。然后，输出数据到这个文件中，关闭文件，然后传送项目 MakeTextWindow 到 ZEMAX。ZEMAX 将读取这个 ASCII 码数据，显示出来，不作任何修改或解释。一个简单的测试编码如下：

```
FILE *out;
out = fopen(szFile, "wt");
fputs( "Write this line\n",out);
fclose(out);
```

不要忘记关闭文件！

图形窗口的生成

扩展程序通过将图形格式数据写到在命令行中提供的临时文件中来产生一个图形窗口。使用与由 ZPL 宏指令语言或者注释特性使用的相似的命令语言来建立一个图形数据。

所有的图形都被写在一个虚拟屏幕上，它的坐标系为宽 1.00 乘以高 1.00，原点（0，0）在左下角。屏幕的外形比例通常是宽 4 乘以高 3，或者宽 5 乘以高 3，这依赖于用户的偏爱（详细内容可参见“ 文件菜单 ”一章中的环境部分）。因此，被画出的虚拟的象素的宽度大于高度。

在这个命令文件中的每一行中应该跟上一个换行符“ \n ”。在将所有的图形命令都写到这个文件中之后，这个文件应该被关闭，禁止写入。

ZEMAX 承认下面的图形命令承认。

ADDRESS

语法：ADDRESS

这个 ADDRESS 命令将导致在图框的右下角显示 “ 地址框 ”，除非用户已选择了这个框不显示。

BOX

语法：BOX x1 y1 x2 y2

这个 BOX 命令连接了一个矩形的四个点，画出四条线。这两个坐标是这个矩形的左上点和右下点的坐标。

DATA

语法：DATA n “ string ”

这个 DATA 命令接受一个在 0 和 5 之间的整数自变量和一个文本字符串。这个整数自变量指出了应该将这个字符串写在这个数据框的哪一行。通常，镜头名称在第 0 行，日期在第 1 行。命令 LENSNAME 和 DATE 将自动实现这个目的。DATA 命令允许将任一字符串放在任

意一行上。

DATE

语法：DATE

当前的日期和时间通常被写在这个数据框中的图表标题的下面的第二行。这个命令没有自变量，将在这第二行上打印日期和时间（根据用户的环境参考）。

FRAME

语法：FRAME

无论是 FRAME 命令，还是 NOFRAME 命令，都必须放在这个命令文件的第一行。FRAME 将画出一个标准的 ZEMAX 图形，整体周围有一个矩形，在 y 方向的 0.2 左右处有两条划线。可以使用 TITLE 命令将这个图表的标题写在这两条刻度线的中心。也可参见 NOFRAME。

LENSNAME

语法：LENSNAME

当前镜头的名称通常被写在这个数据框中的图表标题下面的第一行。这个命令没有自变量，将在这一行打印镜头的名称。

LINE

语法：LINE x1 y1 x2 y2

这是一个最基本的图形命令；它连接两个点，画出一条直线。注意，这个坐标应该在 0 和 1.0 之间。

NOFRAME

语法：NOFRAME

无论是 FRAME 命令，还是 NOFRAME 命令，都必须放在这个命令文件的第一行。FRAME 将画出一个标准的 ZEMAX 图形，但没有两条划线。也可参见 FRAME。

PENSTYLE

语法：PENSTYLE ncolor nstyle nwidth

这个 PENSTYLE 命令改变了线条的颜色、类型、和宽度。整数 ncolor 必须是 0，代表黑色，或者 1 和 12 之间的数，代表相应的线条颜色。这个颜色可以由用户在文件，环境对话框中定义。整数 nstyle 可以是 0，代表实线，或者 1 和 4 之间的数，代表相应的虚线。整数 nwidth 实线条的相应宽度。它的默认值为 1。一个为 2 的数值将线宽为默认宽度的两倍。如果线条类型不是实线（nstyle=0），则线宽必须是 1 个象素；Windows 不支持不是实线的粗线。所有后面的文本和画的线条将应用这个新的颜色、类型、和宽度。

TEXT

语法：TEXT “ string ” x y angle width height

这个 TEXT 命令被用来在图形中放置文本数据。坐标 x 和 y 实字符串的起始坐标。角度 angle 是以度为单位的。宽度 width 和高度 height 的值影响了字体的尺寸。其默认值为 10，表示每个命令将产生与 ZEMAX 通常使用的一样的字体尺寸；这是屏幕宽度的 1/70，是屏幕高度的 1/40。大于或小于 10 的数值将产生按比例缩放的字体尺寸。

TITLE

语法：TITLE “ string ”

如果用 FRAME 来定义图形，而不是 MOFRAME，则图表标题通常将被放在离图表底部 20%的两条划线的中间。TITLE 命令将在这个位置显示提供的字符串 “ string ”。

一个扩展程序的例子

在\Extend 目录中的是一个被称为 DDE_DEMO 的应用程序例子，它举例说明了连接 ZEMAX 的 DDE 的使用。它包括了源代码。DDE_DEMO 可以在两种模式之一中运行：独立程序，或者作为一个扩展程序。

如果从一个命令提示、Windows 资源管理器、或者通过在 DDE_DEMO 图标双击来直接运行，则它将被作为一个独立应用程序，操纵在后台的 DDE。它产生的数据将在 DDE_DEMO 窗口中显示。为了以独立模式运行 DDE_DEMO，则先要运行 ZEMAX，载入

一个镜头文件，然后通过在 DDE_DEMO 图标上双击来运行 DDE_DEMO。试着载入不同的镜头文件，或者编辑一个镜头文件，观察这个 DDE_DEMO 窗口来监视它的变化。DDE_DEMO 将在屏幕上列出一些系统数据、表面数据、视场数据、和波长数据，追迹少量光线，列出这些光线的 x、y、和 z 坐标。这个程序每隔几秒钟就自我更新一次。

为了以一个扩展程序模式来运行 DDE_DEMO，则运行 ZEMAX，载入一个镜头文件，然后选择扩展程序，再选择 DDE_DEMO。注意，放在\Extend 目录中的任意一个.EXE 应用程序都将在 ZEMAX 的扩展菜单中列出，无论它是一个有效的独立程序，还是一个能与 ZEMAX 沟通的扩展程序。DDE_DEMO 将在一个“隐藏”窗口中执行，建立与 ZEMAX 连接的 DDE 链，摘录系统和光线数据，将这些数据写到一个文件中，通知 ZEMAX 这个文件已经准备好了（使用 MakeTextWindow 项目），然后关闭。一旦客户程序产生了这些数据，则它的工作就结束了；它将在那时自我终止。

当 ZEMAX 接受到这个 MakeTextWindow 项目，它将读取这个文件的内容，并将在一个标准的 ZEMAX 文本窗口中显示这些数据。如果这个窗口现在被“更新”，或者选择了“设置”选项，那么 ZEMAX 将再次调用这个客户程序，请求再次计算这些数据。

注意，完全一样的程序可以作为一个独立的客户程序，也可作为一个扩展程序。这种双模式运行的诀窍是，如果 ZEMAX 调用了这个客户程序，则它将传输命令行自变量给客户程序，告诉它要建立那种类型的数据以及将这些数据写到哪里；如果没有提供命令行自变量，则客户程序认为它是从操作系统中被调用的，因此它将以独立模式运行。DDE 交换和后来的处理是同样的方法，仅仅需要少量代码行来确定窗口是可见的还是隐藏的，数据是被写到一个文件中还是客户程序自己的屏幕上。

DDE_DEMO 代码

这里提供了 DDE_DEMO 源代码。它包含了所有客户程序的基本运算法则、变量、说明、必需的 Windows 信息处理。也有一部分代

码举例说明了追迹光线数组的 WM_DDE_POKE 的使用。

DDE_DEMO 作为一个扩展程序的起始点来使用是一个非常困难的程序，虽然它说明了一些高水平的技巧。一个执行 ZEMAX 扩展的简单方法说明如下。

执行扩展程序的简单化技巧

显然，为 Windows 设计的 DDE 不是为那些胆怯者设计的；它包括了大量的程序设计技巧和对信息循环、DDE、指针、基本单元、和全局处理的融会贯通。

然而，被用来与 ZEMAX 沟通的大部分代码是标准的“ 锅炉钢板 ”代码，这对于任意一个客户程序来说都是普通的。这个代码已被全部编写完毕，放在一个被称为 ZCLIENT 的单一源代码模块程序中。ZCLIENT 以 ZEMAX 的源代码形式被提供，可以自由复制（只要保留著作权），以及在新的扩展程序中使用。

ZCLIENT 程序

ZCLIENT 显然是处理所有与 ZEMAX 的信息传送。在 ZCLIENT 内部嵌入是一个对被称为“ 用户函数 ”名称的单一函数的调用。这个函数被放在一个由用户提供的单独的 C 语言程序中，和 ZCLIENT 一起被编辑，来建立这个可执行扩展程序。

当 ZEMAX 调用这个扩展程序时，开始先在 ZCLIENT 中执行。ZCLIENT 建立了 DDE 传输，然后调用用户函数。在用户函数中，由 ZCLIENT 提供的两个函数，PostRequestMessage 和 GetString，都是通常被要求用来从 ZEMAX 中得到需要的数据的函数。然后这些数据作为一个文本数据或图形数据形式被格式化，再传回 ZEMAX，显示出来。

PostRequestMessage 的语法结构如下：

PostRequestMessage(szItemname,szBuffer);

szItemname 包含了要求的项目的名称。可用的项目名称在这一章的前面部分已被说明。然后，将要求的数据传送回字符串 szBuffer 中。

ZEMAX 通常提供逗号来分开数据，以及函数 GetString 来摘录这些单独的数据项。GetString 的语法结构如下：

GetString(szBuffer,nItem,szSubString);

字符串 szBuffer 是由 PostRequestMessage 得到的缓冲器。整数 nItem 是要求的数据项的编号；0 代表第一个数据项，1 代表第二个数据项，等等。符号 szSubString 保存了从 nItem 位置得到的字符串。

例如，为了得到一个镜头的名称，其代码应该是：

PostRequestMessage(“ getname ”,szBuffer);

GetString(szBuffer,0,szLensName);

现在，字符串 szLensName 中包含了这个镜头的名称。因为“ GetName ”仅仅得到 1 个数据项，所以这个代码可以被缩短成只有一行：

PostRequestMessage(“ getname ”, szLensName);

这里也有 PostRequestMessage 快速追迹大量光线的一个修改版本。其函数为 PostArrayTraceMessage。这个函数接受两个自变量：缓冲器字符串，和光线数据数组的地址。这里提供了使用 PostArrayTraceMessage 的数组光线追迹代码的例子，被称为 ARR_DEMO.C。

为了使用 PostArrayTraceMessage 来追迹光线，必须先定义光线列表，然后必须制作一个专门的 PostArrayTraceMessage 调用。这是这个技巧的一个例子：

/*以标题数据填充 RD 的数组位置 0。*/

RD[0].x = 0.0;

RD[0].y = 0.0;

RD[0].z = 0.0;

RD[0].l = 0.0;

```
RD[0].m =0.0;
RD[0].n = 0.0;
RD[0].opd = 0.0; /* 这是我们设置模式的地方，模式 0 与 GetTrace 一样 */
RD[0].intensity = 0.0;
RD[0].wave = 0;
RD[0].error = 25; /* 追迹 25 条光线 */
RD[0].vigcode = 0;
RD[0].want_opd = -1;
/*定义 25 条光线。当然，你可以定义你想要的任意光线……*/
k = 0;
for(i=-2;I<=2;I++)
    {
     for(j=-2;j<=2;j++)
         {
          k++;
          RD[k].x = 0.0;
          RD[k].y = 0.0;
          RD[k].z = (double) j/4.0;
          RD[k].l = (double) j/4.0;
          RD[k].m = 0.0;
          RD[k].n = 0.0;
          RD[k].opd = 0.0;
          RD[k].intensity = 0.0;
```

```
        RD[k].wave = 1;
        RD[k].error = 0;
        RD[k].vigcode = 0;
        RD[k].want_opd = 0;
        }
    }
/*现在可去得到数据*/
PostArrayTraceMessage(szBuffer,RD);
/*输出数据现在被存放在 RD 中，准备使用！*/
/*注意，仅仅用一行代码来追迹所有的光线 */
```

使用 ZCLIENT 比自己编写所有的 DDE 传输代码要容易得多。这个被追迹数组例子可从 ARR_DEMO 中得到，这个一个使用 ZCLIENT 得 DDE 代码例子。

一个使用 ZCLIENT 得扩展程序的例子

这里提供了一个使用 ZCLIENT 的完整的扩展程序的源代码。这个扩展程序被称为 PHASPLOT，它将使用用 ZCLIENT.C 和 PHASPLOT.C。PHASPLOT 将产生了一个二元光学面 2 的相位的一个图形或者一个文本列表。这个代码举例说明了如何制作一个图形和文本窗口，如何格式化输出图形和文本显示。

PHASPLOT 也有一个 ZEMAX 类型的“设置”框，对于这个“设置”框的源代码被包括在“SurDlgProc”函数中的 PHASPLOT.C 中；它使用 PHASPLOT.RC 来定义这个对话框的界面。

由 PhasPlot 模板编写扩展程序

学习如何去编写扩展程序的最好的方法是从 PhasPlot 开始，以及开始编辑！制作 PhasPlot.C 和 PhasPlot.RC 的一个副本，将副本重新命名成其它名字（MY_CODE.C，MY_CODE.RC）。然后使用任

意一个商业性好的、有效的 C 语言编辑器（Borland 和 MicroSoft 都制造了优秀的工具）来建立一个新的方案，将这 3 个文件载入这个方案。重新编辑，将这个方案建成一个测试 EXE，来确保所有都是适当的。

然后，在用户函数和设置对话框函数内部编辑代码，使之适合你的要求。