# TWAIN学习报告

## TWAIN协议剖析

撰写人：Leo Yang
日　期：2009-11-20

# 1. TWAIN简介

- 1.1 TWAIN存在的必要性
- 1.2 使用TWAIN接口的好处
  - 1.2.1对应用程序开发者的好处
  - 1.2.2对硬件开发商的好处
  - 1.2.3对终端客户的好处

# 1.1 TWAIN存在的必要性

- 在 TWAIN 诞生以前 ,软件和硬件开发商都是定义自己的图像采集设备的接口.这样 ,对于软件开发商来说 ,就要开发出每种设备的驱动 ,对于硬件开发商来说就要为每个应用程序开发不同的驱动.这种作法显然是低效 ,而且是不现实的.于是 ,就需要制定一个开放的工业标准 ,让它作为图象采集设备和应用程序之间的桥梁 ,允许应用程序透明地使用图象采集设备. TWAIN 就是在这种背景下 ,由 TWAIN Working Group 制定的.它提供了一个应用程序和图象采集设备之间的公共接口.

# 1.2.1对应用程序开发者的好处

- 提供给终端用户一个简单的方法，在不离开你的应用程序的情况下，访问任何兼容性的光栅设备。
- 节省时间和金钱。假如你提供了低级别的扫描仪驱动，你不再需要支持和发行这些驱动。TWAIN的兼容设备将提供源程序模组来实现相同的功能。
- 允许你的程序访问任何TWAIN的兼容性设备的数据，包括扫描仪，数码相机，图像数据库等
- 允许你根据应用程序的需求，来决断具体使用哪些图像获取设备的功能和特性
- 您不需要提供控制图像获取过程的用户界面。这个界面由每个兼容TWAIN的源设备DS来提供。当然，你也可以提供你自己的用户界面。

# 1.2.2对硬件开发商的好处

- 提升了你的产品的价值和支持
- 允许为你的设备提供专有的用户界面，这样你可以为你的用户呈现设备最新的特性
- 无需整合各种应用，只需创建一个单一的TWAIN兼容的Source

# 1.2.3对终端客户的好处

- 提供给客户一个简单的方法，在不需要离开应用程序的情况下，用极少的步骤，直接把图像导入他们的文档中。

# 2. TWAIN 基本原理

- 2.1 TWAIN 协议的软件体系结构
- 2.2 TWAIN 接口通讯
  - 2.2.1 DSM_Entry()
  - 2.2.2 DS_Entry()
- 2.3 TWAIN 的工作流程
- 2.4 TWAIN 的传输方式

# 2.1 TWAIN 协议的软件体系结构

- TWAIN 提供了应用程序和支持 TWAIN 的设备之间连接的方法.应用程序与设备之间交互的模型是一个 4 层结构 ,这 4 层分别是:应用层(Application Layer) 、 协议层( Protocol Layer) 、 采集层(Acquisition Layer)和设备层(Device Layer) .与这 4 层相对应的组件分别是:应用程序(Application) 、 源管理器(Source Manager) 、 源(Source)和物理设备(Physical Hardware Device) ,如图 1 所示.

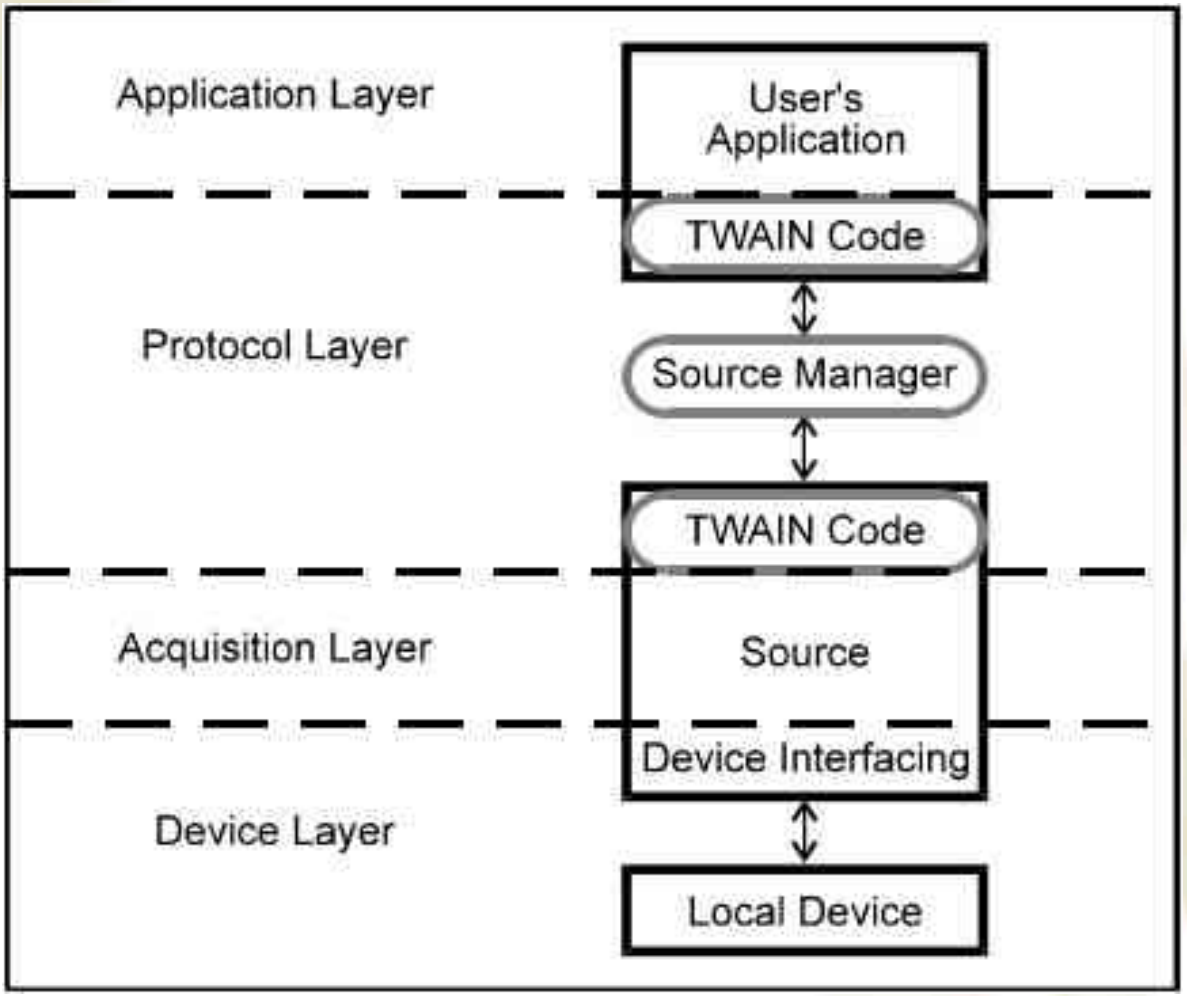
- 应用层(Application Layer) :用户的应用程序运行的所在层.

- 协议层(Protocol Layer) :该层包括 3 个部分. (1) 提供应用程序和 TWAIN 之间接口的程序; (2)源管理器; (3)源管理器和源之间交互的程序.其中 ,源管理器(SM ,Source Manager)是整个体系的核心. SM 的主要任务是建立并管理应用程序和源之间的连接 ,它允许用户选择需要的源 ,载入、 卸载选中的源 ,并确信所有来自应用程序的请求都被正确的传送到正确的源

- 采集层(Acquisition Layer) :该层由源(Source)组成.源是 TWAIN 体系结构中的硬件接口 ,它的作用是利用源和应用程序都认知的格式以及传输机制, 从硬件设备中获得数据, 并把数据传送到应用程序.

- 设备层 (Device Layer) : 物理设备的所在层.

# 图1：TWAIN软件体系结构图



# 2.2 TWAIN 接口通讯

- TWAIN软件系统由3部分协同工作， 完成图像的请求、 扫描、传输。TWAIN 的3 个组成部分之间可以并且仅可以通过两个接口(函数)进行通信， 它们是 DSM_Entry()， DS_Entry()，如图 2 所示。

- 应用程序不能和数据源(Source)直接对话，所有的请求、 属性等信息必须经由数据源管理器(DSM)中介，才能获取和设置图像分辨率、色彩、亮度、对比度等性能参数。

- 数据源管理器(DSM)是应用程序(APP)和数据源(Source)之间通信的桥梁，提供界面让用户选择数据源、装载数据源，当 DSM_Entry()指明的目的地是自己时，则由自己处理，当目的地是数据源时，数据源管理器通过DS_Entry()调用，将有关参数传递给数据源，由数据源进行处理。

# 2.2.1 DSM_Entry()

- On Windows:

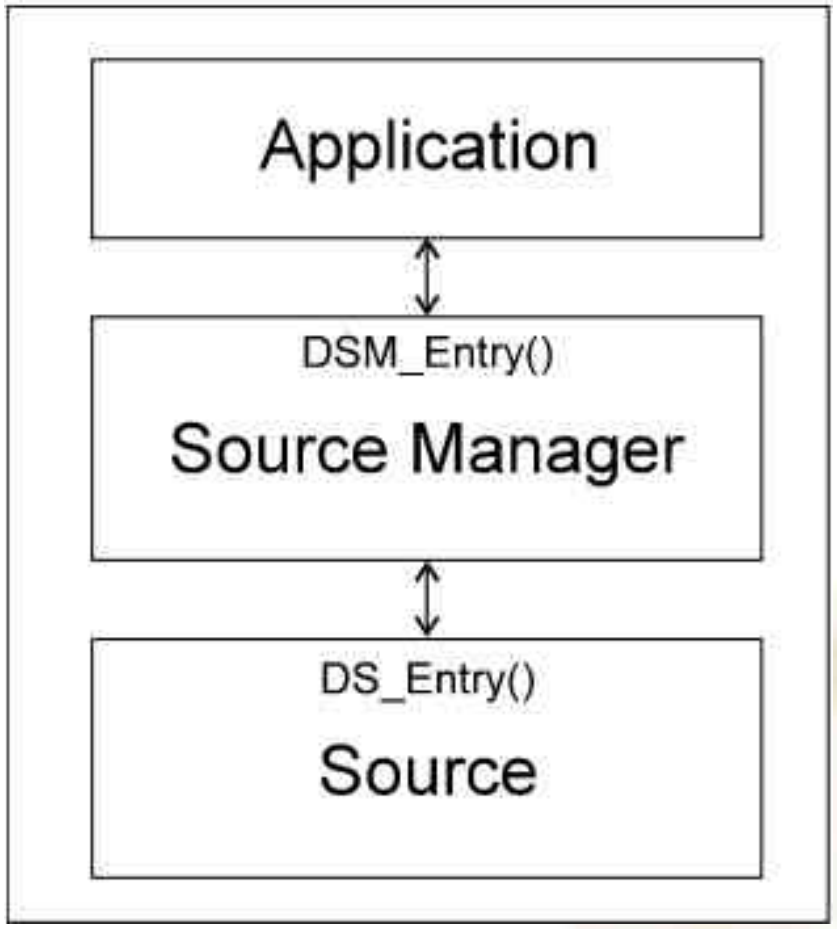
```
TW_UINT16 FAR PASCAL DSM_Entry
(   pTW_IDENTITY pOrigin, //消息的发出地
    pTW_IDENTITY pDest,//消息的目的地：SM或DS
    TW_UINT32 DG, //数据组标识(ID)， 格式为 DG-xxxx
    TW_UINT16 DAT, //数据变量类型， 格式为 DAT-xxxx
    TW_UINT16 MSG, //messageID， 格式为 MSG-xxxx
    TW_MEMREF pData //存放数据的缓存指针
);
```

# 2.2.2 DS_Entry()

- On Windows

```
TW_UINT16 FAR PASCAL DS_Entry
   (pTW_IDENTITY pOrigin, // source of message
   TW_UINT32 DG, // data group ID: DG_xxxx
   TW_UINT16 DAT, // data argument type:
   DAT_xxxx
   TW_UINT16 MSG, // message ID: MSG_xxxx
   TW_MEMREF pData // pointer to data
);
```

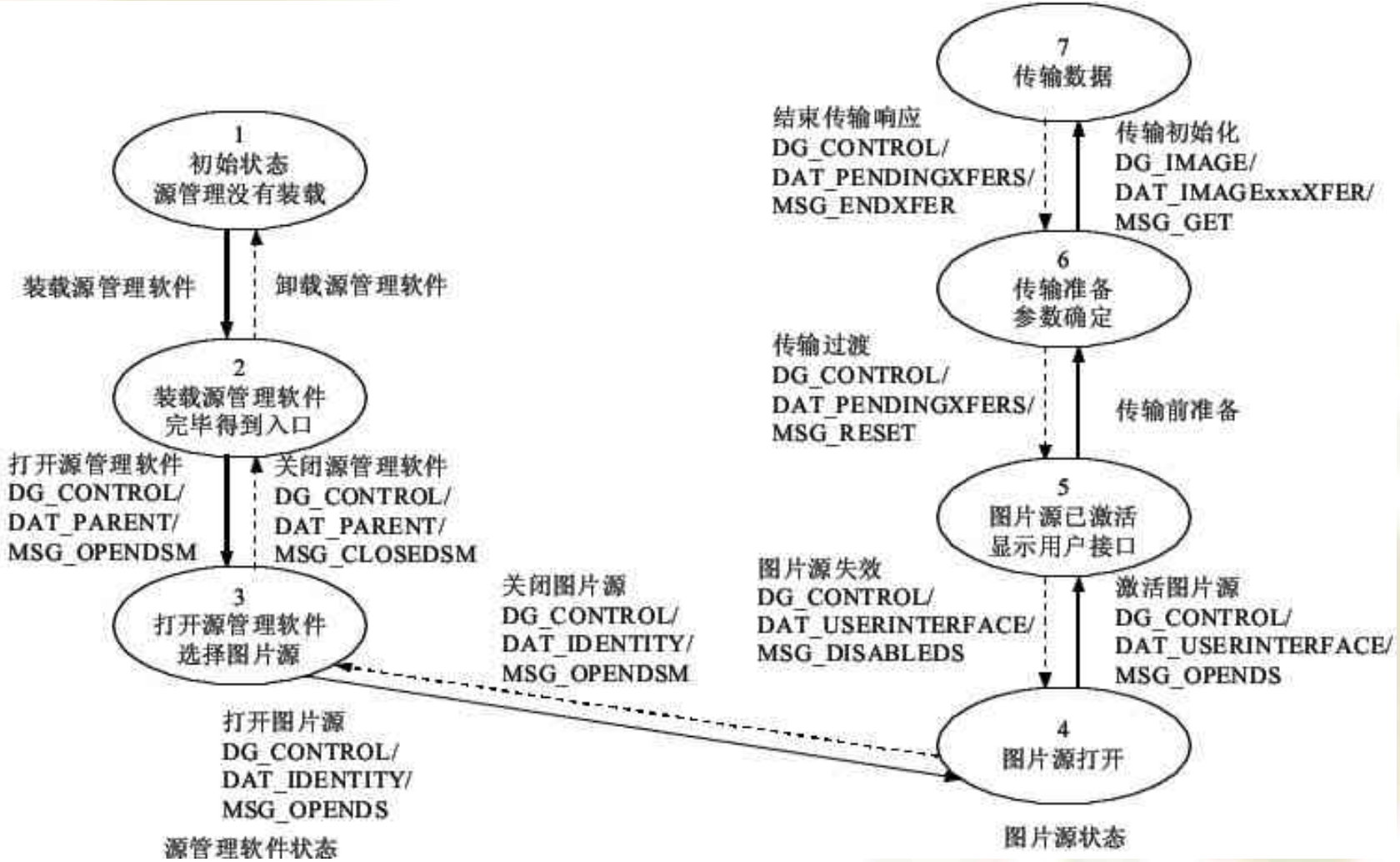
# 图2：TWAIN 接口通讯图



# 2.3 TWAIN 的工作流程

- 函数的所要执行的操作主要是通过DG、DAT、MSG 这 3个参数描述的。图像输入程序是 TWAIN 工作的具体实现，整个流程分为 8 个步骤如图 3 所示：先载入、 打开 SM ,让用户选择源;再打开选中源;获取、 传送图象;最后关闭源和 SM.

# 图3：TWAIN 的工作流程图



# TWAIN 的工作流程详解

- (1) 从状态1 到状态2。调入源管理软件，并取得DSM_Entry()的地址。
- (2)状态 2 至状态3。打开源管理软件，将DSM_Entry()的参数DG/DAT/MSG分别设置为DG_CONTROL/DAT_PARENT/MSG_OPENDSM，调用函数即可打开源管理软件。
- (3)状态 3。选择数据源。参数 DG/DAT/MSG 分别设置为DG_CONTROL/DAT_IDENTITY/MSG_USERSELECT。

- (4)状态 3 至 4。打开数据源。参数 DG/DAT/MSG 分别设置为 DG_CONTROL/DAT_IDENTITY/MSG_OPENS。
- (5)状态 4。获取和设置设备属性。参数 DG/DAT/MSG 分别设置为 DG_CONTROL/DAT_CAPABILITY/MSG_GET。
- (6)状态4至5。确定图片的属性。参数DG/DAT/MSG分别设置为 DG_CONTROL/DAT_USERINTERFACE/MSG_ENABLEDS。
- (7)状态 5 至 6。所需数据准备完成。这一步不需要发出操作而是等待数据源的准备就绪的消息。
- (8)状态 6 至状态 7。发动传输过程。这一步完成两个操作。 第一，DG/DAT/MSG分别设置为DG_IMAGE/DAT_IMAGE-INFO/MSG_GET。Porigin为前述的应用程序描述结构，Pdest为前述数据源的描述结构， Pdata指向TW_IMAGEINFO结构。这个操作将取回待传送的图片的信息。第二，DG/DAT/MSG分 别 设 置 为 DG_IMAGE/DAT_IMAGENATIVEXFER/MSG_GET。这一过程需要应用程序考虑的工作比较多，如果希望能够输出Windows可识别的BMP图像文件， 则需要构造BMP文件头， 然后和获取的图像数据一起存放在文件中。
- 按照上述 7 个状态 8个步骤完成图像输入的过程，只有这8个步骤前后协调有序的工作才能正确地进行数据的传输，而这每一步都需要 TWAIN 协议标准的支持。

# 2.4 TWAIN 的传输方式

- 从设备获取图象数据后 ,就需要将数据传送到应用程序. TWAIN 提供了 3 种文件传输的方式 — — — 本地方式(Native mode) 、 文件方式(Disk File Mode)和缓存方式(Buffered Memory mode) .

- 本地方式：源(source)分配一块内存 ,并把图象数据写入该内存 ,然后把指向这块内存的指针交给应用程序(Application) ,在传送完数据后 ,应用程序把这块内存释放.这种方式是任何源都支持的 ,也是最简单的一种方式.但是 ,这种方式只能支持非压缩的图片格式 ,而且也相当耗内存.

- 文件方式：应用程序创建一个文件 ,并确定该文件的格式和文件名 ,源按照应用程序规定的格式 ,把图象数据写入该文件.这种方式避免了本地方式需要大量内存的缺点 ,但是由于受到磁盘 I/ O 的限制 ,它的速度比较慢.这种方式一般用于大图片的扫描.

- 缓存方式：应用程序分配 1 个或者多个缓冲区 ,图象数据是作为无格式位图传送 ,应用程序通过传送中的信息( TW- IMAGEINFO 和 TW- IMAGEMEMXFER)来实现各个缓冲区之间的连接.最后 ,应用程序释放缓冲区.这种方式结合了本地方式和文件方式的优点 ,对内存资源多和内存资源少的机器都适用.

# 3、Application的TWAIN接口实现

- 3.1.1 安装Source Manager软件
- 3.1.2 从你的应用程序控制TWAIN Session
- 3.1.3 TWAIN兼容应用程序的开发步骤

# 3.1.1 安装Source Manager软件

- 1）在Windows目录寻找SM文件(TWAIN.DLL, TWAIN_32.DLL, TWUNK_16.EXE, and TWUNK_32.EXE)
- 2) 假如没有SM被安装，则安装和你的应用程序一起派发的SM程序
- 3) 假如存在则检测版本号，假如更新则替换并备份原文件。

# 3.1.2 从应用程序控制TWAIN 会话

- 装载 Source Manager并且获取 DSM_Entry (State 1 to 2)
- 打开 Source Manager (State 2 to 3)
- 选择 Source (during State 3)
- 打开 Source (State 3 to 4)
- 和Source协商Capabilities (during State 4)
- 请求从Source获取数据 (State 4 to 5)
- 识别数据传输是否准备就绪 (State 5 to 6)
- 开始传输数据 (State 6 to 7)
- 结束数据传输 (State 7 to 6 to 5)
- 断开 TWAIN Session (State 5 to 1 in sequence)

# 3.1.3 TWAIN兼容程序的开发步骤1

在 Windows下用 C/C+ +语言开发 TWAIN 应用软件 ,开发人员必须对应用程序作适当的修改。

- (1)修改资源文件:在资源文件的菜单部分中 ,增加两个菜单项(选择扫描仪和扫描图像) 。
- (2)修改应用程序的初始化部分:在创建应用程序(CreateWindow)时 ,可以同时对 TWAIN赋初值 ,并调用DCInitialize ()函数进行初始化。
- (3)修改应用程序的消息循环:首先调用 Process DCMessage ()函数处理有关 TWAIN消息。

```
while (GetMessage ( (LPMSG) &msg , NULL , 0 , 0) ) {
        //先处理有关 TWAIN的消息
        if ( ( ! gbDCDSOpen) ( ! ProcessDCMessage (
        (LPMSG) &msg , ghMainWnd) ) ) {
        TranslateMessage ( (LPMSG) &msg) ;
        DispatchMessage ( (LPMSG) &msg) ; }}
```

# 3.1.3 TWAIN兼容程序的开发步骤2

- (4)修改应用程序的菜单消息处理部分:在菜单中,当选中“选择扫描仪”时,调用 void IDMSourceProc
  (void) {
  if (DCOpenDSM() = TRUE) {
  DCSelectDS() ; DCCloseDSM() ; }}
  当选中“扫描图像”时,调用DCAcquire ()函数。
- (5)修改应用程序的窗口过程:在窗口过程中,增加 PMË XFERDONE消息的处理。
- void PMXferdoneProc (WPARAM wParam) {
  //从wParam参数中获得扫描图像的位图句柄 }
- (6)在应用程序的相应处增加 DCTerminate ( )函数,以关闭驱动资源,结束扫描过程。

# TWAIN Session 表解

| Application 所接收的 | State | Application 所做的动作 |
|---|---|---|
| 选择 Source... | 1 -> 2 | 装载 Source Manager |
| | 2 -> 3 | DG_CONTROL / DAT_PARENT / MSG_OPENDSM<br>DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT |
| | 3 -> 2 | DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM |
| | 2 -> 1 | 卸载 Source Manager |
| 获取... | 1 -> 2 | 装载 Source Manager |
| | 2 -> 3 | DG_CONTROL / DAT_PARENT / MSG_OPENDSM |
| | 3 -> 4 | DG_CONTROL / DAT_IDENTITY / MSG_OPENDS<br>Capability Negotiation |
| | 4 -> 5 | DG_CONTROL/DAT_USERINTERFACE/ MSG_ENABLEDS |
| MSG_XFERREADY | 6 | For each pending transfer:<br>DG_IMAGE / DAT_IMAGEINFO / MSG_GET<br>DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET<br>DG_CONTROL/DAT_CAPABILITY/ MSG_GETCURRENT |
| | 6 -> 7 | DG_IMAGE / DAT_IMAGExxxxXFER / MSG_GET |
| | 7 -> 6 | DG_CONTROL/ DAT_PENDINGXFERS / MSG_ENDXFER |
| | 6 -> 5 | 假如TW_PENDINGXFERS自动转换到 State 5. 计数等于 0. |
| MSG_CLOSEDSREQ | 5 -> 4 | DG_CONTROL/DAT_USERINTERFACE/ MSG_DISABLEDS |
| | 4 -> 3 | DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS |
| | 3 -> 2 | DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM |
| | 2 -> 1 | 卸载Source Manager |

# 4. Source 的TWAIN接口实现

- 4.1 Source的结构
- 4.2 Sources and the Event Loop
- 4.3 用户界面 Guidelines
- 4.4 数据传输Data Transfers
- 4.5

# 4.1 Source的结构

- Source是一个动态连接dll文件，作为DLL被调用，以.DS为文件后缀，不能单独运行，在Application第一次装载后，Source将分配自己的堆栈空间，用来维护不同的应用程序的对话及配置信息，且只运行一份拷贝。
- Source 存放在Windows目录的TWAIN子目录
- 每个Source都有一个Entry Point(DS_Entry)

# 4.2 Sources and the Event Loop

- 4.2.1 处理事件(Event)
  当Source处于enabled状态 (如States 5, 6, and 7), 应用程序必须传递所有的events 给Source. Event通过TW_EVENT数据结构来传递

假如事件属于Source，则开始决断

设置Return Code为TWRC_DSEVENT
设置TWMessage域为MSG_NULL
处理event

否则

设置Return Code为TWRC_NOTDSEVENT
设置TWMessage域MSG_NULL
立即返回到应用程序 (App)

# 4.2.2 与应用程序通讯

- 有四种情况Source必须传递通知给App
  - 当数据传输已经准备就绪的时候(MSG_XFERREADY)
  - 当他需要让它的用户界面禁用的时候(MSG_CLOSEDSREQ)
  - 当用户已经按了已弹出的Source对话框中的OK按钮的时候
  - 当Source必须报告硬件Event的时候.

# 4.3 用户界面 Guidelines

- 显示用户界面
  - App通过发送DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS来通知Source显示UI

TW_USERINTERFACE 包含下面这些域:

- ShowUI - 假如设为TRUE,Source将显示它自己的UI.

假如为FALSE,App将提供它自己的UI.

- hParent -仅仅被Windows的Source使用. 它指向App的handle. 这将被指定为Source UI的parent
- ModalUI - 将被App 设置为TRUE or FALSE.

# 4.4 数据传输Data Transfers

- 所有Source都必须支持Native和Buffered Memory数据传输模式，默认模式是Native
- 可以通过协商ICAP_XFERMECH capability来选择传输模式。
- 传输通过App进行初始化，一个成功的传输，State将切换到7.
- Source通过返回TWRC_XFERDONE来标识传输成功，State退回到6，如果没有更多数据传输则退回到5
- 假如用户取消传输，Source应该返回TWRC_CANCEL来标示过早的中断。

# 4.4.2.1 Native mode传输模式

- Native mode传输需要一块单独的大RAM块，因此需要有足够的RAM来保证传输成功，当使用该模式时，Source不能被App中断
- 所有的输入设备都支持这种本地数据传输模式，同时它也是TWAIN默认的数据传输模式，并且它还是最容易使用的数据传输模式。但是，它有一定的局限性，它传输的数据必须是DIB图像数据，并且在传输时，会受到系统内存大小限制。
- 传输数据的格式:DIB(Device-IndependentBitmap)
- 使用该模式，在数据传输时Source分配一块单独的内存区域，并把图形数据写入这个内存区域内。然后它通过一个指向该内存地址的指针告诉Application，数据存放在什么地方。你的应用程序通过访问该内存区域去获得具体的图像数据。注意，Application在获得数据后要负责去释放这部分的内存。如果你的图像数据大于系统当前可用内存，会导致传输失败。

# 4.4.2.2 Disk File mode传输模式

- Disk File Mode
  - 该模式是让Application创建一个文件，这个文件用于储存传输的数据，Source将对该文件进行读写操作。Source将把要传输的数据写到该文件中，你的应用程序通过访问该文件，就可以获得传输的数据。
  - 在使用本地模式传输一个大的图像文件时，如果内存不够大，可以考虑使用文件传输模式来传输。文件传输模式与缓存传输模式相比，在使用方法上要简单些，但是该模式在传输速度上比缓存模式的传输速度要慢一些，并且在数据传输完毕后，你的应用程序还必须去管理这个数据文件。
  - Source通过对App DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET 操作的响应来反馈传输的文件格式和文件名

    当Source接收到DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET操作时，应该传输数据到指定文件

# 4.4.2.3 Buffered Memory mode传输模式

- Buffered Memory Mode
  当用该模式传输数据时，Source分配了Buffer空间，App必须负责卸载该Buffer空间。
- 缓存模式在整个传输过程中，将使用一个或多个内存缓存区，内存缓存区的分配和释放工作由Application来控制。在传输过程中，传输数据被当作一个未知格式的位图。Application必须使用TW_IMAGEINFO和TW_IMAGEMEMXFER操作，去得到各个缓存区的信息并把它们正确组织为一个完整的位图。
- 如果使用本地模式或文件模式去传输数据，整个传输过程在只需要一个Triplets操作就可以完成。如果使用缓存模式传输数据,你的应用程序可能需要使用多个Triplets操作，不停地去获得缓存区的数据信息。但是，该传输模式具有很好的灵活性,可以很好的去控制获得的数据，只不过在编程应用上要麻烦一些。

# 4.5 TWAIN最常用的Triplets操作1

这里将对TWAIN中最常用的Triplets操作做一个简单的介绍，为了便于理解和记忆，我将结合前面讲的操作流程顺序去介绍这些常用的Triplets操作。

- I加载SourceManager并获得DSM_Entry入口函数(状态1到2)

  应用程序在调用DSM_Entry函数指针前必须加载SourceManager。这里没有使用Triplets操作。你可以使用LoadLibrary()函数，加载TWAIN_32.DLL文件。并使用GetProcAddress()函数，获得DSM_Entry函数指针

- 打开SourceManager(状态2到3)

  Triplets操作：DG_CONTROL/DAT_PARENT/MSG_OPENDSM

  通过该操作，你可以打开SourceManager，并且还要在你的应用程序中，指定一个窗体作为Source的父窗口。SourceManager将通过该窗体，把Source的消息传递给你的应用程序。

- 选择Source(状态3期间)

  Triplets操作：DG_CONTROL/DAT_IDENTITY/MSG_USERSELECT

  你的应用程序发送该操作后，将显示SourceManager的用户界面，它是一个对话框。这个对话框中显示了系统中所有支持Twain的设备列表。系统默认设备将高亮显示在列表框中。你可以通过该列表框选择你想要的输入设备。

- 打开Source(状态3到4)

  Triplets操作：DG_CONTROL/DAT_IDENTITY/MSG_OPENDS

  该操作可以打开你选择的Source(图像输入设备)，同时，SourceManager会给该Source分配一个唯一的标识符。你要把打开的这个Source放在一个指定的结构中，以便于在后面和该Source进行通讯。

# 4.5 TWAIN最常用的Triplets操作2

- 设置Source的性能参数(状态4期间)

  Triplets操作：DG_CONTROL/DAT_CAPABILITY/MSG_GET

  DG_CONTROL/DAT_CAPABILITY/MSG_SET

  这里有两个Triplets操作，通过使用这两个操作可以去查询当前设备是否支持的某种功能，如果支持，还可以获得设备功能的当前值、默认值、以及可以重新设置的范围。你还可以根据查询的结果，按你的要求去重新设置该功能的当前值。
- 请求从Source获取数据(状态4到5)

  Triplets操作：DG_CONTROL/DAT_USERINTERFACE/MSG_ENABLEDS

  通过该操作，可以让Source显示它的用户界面，Source会去为数据传输作准备。
- 确认数据准备传输(状态5到6)

  Triplets操作：DG_CONTROL/DAT_EVENT/MSG_PROCESSEVENT

  首先要说明一下，从状态5到状态6的这个过程，不是由你的应用程序通过Triplets操作来发起的。而是当Source准备好去传输数据时，它会发出一个事件信号来实现的。你的应用程序应该要去检查这个事件信号。

如何去检查这个事件信号？我们在加载SourceManager时，就为Source指定了一个父窗口，Source会把它事件信号封装成一个Windows的消息结构发送给它的父窗口。你可以在这个窗体的消息循环中去，使用DG_CONTROL/DAT_EVENT/MSG_PROCESSEVENT操作，来判断Source是否有事件发生。MSG_XFERREADY就表示这个过程的状态位从5变为6了。

# 4.5 TWAIN最常用的Triplets操作3

- 开始进行数据传输(状态6到7)

  Triplets操作：DG_IMAGE/DAT_IMAGEINFO/MSG_GET

  DG_IMAGE/DAT_IMAGENATIVEXFER/MSG_GET

  在开始数据传输前，可以通过DG_IMAGE/DAT_IMAGEINFO/MSG_GET操作，去获得将要传输的图像的相关信息，比如位图大小、宽度、长度…。

  通过DG_IMAGE/DAT_IMAGENATIVEXFER/MSG_GET操作，可以实现使用本地传输模式去传输数据。传输结束了，Source将给它的父窗口一个PM_XFERDONE的消息。Source将在DSM_Entry()中返回为一个指向DIB位图的指针。
- 中止传输(状态7到6到5)

  Triplets操作：DG_CONTROL/DAT_PENDINGXFERS/MSG_ENDXFER

  在每次数据传输结束(成功、退出)后，可以发送该操作给Source，去表示应用程序已经接受完了所有的数据了。同时还可以根据它的返回值，去检查是否有其它的图像等待传送。
- 断开TWAIN会话(状态5到4)

  Triplets操作：DG_CONTROL/DAT_USERINTERFACE/MSG_DISABLEDS

  该操作让打开Source失效。
- 关闭Source(状态4到3)

  Triplets操作：DG_CONTROL/DAT_IDENTITY/MSG_CLOSEDS

  该操作可以关闭指定的Source。
- 关闭SourceManager(状态3到2)

  Triplets操作：DG_CONTROL/DAT_PARENT/MSG_CLOSEDSM

  关闭打开的SourceManager。