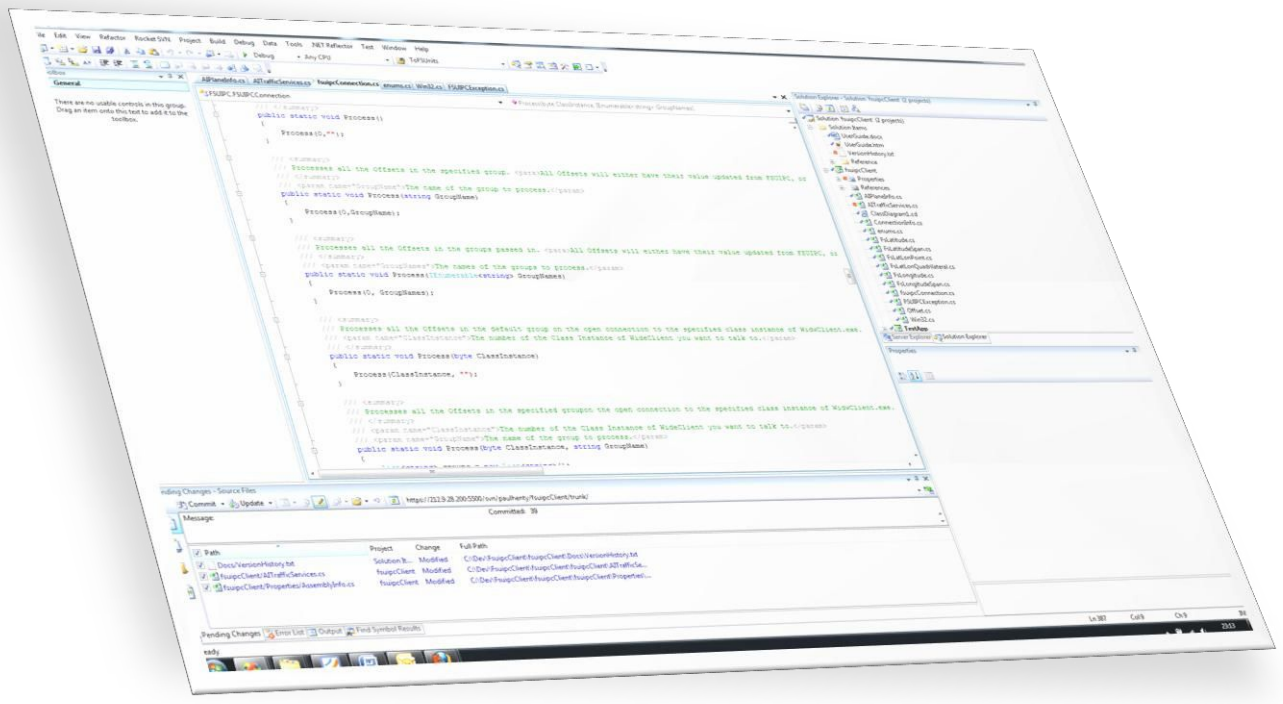
FSUIPC Client DLL for .NET

2.0版本

by Paul Henty

用户指南

Chinese Ver: 0.1
by Boyin Chen



目录

介绍

动态链接库

这个动态链接库允许.NET应用使用一个面向对象的.Net类库连接到Pete Dowson的FSUIPC或WideFS。它也包括了一些使得操作某些FSUIPC数据更容易的帮助类。

许可

这份用户指南

所有的用户都应该阅读用户指南的「开始」章节，它包含了使用这个动态链接库书写应用程序所至少应该知道的内容。

用户书写更高级的程序或他们想要知道动态链接库能做到的更多的功能，应阅读与之相关的其余章节。

C#和Visual Basic .NET的代码示例均有提供。它们大部分都是从这个文件包里的示例应用中节选的。当两种语言的语法相同时（除了C#的断句符分号），仅给出C#的示例。

当书写用到FSUIPC的应用时你必须参考FSUIPC SDK中的文档以完全地理解这些Offset和其他你想要用到的功能。不要完全依赖这一份文档。

参考手册

HTML参考手册提供了这个动态链接库的技术文档。所有的类及其它的全部属性、域、方法均作出了解释。

示例应用

这里提供了一个演示除了深奥的功能不包含但包含了其余全部使用的示例应用的全部源码，包含C#和VB .NET。解决方案创建于Visual Studio 2008但可以在更新版本的Visual Studio中打开，将自动转换。

这是一个有三个选项卡的简单WinForm应用。第一个选项卡从FSUIPC读取数据，也演示向FSUIPC写入数据；第二个演示经纬度转换类；第三个通过展示一个简单的显示活动中的AI飞机的ATC雷达来演示AI交通功能。

发布

当你构建你的项目时，Visual Studio将自动复制这个动态链接库到你的构建目录中。为了使你的程序正常工作，你需要同时发布这个动态链接库。有些版本的Visual Studio提供构建一个发布包或安装程序的功能，动态链接库文件需要选中包含在安装包中。

.NET框架需求和64位元问题

动态链接库可用于.NET 2.0/4.0框架。动态链接库要运行，必须安装它们。你自己的应用可以基于任何版本的.NET框架。

注意动态链接库仅编译为基于x86的系统。这意味着在x64系统上，它将运行在WOW64兼容层上。这是唯一的方法，否则它无法与FSUIPC或WideFS这些作为x86进程运行的程序通信。

如果你在64位操作系统上开发，你需要在配置选项中选中x86(Win32)平台，否则你的应用将无法在64位操作系统上链接到动态链接库。

保证

本软件以「现状」来提供，不提供任何明示或暗示的保证。

译注

译文忠实原文，修改极少。代码注释没有处理，但中文注释在C#示例源码中给出。

开始

安装（快速）

在你的.NET应用项目中添加一个引用到动态链接库。编译器将在构建项目时将其复制到二进制构建目录。

包含了动态链接库中所有类的命名空间叫做`FSUIPC`。`BitArray`类在`System.Collections`命名空间中。

安装（在Visual Studio中一步一步完成）

- 打开或新建你的应用项目。
- 添加一个新的引用
  - 找到或打开解决方案资源管理器。
  - 找到你的应用的主节点并选中。
  - 点击项目菜单，选择引用。
  - 浏览到FSUIPCClient.dll并添加。
- 在任何需要用到这个库的文件头部写入如下语句以导入FSUIPC命名空间。

C#

```
using FSUIPC;
```

VB.NET

```
Imports FSUIPC
```

- 如果你想要将`BitArray`类作为一个数据类型使用，你还需要导入`System.Collections`命名空间（如果Visual Studio）没有自动导入的话。

打开与FSUIPC的连接

使用`FSUIPCConnection`静态类中的`Open()`方法。若在连接过程中出现了任何问题，这个方法将抛出包含错误码的`FSUIPCException`异常。

C#

```
try
{
    // 尝试打开FSUIPC连接（任何版本的FS）
    FSUIPCConnection.Open();
    // 成功打开
}
catch (Exception ex)
{
    // 发生错误，展示错误信息
    MessageBox.Show(ex.Message, AppTitle, MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Visual Basic.NET

```
Try
    ' 尝试打开FSUIPC连接（任何版本的FS）
    FSUIPCConnection.Open()
    ' 成功打开
Catch ex As Exception
    ' 发生错误，展示错误信息
    MessageBox.Show(ex.Message, AppTitle, MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

如果你需要将你的程序限制在一个特定的FS版本上运行，你可以使用包含一个`FlightSim`枚举的重载。

如果FSUIPC没有在给定的FS版本中运行，`FSUIPCConnection`类将抛出一个`FSUIPCErrorCode`属性为`FSUIPC_ERR_VERSION`的`FSUIPCException`异常。

- 尝试打开一个向FS2004的连接。

```
FSUIPCConnection.Open(FlightSim.FS2K4);
```

关闭与FSUIPC的连接

使用`FSUIPCConnection`静态类中的`Close()`方法。

```
FSUIPCConnection.Close();
```

这个方法清空用于与FSUIPC通信的未管理的内存。你需要确保在你的应用退出前调用了这个方法。

如果你的应用因为某些原因丢失了与FSUIPC的连接，你也需要调用这个方法。在你打开一个新的连接前，你必须关闭损坏的这个。

将你的所需内容注册为一个Offset

你必须创建一个`Offset`类的实例来告诉动态链接库你要向这些FSUIPC Offset读写数据：

C#

```
Offset<DataType>
```

Visual Basic.NET

```
Offset(Of DataType)
```

创建实例时，你必须指定一个类型（`DataType`）。这是Offset将要以某种.NET类型储存的类型。

举个例子，如果这个FSUIPC Offset是一个4字节长的整型，你需要将类型指定为Int32或是你的语言中与Int32类型相同的原生类型（C# = `int`，VB.NET = `Integer`）。

在这个例子中我们创建了一个名为「airspeed」的`Offset`类的实例。它被设定从0x02BC（指示空速），一个4字节整数的内存空间读取数据。因此我们通过这样指定这个`Offset`实例的类型：

C#

```
Offset<int> airspeed = new Offset<int>(0x02BC);
```

Visual Basic.NET

```
Dim airSpeed As Offset(Of Integer) = New FSUIPC.Offset(Of Integer)(&H2BC)
```

下列.NET类型可用作Offset类型。这个表格也展示了用于C#和VB.NET的别名和适用于哪类FSUIPC Offset。

FSUIPC Offset长度	.NET框架类型	C#	VB.NET
1	`System.Byte`	`Byte`	`Byte`
2	`System.Int16` `System.UInt16`	`Short` `Ushort`	`Short` `UShort`
4	`System.Int32` `System.UInt32`	`Int` `Uint`	`Integer` `UInteger`
4 (32位浮点数)	`System.Single`	`Float`	`Single`
8	`System.Int64` `System.UInt64`	`Long` `Ulong`	`Long` `ULong`
8 (64位浮点数)	`System.Double`	`Double`	`Double`
不限，字符串	`System.String`	`String`	`String`
不限，仅用于BitArray类型和位域Offset	`System.Collections.BitArray`		
不限	`Array of System.Byte`	`byte[]`	`Byte()`

当你创建一个`Offset`，类型为`String`、`BitArray`或任何`Byte`数组时你必须同时指定你想要从FSUIPC读取的数据长度。通常单位是字节。请参照示例应用中读取不同类型数据的部分。

从一个Offset中读取数据

首先，创建一个`Offset`类的实例。查看更多细节解释，请转到上面的「将你的所需内容注册为一个Offset」部分。

下一步是调用`FSUIPCConnection`中的`Process()`方法。

这将从FSUIPC中读取值并存储在Offset实例中的`Value`属性里。如果这个过程中出现了任何问题，这个方法将抛出包含错误码的`FSUIPCException`异常。更多细节查看「处理FSUIPC错误」部分。

在这个例子中，处理了空速，它的值被转换成节，显示在一个Textbox中。

C#

```
FSUIPCConnection.Process();
double airpeedKnots = (double)airspeed.Value / 128d;
this.txtIAS.Text = airpeedKnots.ToString("f1");
```

Visual Basic.NET

```
FSUIPCConnection.Process()
Dim airpeedKnots As Double = airSpeed.Value / 128D
Me.txtIAS.Text = airpeedKnots.ToString("f1")
```

下一次你想要更新空速的值，仅需调用`Process()`。你不需要再重新创建一个`Offset`实例。

向一个Offset中写入数据

任何你创建的Offset都可以被写入。

大多数情况下，当你调用`Process()`时它们将被读取。如果你想向FSUIPC写入一个新值，你需要改变Offset实例的`Value`属性。

下一次你调用`Process()`时，这个Offset的值将被写入而不是读取。写入后，这个Offset又会继续读取。

当你创建一个`Offset<>`时，你可以将其指定为一个只写的Offset。更多细节见「只写类型的Offset」部分。

这个例子展示了打开主航空电子电门的步骤：向Offset 0x2E80写入1。注意每行代码都来自一个应用的不同部分。

C#

```
Offset<int> avionics = new Offset<int>(0x2E80);
...
avionics.Value = 1;
...
FSUIPCConnection.Process();
```

Visual Basic.NET

```
Dim avionics As Offset(Of Integer) = New FSUIPC.Offset(Of Integer)(&H2E80)
...
avionics.Value = 1
...
FSUIPCConnection.Process()
```

处理FSUIPC错误

在FSUIPC操作中发生了任何错误将会抛出一个FSUIPCException异常。

你需要通过捕获通用异常类或捕获FSUIPCException类来处理这个异常。

这两种情况下，异常的Message属性都是对错误发生的复杂的解释，原始的C语言SDK的FSUIPC错误码将作为Message字符串的一部分包含进来。

如果你选择处理FSUIPCException类，这里有一个额外的属性叫做FSUIPCErrorCode，它将把原始的C语言SDK的FSUIPC错误码分割，作为FSUIPCError枚举返回。如果需要，你可以通过这个来检查特定的错误。

FSUIPCException是被FSUIPCConnection的Open()和Process()方法抛出的。

在这个例子中我们捕获可能在Process()中抛出的FSUIPCException。我们在寻找一个特定的错误：FSUIPC_ERR_SENDMSG，它在FSUIPC连接丢失时抛出。最可能的原因是FS被关闭了。

为了避免应用因未处理的异常而崩溃，下面的例子捕获这个错误然后通过更平滑的方式处理。它向用户显示消息，然后激活「btnStart」按钮，用户可以尝试重新连接。它同时关闭当前的连接。

C#

```
try
{
    FSUIPCConnection.Process();
    // 成功执行
}
catch (FSUIPCException ex)
{
    if (ex.FSUIPCErrorCode == FSUIPCError.FSUIPC_ERR_SENDMSG)
    {
        // FSUIPC连接丢失
        // 展示消息，重新点亮连接按钮
        // 同时关闭损坏的连接
        this.btnStart.Enabled = true;
        FSUIPCConnection.Close();
        MessageBox.Show("The connection to Flight Sim has been lost.");
    }
    else
    {
        // 不是连接断开而是其他错误，重抛异常
        throw ex;
    }
}
```

Visual Basic.NET

```
Try
    FSUIPCConnection.Process()
    ' 成功执行
Catch exFSUIPC As FSUIPCException
    If exFSUIPC.FSUIPCErrorCode = FSUIPCError.FSUIPC_ERR_SENDMSG Then
        ' FSUIPC连接丢失
        ' 展示消息，重新点亮连接按钮
        ' 同时关闭损坏的连接
        Me.btnStart.Enabled = True
        FSUIPCConnection.Close()
        MessageBox.Show("The connection to Flight Sim has been lost.")
   Else
       ' 不是连接断开而是其他错误，重抛异常
       Throw exFSUIPC
   End If
End Try
```

高级功能

只写类型的Offset

当你创建一个Offset时，你可以通过将`WriteOnly`参数设置为`true`将其作为只写Offset。这意味着它的值永远不会在`Process()`被调用时被读取。

这个例子展示了创建只写Offset的过程。Offset是暂停功能，内存地址0x0262。

C#

```
Offset<short> pause = new Offset<short>(0x0262, true);
```

Visual Basic.NET

```
Dim pause As Offset(Of Short) = New FSUIPC.Offset(Of Short)(&H262, True)
```

你也可以在任何时候通过将Offset的`WriteOnly`属性设置为`true`来将其作为只写Offset。

注意只有上次调用`Process()`后`Value`属性被改变，调用`Process()`时只写Offset的值才会写入。

分组管理Offset

你可以在你的应用中通过Offset分组的方式控制某些Offset读取或写入。

典型地，应用需要在不同的时刻请求不同的数据集。比如，飞机的位置可能几秒就需要获取一次，然而气象数据也许只需要十分钟更新一次就行了。

你可以通过把Offset放入不同的组，来在不同的时刻执行它们。

当你创建一个Offset时，你可以选择性地指定一个「组」。如果没有指定，将放入默认组。默认组在没有定义组的重载被调用时被用到。

在这个例子中，一个Offset被创建在一个名为「AircraftInfo」的组中。Offset从0x3160内存中请求飞机类型。

C#

```
Offset<string> aircraftType = new Offset<string>("AircraftInfo", 0x3160, 24);
```

Visual Basic.NET

```
Dim aircraftType As Offset(Of String) = New FSUIPC.Offset(Of String)("AircraftInfo", &H3160,
24)
```

执行组内的Offset，使用指定了组名的重载。

```
FSUIPCConnection.Process("AircraftInfo");
```

你可以通过调用使用`IEnumerable`组名列表的重载在一次`Process()`调用中执行多个组。

下面是一个执行多个组的简单示例。

C#

```
private Offset<string> aircraftName = new Offset<string>("info", 0x3D00, 24);
private Offset<int> pitch = new Offset<int>("attitude", 0x0578);
private Offset<int> bank = new Offset<int>("attitude", 0x057c);

public void processGroups()
{
    List<string> groupsToProcess = new List<string>();
    if (needPlaneName)
    {
        groupsToProcess.Add("info");
    }
    if (needPlaneAttitude)
    {
        groupsToProcess.Add("attitude");
    }
    FSUIPCConnection.Process(groupsToProcess);
}
```

VisualBasic.NET

```
Private aircraftName As Offset(Of String) = New Offset(Of String)("info", &H3D00, 24)
Private pitch As Offset(Of Integer) = New Offset(Of Integer)("attitude", &H578)
Private bank As Offset(Of Integer) = New Offset(Of Integer)("attitude", &H57C)

Public Sub processGroups()
    Dim groupsToProcess As List(Of String) = New List(Of String)()
    If (needPlaneName) Then
        groupsToProcess.Add("info")
    End If
    If (needPlaneAttitude) Then
        groupsToProcess.Add("attitude")
    End If
    FSUIPCConnection.Process(groupsToProcess)
End Sub
```

Alternatively you can use a list of literal group names by creating a new string array:

C#

```
FSUIPCConnection.Process(new string[] {"info","attitude"});
```

VisualBasic.NET

```
FSUIPCConnection.Process(New String() {"info", "attitude"})
```

在一次调用Process()中执行多个组比多次调用Process()更好。真实执行（与FSUIPC进行数据交换）速度是很慢的。`Process()`调用得越少，你的应用和FS运行得就越快。数据读写总量的变化几乎不影响数据交换时的速度。

独立管理Offset

你可以控制Offset是否被读写。

使用`Disconnect()`方法以阻止Offset被读写。

这将把Offset从`FSUIPCConnection`类中注销，于是Offset就不会被读写了。

你可以通过将`AfterNextProcess`参数设置为`true`来使得下一次`Process()`调用后Offset被注销。

如果你想要重新使用这个Offset，你可以使用`Reconnect()`方法重新连接。

你可以通过将`ForNextProcessOnly`参数设置为`true`，选择性地为一次`Process()`调用重连Offset。

使用BitArray管理位域

这个动态链接库允许你以`BitArray`类型管理位域类型的Offset，这比位操作方便。

在这个例子中使用了内存为0x0D0C的灯光Offset。在这个两字节的Offset中每个比特（位）代表其对应的灯光明灭情况。

我们像下面这样声明这个Offset（长度传递为2，这个Offset是2字节长）：

C#

```
Offset<BitArray> lights = new Offset<BitArray>(0x0D0C, 2);
```

Visual Basic.NET

```
Dim lights As Offset(Of BitArray) = New FSUIPC.Offset(Of BitArray)(&HD0C, 2)
```

这个Offset的`Value`属性是`BitArray`类型。我们可以用索引器访问每个比特。比如：比特5（从0开始）代表仪表灯。在下面的例子中，比特5的值用于设置一个Checkbox的`Checked`属性，这用于向用户展示仪表灯的状态。

C#

```
this.chkInstuments.Checked = lights.Value[5];
```

Visual Basic.NET

```
Me.chkInstuments.Checked = lights.Value(5)
```

我们仅需要设置合适的比特（5号）就可以打开仪表灯。

C#

```
this.lights.Value[5] = true;
```

Visual Basic.NET

```
Me.lights.Value(5) = True
```

当你下一次调用Process()时，灯就会点亮了。

从FSUIPC读取原始数据块

为了达到最大的灵活性，你可以使用字节数组向FSUIPC读写任何长度的数据。然后你就可以以任何你想要的方式处理数据。比如，使用FSUIPC的天气功能。

通常地，你将需要把数组中的字节转换为原生的数据变量。`BitConverter`类对此很有用。查看你的.NET帮助来了解如何使用这个类。

下面的例子展示了创建一个从0x0238内存读取10字节包含关于本地日期和时间数据的Offset的过程。

C#

```
Offset<byte[]> fsLocalDateTime = new Offset<byte[]>(0x0238, 10);
```

Visual Basic.NET

```
Dim fsLocalDateTime As Offset(Of Byte()) = New FSUIPC.Offset(Of Byte())(&H238, 10)
```

你将需要自己转换这些原始字节数据。在这个例子中，要分割出日期和时间的组成。

连接到WideClient的多个实例

在同一台计算机上运行多个WideClient是可能的，每个WideClient连接到不同的FS（WideServer）计算机上。每个WideClient实例都拥有设置指向正确FS计算机的它自己的.ini文件。

这个.ini文件一定包含了这一行：

```
ClassInstance=x
```

x是一个0到255之间的数。它告诉动态链接库应该与哪个WideClient通信。（这个功能的更多细节，以及如何配置WideClient的ini文件，可以在WideFS的文档中找到。）

在你的软件中，你需要使用一个包含ClassInstance参数的FSUIPCConnection.Open()方法重载。你的软件需要通信的每一个实例，都需要调用Open()打开。

同样地，每一次你调用FSUIPCConnection.Process()方法，你也需要使用包含ClassInstance参数的重载。

在调用Process()之后，所有Offset将从WideClient取得数据。

你可以调用Close(ClassInstance)来关闭一个特定的实例，调用Close()关闭全部。

这是一些示例代码，它打开两个WideClient实例（0和1），从它们那儿获取飞机的名字：

C#

```
Offset<string> aircraftName = new Offset<string>("aircraft name", 0x3D00, 24);

FSUIPCConnection.Open(0, FlightSim.Any);
FSUIPCConnection.Open(1, FlightSim.Any);

FSUIPCConnection.Process(0, "aircraft name");
string strAircraftName1 = aircraftName.Value;

FSUIPCConnection.Process(1,"aircraft name");
string strAircraftName2 = aircraftName.Value;

FSUIPCConnection.Close();
```

VisualBasic.NET

```
Dim aircraftName As Offset(Of String) = New Offset(Of String)("aircraft name", &H3D00, 24)

FSUIPCConnection.Open(0, FlightSim.Any)
FSUIPCConnection.Open(1, FlightSim.Any)

FSUIPCConnection.Process(0, "aircraft name")
Dim strAircraftName1 As String = aircraftName.Value

FSUIPCConnection.Process(1, "aircraft name")
Dim strAircraftName2 As String = aircraftName.Value

FSUIPCConnection.Close()
```

引用和垃圾回收问题

确保你一直持有你创建的Offset实例的引用。如果它们超出范围你将没法重新获得引用。

它们将仍然在FSUIPCConnection类中注册，在调用Process()时它们仍然被更新。然而垃圾回收器不会处理它们，因为FSUIPCConnection仍然持有这些引用。

如果你使用合理的分组体系，你就不用担心Offset超出范围。如果还是发生了，这不是大问题。因为你不会再去执行这个分组。它们仍然占用一小块内存，但是它们永远不会被读取了。

如果你真的不再需要一个Offset，你想让它被处理，在你丢掉对它的引用前，确保先断开（Disconnect）它。

如果你的应用有很多一次性的读取，你可以将它们放在一个组内，使用完毕后像这样将整个组删除：

```
FSUIPCConnection.DeleteGroup(GroupName);
```

经纬度转换类

读取和显示经纬度

动态链接库提供了两个辅助类，这使得从FSUIPC读写经纬度更为方便。

FSUIPC使用一个8字节整型格式存储经纬度。在你的应用使用它们前，需要将其转换为十进制度数值或字符串。

`FsLongitude`和`FsLatitude`类帮你做这些事情。

下面的示例展示了读取并将玩家的经纬度写入两个Textbox。经纬度使用含有8字节FS单位的原始数据参数的重载创建。（在FSUIPC中有的经纬度Offset使用低精度的4字节FS单位。也提供能够理解这个值的构造器。）

接着，这些类使用`ToString()`重载将其显示为字符串。（注意这里还有另一个`ToString()`的重载，它可以让你定义格式。例如，如果你想要分或秒，亦或是小数点位数。查看参考手册或Intellisense获得更多信息。）

C#

```
private Offset<long> playerLatitude = new Offset<long>(0x0560);
private Offset<long> playerLongitude = new Offset<long>(0x0568);
private void displayCurrentPosition()
{
    // Create new instances of FsLongitude and FsLatitude using the
    // raw 8-Byte data from the FSUIPC Offsets
    FSUIPCConnection.Process()
    FsLongitude lon = new FsLongitude(playerLongitude.Value);
    FsLatitude lat = new FsLatitude(playerLatitude.Value);
    // Use the ToString() method to output in human readable form:
    this.txtLatitude.Text = lat.ToString();
    this.txtLongitude.Text = lon.ToString();
}
```

VisualBasic.NET

```
Dim playerLatitude As Offset(Of Long) = New Offset(Of Long)(&H560)
Dim playerLongitude As Offset(Of Long) = New Offset(Of Long)(&H568)

Private Sub DisplayCurrentPosition()
   ' Create new instances of FsLongitude and FsLatitude using the
   ' raw 8-Byte data from the FSUIPC Offsets
   FSUIPCConnection.Process()
   Dim lon As FsLongitude = New FsLongitude(playerLongitude.Value)
   Dim lat As FsLatitude = New FsLatitude(playerLatitude.Value)
   ' Use the ToString() method to output in human readable form:
   Me.txtLatitude.Text = lat.ToString()
   Me.txtLongitude.Text = lon.ToString()
End Sub
```

写入经纬度和偏置点

这个例子展示了将当前位置移动到伦敦希思罗机场27L跑道的过程。功能是围绕设置当前的经纬度Offset实现的。这些Offset接收FS单位的8字节整型数据。你可以使用`FsLongitude`和`FsLatitude`类的`ToFSUnits8()`方法代替手动转换。

某些FSUIPC的经纬度Offset是低精度的4字节FS单位值。转换到这个值，使用`ToFSUnits4()`方法。

我们只有跑道入口的经纬度数据。我们需要将飞机放置在更前一些的位置以便起飞。我们使用`OffsetByMetres()`来计算跑道航向方向距入口150米处的点。当然还有使用海里和英尺的方法。这些方法需要方位角（原点与目标点的方位）和距离来工作。

在这个例子中我们还要将将FS设置为位移模式，设置航向和高度然后刷新场景。

C#

```
private Offset<long> playerLatitude = new Offset<long>(0x0560);
private Offset<long> playerLongitude = new Offset<long>(0x0568);
private Offset<uint> playerHeadingTrue = new Offset<uint>(0x0580);
private Offset<long> playerAltitude = new Offset<long>(0x0570);
private Offset<short> slewMode = new Offset<short>(0x05DC, true);
private Offset<int> sendControl = new Offset<int>(0x3110, true);
private readonly int REFRESH_SCENERY = 65562;

private void MoveToEGLL()
{
    // Put the sim into Slew mode
    slewMode.Value = 1;
    FSUIPCConnection.Process();
    // Make a new point representing the centre of the threshold for 27L
    FsLatitude lat = new FsLatitude(51.464943d);
    FsLongitude lon = new FsLongitude(-0.434046d);
    FsLatLonPoint newPos = new FsLatLonPoint(lat, lon);
    // Now move this point 150 metres up the runway
    // Use one of the OffsetBy methods of the FsLatLonPoint class
    double rwyTrueHeading =  269.7d;
    newPos = newPos.OffsetByMetres(rwyTrueHeading, 150);
    // Set the new position
    playerLatitude.Value = newPos.Latitude.ToFSUnits8();
    playerLongitude.Value = newPos.Longitude.ToFSUnits8();
    // set the heading and altitude
    playerAltitude.Value = 0;
    playerHeadingTrue.Value = (uint)(rwyTrueHeading * (65536d * 65536d) / 360d);
```

```
    FSUIPCConnection.Process();
    // Turn off the slew mode
    slewMode.Value = 0;
    FSUIPCConnection.Process();
    // Refresh the scenery
    sendControl.Value = REFRESH_SCENERY;
    FSUIPCConnection.Process();
}
```

Visual Basic.NET

```
Dim playerLatitude As Offset(Of Long) = New Offset(Of Long)(&H560)
Dim playerLongitude As Offset(Of Long) = New Offset(Of Long)(&H568)
Dim playerHeadingTrue As Offset(Of UInteger) = New Offset(Of UInteger)(&H580)
Dim playerAltitude As Offset(Of Long) = New Offset(Of Long)(&H570)
Dim slewMode As Offset(Of Short) = New Offset(Of Short)(&H5DC, True)
Dim sendControl As Offset(Of Integer) = New Offset(Of Integer)(&H3110, True)
Const REFRESH_SCENERY As Integer = 65562

Private Sub MoveToEGLL()
    ' Put the sim into Slew mode
    slewMode.Value = 1
    FSUIPCConnection.Process()
    ' Make a new point representing the centre of the threshold for 27L
    Dim lat As FsLatitude = New FsLatitude(51.464943D)
    Dim lon As FsLongitude = New FsLongitude(-0.434046D)
    Dim newPos As FsLatLonPoint = New FsLatLonPoint(lat, lon)
    ' Now move this point 150 metres up the runway
    ' Use one of the OffsetBy methods of the FsLatLonPoint class
    Dim rwyTrueHeading As Double = 269.7D
    newPos = newPos.OffsetByMetres(rwyTrueHeading, 150)
    ' Set the new position
    playerLatitude.Value = newPos.Latitude.ToFSUnits8()
    playerLongitude.Value = newPos.Longitude.ToFSUnits8()
    ' set the heading and altitude
    playerAltitude.Value = 0
    playerHeadingTrue.Value = rwyTrueHeading * (65536D * 65536D) / 360D
    FSUIPCConnection.Process()
    ' Turn off the slew mode
    slewMode.Value = 0
    FSUIPCConnection.Process()
    ' Refresh the scenery
    sendControl.Value = REFRESH_SCENERY
    FSUIPCConnection.Process()
End Sub
```

两点间的距离和方位

`FsLatLonPoint`类还有一个计算两点间方位和距离的方法。这个例子计算到伦敦希思罗机场的距离和真航向。

C#

```
private Offset<long> playerLatitude = new Offset<long>(0x0560);
private Offset<long> playerLongitude = new Offset<long>(0x0568);

private void showDirectionAndHeadingToEGLL()
{
    // Setup the info for EGLL
    FsLatitude lat = new FsLatitude(51, 28, 39.0d);
    FsLongitude lon = new FsLongitude(0, -27, -41.0d);
    EGLL = new FsLatLonPoint(lat, lon);
    // Next get the point for the current plane position
    lon = new FsLongitude(playerLongitude.Value);
    lat = new FsLatitude(playerLatitude.Value);
    FsLatLonPoint currentPosition = new FsLatLonPoint(lat, lon);
    // Get the distance between here and EGLL
    double distance = currentPosition.DistanceFromInNauticalMiles(EGLL);
```

```
    // Get the bearing (True)
    double bearing = currentPosition.BearingTo(EGLL);
    // Write the distance to the text box formatting to 2 decimal places
    this.txtDistance.Text = distance.ToString("N2");
    // Display the bearing in whole numbers
    this.txtBearing.Text = bearing.ToString("F0");
}
```

VisualBasic.NET

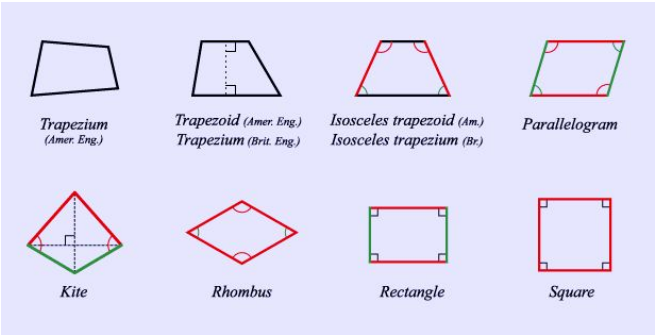
```
Dim playerLatitude As Offset(Of Long) = New Offset(Of Long)(&H560)
Dim playerLongitude As Offset(Of Long) = New Offset(Of Long)(&H568)

Private Sub showDirectionAndHeadingToEGLL()
    ' Setup info for EGLL
    Dim lat As FsLatitude = New FsLatitude(51, 28, 39D)
    Dim lon As FsLongitude = New FsLongitude(0, -27, -41D)
    EGLL = New FsLatLonPoint(lat, lon)
    ' get current plane position
    lon = New FsLongitude(playerLongitude.Value)
    lat = New FsLatitude(playerLatitude.Value)
    Dim currentPosition As FsLatLonPoint = New FsLatLonPoint(lat, lon)
    ' Get the distance between here and EGLL
    Dim distance = currentPosition.DistanceFromInNauticalMiles(EGLL)
    ' Get the bearing (True)
    Dim bearing As Double = currentPosition.BearingTo(EGLL)
    ' Write the distance to the text box formatting to 2 decimal places
    Me.txtDistance.Text = distance.ToString("N2")
    ' Display the bearing in whole numbers and tag on a degree symbol
    Me.txtBearing.Text = bearing.ToString("F0") & Chr(&HB0)
End Sub
```

判断某点是否在给定区域内

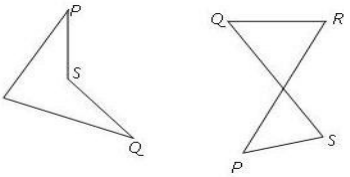
`FsLatLonQuadrilateral`类可以让你在地球上定义一个四边形区域。要定义区域，给定四个角位置。

这个区域应该是四条线围成的。注意必须是凸四边形，凹四边形不行的。

这些形状可以：



这些形状不行：



创建一个`FsLatLonQuadrilateral`，要传递定义了四个角的`FsLatLonPoint`类。这些角可以无序传入，在类的内部，它们以东北、西南等方式储存。

下面的例子创建了一个代表伦敦希思罗机场的四边形。

C#

```
// First corner
FsLatitude lat = new FsLatitude(51,27.34);
FsLongitude lon = new FsLongitude(0,-26.97);
FsLatLonPoint point0 = new FsLatLonPoint(lat, lon);

// Second corner
lat = new FsLatitude(51, 29.03);
lon = new FsLongitude(0, -24.48);
FsLatLonPoint point1 = new FsLatLonPoint(lat, lon);

// Third corner
lat = new FsLatitude(51, 29.02);
lon = new FsLongitude(0, -29.66);
FsLatLonPoint point2 = new FsLatLonPoint(lat, lon);

// Forth corner
lat = new FsLatitude(51, 27.61);
lon = new FsLongitude(0, -29.65);
FsLatLonPoint point3 = new FsLatLonPoint(lat, lon);

// Create Quadrilateral
FsLatLonQuadrilateral quad = new FsLatLonQuadrilateral(point0, point1, point2, point3);
```

Visual Basic.NET

```
' First corner
Dim lat As FsLatitude = New FsLatitude(51, 27.34)
Dim lon As FsLongitude = New FsLongitude(0, -26.97)
Dim point0 As FsLatLonPoint = New FsLatLonPoint(lat, lon)

' Second corner
lat = New FsLatitude(51, 29.03)
lon = New FsLongitude(0, -24.48)
Dim point1 As FsLatLonPoint = New FsLatLonPoint(lat, lon)

' Third corner
lat = New FsLatitude(51, 29.02)
lon = New FsLongitude(0, -29.66)
Dim point2 As FsLatLonPoint = New FsLatLonPoint(lat, lon)

' Forth corner
lat = New FsLatitude(51, 27.61)
lon = New FsLongitude(0, -29.65)
Dim point3 As FsLatLonPoint = New FsLatLonPoint(lat, lon)

' Create Quadrilateral
Dim quad As FsLatLonQuadrilateral = New FsLatLonQuadrilateral(point0, point1, point2, point3)
```

这个类的主要作用是判断一个给定的点是否在定义的区域内。比如，使用上面我们创建的四边形，我们可以告诉用户Ta是否在希思罗机场内。

C#

```
lat = new FsLatitude(playerLatitude.Value);
lon = new FsLongitude(playerLongitude.Value);
FsLatLonPoint playerPosition = new FsLatLonPoint(lat, lon);

if (quad.ContainsPoint(playerPosition))
{
    // Player is inside the boundary of EGLL
    // Do stuff
}
```

Visual Basic.NET

```
lat = New FsLatitude(playerLatitude.Value)
lon = New FsLongitude(playerLongitude.Value)
Dim playerPosition As FsLatLonPoint = New FsLatLonPoint(lat, lon)

If quad.ContainsPoint(playerPosition) Then
   ' Player is inside the boundary of EGLL
   ' Do stuff
End If
```

判断玩家是否在跑道上

如果你知道跑道四个角的经纬度，你可以简单地创建`FsLatLonQuadrilateral`，然后测试飞机是否在这个区域内。

更普遍的情况时，我们可能知道关于一条跑道不同的信息集合。比如，Pete Dowson的Make Runways程序生成包含跑道入口经纬度、方向、长度和宽度的文本文件。

`FsLatLonQuadrilateral`类中有一个辅助方法可以使你使用下面这些信息创建一个四边形：

- 入口中点经纬度
- 宽度（英尺）
- 长度（英尺）
- 真航向

这个方法返回一个普通的`FsLatLonQuadrilateral`，你可以向之前章节描述的那样使用它。

下面的例子判断玩家是否在伦敦希思罗机场的27L跑道上。关于跑道的信息（航向、宽度等）在示例中很难书写。我从Make Runways程序中生成了这些信息。一个需要处理跑道的真正的应用应该从数据库中获取这些信息，或者从Make Runways文本文件中截取。

在一个真正的应用中你不会想着每时每刻去生成一个四边形。当你第一次知道需要测试的跑道时生成它，然后使用同一个`FsLatLonQuadrilateral`对象去测试就好了。示例程序提供了这个方法的示例。

C#

```
private Offset<long> playerLatitude = new Offset<long>(0x0560);
private Offset<long> playerLongitude = new Offset<long>(0x0568);
private Offset<short> onGround = new Offset<short>(0x0366);

private void CheckPlayerIsOnRunway()
{
    FsLongitude lon = new FsLongitude(playerLongitude.Value);
    FsLatitude lat = new FsLatitude(playerLatitude.Value);
    // Get the point for the current plane position
    FsLatLonPoint currentPosition = new FsLatLonPoint(lat, lon);
    // Now define the Quadrangle for the 27L (09R) runway.
```

```
    // We could just define the four corner Lat/Lon points if we knew them.
    // In this example however we're using the helper function to calculate the points
    // from the runway information.  This is the kind of info you can find in the output files
    // from Pete Dowson's MakeRunways program.
    FsLatitude rwyThresholdLat = new FsLatitude(51.464943d);
    FsLongitude rwyThresholdLon = new FsLongitude(-0.434046d);
    double rwyMagHeading = 272.7d;
    double rwyMagVariation = -3d;
    double rwyLength = 11978d;
    double rwyWidth = 164d;

    // Call the static helper on the FsLatLonQuarangle class to generate
    // the Quadrilateral for this runway...
    FsLatLonPoint thresholdCentre = new FsLatLonPoint(rwyThresholdLat, rwyThresholdLon);
    double trueHeading = rwyMagHeading + rwyMagVariation;
    runwayQuad = FsLatLonQuadrilateral.ForRunway(thresholdCentre, trueHeading, rwyWidth,
rwyLength);

    // Now check if the player is on the runway:
    // Test is the plane is on the ground and if the current position is in the bounds of
    // the runway Quadrangle we calculated in the constructor above.
    if (this.onGround.Value == 1 && runwayQuad.ContainsPoint(currentPosition))
    {
        // Player is on the runway
        // Do Stuff
    }
}
```

Visual Basic.NET

```
Dim playerLatitude As Offset(Of Long) = New Offset(Of Long)(&H560)
Dim playerLongitude As Offset(Of Long) = New Offset(Of Long)(&H568)
Dim onGround As Offset(Of Short) = New Offset(Of Short)(&H366)

Private Sub CheckPlayerIsOnRunway()
    Dim lon As FsLongitude = New FsLongitude(playerLongitude.Value)
    Dim lat As FsLatitude = New FsLatitude(playerLatitude.Value)
    ' Get the point for the current plane position
    Dim currentPosition As FsLatLonPoint = New FsLatLonPoint(lat, lon)
    ' Now define the Quadrangle for the 27L (09R) runway.
    ' We could just define the four corner Lat/Lon points if we knew them.
    ' In this example however we're using the helper function to calculate the points
    ' from the runway information.  This is the kind of info you can find in the output files
    ' from Pete Dowson's MakeRunways program.
    Dim rwyThresholdLat As FsLatitude = New FsLatitude(51.464943D)
    Dim rwyThresholdLon As FsLongitude = New FsLongitude(-0.434046D)
    Dim rwyMagHeading As Double = 272.7D
    Dim rwyMagVariation As Double = -3D
    Dim rwyLength As Double = 11978D
    Dim rwyWidth As Double = 164D

    ' Call the static helper on the FsLatLonQuarangle class to generate
    ' the Quadrilateral for this runway...
    Dim thresholdCentre As FsLatLonPoint = New FsLatLonPoint(rwyThresholdLat, rwyThresholdLon)
    Dim trueHeading As Double = rwyMagHeading + rwyMagVariation
    runwayQuad = FsLatLonQuadrilateral.ForRunway(thresholdCentre, trueHeading, rwyWidth,
rwyLength)

    ' Now check if the player is on the runway:
    ' Test is the plane is on the ground and if the current position is in the bounds of
    ' the runway Quadrangle we calculated in the constructor above.
    If onGround.Value = 1 And runwayQuad.ContainsPoint(currentPosition) Then
        ' Player is on the runway
        ' Do Stuff
    End If
End Sub
```

AI交通

获取AI交通信息

AITrafficServices类提供AI交通信息。FSUIPCConnection类提供一个这个类的可用实例。如果你想使用AI交通服务，我建议你在程序开始时获取一个这个实例的引用，然后使用它。注意在连接打开前，AITrafficServices实例不会初始化。

这不是必须的，但它使得写代码更容易，代码量更少。大多数的示例都尝试使用了全部的语法，以把这个对象模型解释得更清楚。

你需要调用AITrafficServices的RefreshAITrafficInformation()来从FSUIPC中读取最近的AI交通。注意你不需要调用Process()，交通服务独立于你应用调用的任何Process()。

如果你只需要获取地面或空中的交通，这里有一个重载可以让你选择性地刷新它们。

每当你调用RefreshAITrafficInformation()时，你会得到三个可用的AIPlaneInfo实例的强类型列表。它们是：

```
AITrafficInformation.AirbourneTraffic
AITrafficInformation.GroundTraffic
AITrafficInformation.AllTraffic
```

你可以在每架上重新声明这些列表。这些飞机按照距玩家的距离排序，最近的下标索引为0。如果你对某一或某些AI飞机感兴趣，你可以创建一个对应AIPlaneInfo对象的引用。在你下一次调用RefreshAITrafficInformation()时它们将更新。

在下面的例子中，展示了将每架飞机的AtcID和状态显示在ListBox的过程。AIPlaneInfo类中还有很多其他可用的属性。查看参考手册或Intellisense以获取更多内容。

C#

```
private void readAI()
{
    // Get the AI Traffic info back from FSUIPC
    // This happens immediatly, no need to call Process()
    FSUIPCConnection.AITrafficServices.RefreshAITrafficInformation();
    // The following code iterates through the list and adds each plane's ATCid to a listbox
    // along with it's state in brackets. (Landing, enroute etc..)
    this.listBox1.Items.Clear();
    foreach (AIPlaneInfo plane in FSUIPCConnection.AITrafficServices.AllTraffic)
    {
        this.listBox1.Items.Add(plane.ATCIdentifier + " (" + plane.State.ToString() + ")");
    }
}
```

Visual Basic.NET

```
Private Sub readAI()
    ' Get the AI Traffic info back from FSUIPC
    ' This happens immediatly, no need to call Process()
    FSUIPCConnection.AITrafficServices.RefreshAITrafficInformation()
    ' The following code iterates through the list and adds each plane's ATCid to a listbox
    ' along with it's state in brackets. (Landing, enroute etc..)
    this.listBox1.Items.Clear()
    For Each Plane As AIPlaneInfo In FSUIPCConnection.AITrafficServices.AllTraffic
        Me.listBox1.Items.Add(Plane.ATCIdentifier & " (" & Plane.State.ToString() & ")")
    Next Plane
End Sub
```

过滤AI交通

在刷新交通信息后你可以选择性地应用一个过滤器来删除你不需要的飞机。你可以过滤掉不在某高度范围内、不在某距离范围内或者不在某航向范围内的飞机。

你也可以选择过滤空中或地面的或两者一起。

要应用过滤器，你要在每次`RefreshAITrafficInformation()`调用后调用ApplyFilter()方法。要知道需要传递哪些参数到这个方法，查看Intellisense或参考手册。

距离过滤已经被动态链接库处理了，FSUIPC也可以通过INI文件控制距离过滤。这里的过滤器不能控制FSUIPC过滤器。如果FSUIPC过滤器设定为30海里内，那就是30海里了，即使你将`ApplyFilter`的`WithinDistance`参数设定为40海里也没用。

FSUIPC过滤器默认设置为40海里了。要通过这个动态链接库覆盖INI设置，查看后面的章节：「重写FSUIPC中的AI交通设置」。

下面的例子应用了一个空中AI交通过滤器。它过滤掉10000尺以上、30海里以外的飞机。

C#

```
FSUIPCConnection.AITrafficServices.ApplyFilter(false, true, 0, 360, null, 10000d, 30d);
```

Visual Basic.NET

```
FSUIPCConnection.AITrafficServices.ApplyFilter(False, True, 0, 360, Nothing, 10000D, 30D)
```

获取额外的识别信息

`AIPlaneInfo`类的下列属性默认不会填写。

```
TailNumber
AirlineAndFlightNumber
AircraftTypeAndModel
AircraftTitle
```

理由是，这些信息从FSUIPC取得需要很长的时间（每架飞机1秒）。这些信息几乎永远不会变化，所以性能决定每架飞机只需获得一次即可。

这里有两种方法获取这些信息。

e 你可以使用AIPlaneInfo类的GetExtendedPlaneIndentifiers()方法。可以取得给定飞机的某个指定字段。

f 你可以使用AITrafficServices类的UpdateExtenedPlaneIndentifiers()方法。可以取得所有飞机的某个指定字段。当一架新飞机被发现，将获取字段。请求可能造成1秒或更久的延迟，取决于你请求的字段。这发生一次就够了。

主要的问题将发生在第一次RefreshAITrafficInformation()调用时，也就是需要飞机的这些信息被填写的时候。如果你的用户的AI交通复杂（比如60架飞机），你的应用请求这四个字段，可能Refresh()调用的耗时会超过一分钟。

请小心地使用这些信息。

通过更改AI飞机的ATC标识，可以包含一些额外的信息。这不会对性能造成多大影响，然而需要注意如下两点：

.N 在FSUIPC中ATC标识符只有15字符长，所以信息可能会被截断。（不过FSUIPC文档说信息被截断极少发生）

.O 其他使用FSUIPC的程序可能会以默认格式请求ATC标识符。这意味着如果你更它，其他程序可能会停止运行。

要更改ATC标识符，请参看下面的「重写FSUIPC.INI中的AI交通设置」章节。

寻找可用跑道

这个动态链接库可以为你提供某机场可用的跑道列表。这个机场必须存在活动的AI交通。这些信息仅能在AI交通信息中提供。要得到实时的跑道信息，你必须先调用RefreshAITrafficInformation()。

某个机场的AI交通越多，可能可用的跑道信息就越多。

动态链接库内部自动获取这些数据它不使用D000内存的内容因为这需要时间从FSUIPC获取内容。动态链接库可以实时地向你提供。然而，不像FSUIPC的功能，这个动态链接库只能从FSUIPC.INI文件设置的范围内的飞机处获取信息。

使用AITrafficServices类提供的这两个方法：

```
GetDepartureRunwaysInUse(AirportICAOCode)
GetArrivalRunwaysInUse(AirportICAOCode)
```

你需要传递你所需的机场的ICAO代码。这个方法将返回给你可以使用foreach循环的FSRunway对象列表。FSRunway类有一些属性告诉你跑道编号和位置（左、中、右、水上）。有一个方便的ToString()方法提供给你代表跑道编号和位置的字符串。

下面的例子中，将EGLL的可用起降跑道显示在ListBox中。

C#

```
private void getActiveRunways()
{
    // Get a reference to the AITrafficServices class (saves typing)
    AITrafficServices AI = FSUIPCConnection.AITrafficServices;
    // Refresh the traffic information
    AI.RefreshAITrafficInformation();
    // Get the arrival runways in use
    List<FSRunway> runways = AI.GetArrivalRunwaysInUse("EGLL");
    // Display in the listbox
    this.lstArrival.Items.Clear();
    foreach (FSRunway rw in runways)
    {
        this.lstArrival.Items.Add(rw.ToString());
    }
    // same for departure runways
    runways = AI.GetDepartureRunwaysInUse("EGLL");
    this.lstDeparture.Items.Clear();
    foreach (FSRunway rw in runways)
    {
        this.lstDeparture.Items.Add(rw.ToString());
    }
}
```

Visual Basic.NET

```
Private Sub getActiveRunways()
    ' Get a reference to the AITrafficServices class (saves typing)
    Dim AI As AITrafficServices = FSUIPCConnection.AITrafficServices
    ' Refresh the traffic information
    AI.RefreshAITrafficInformation()
    ' Get the arrival runways in use
    Dim runways As List(Of FSRunway) = AI.GetArrivalRunwaysInUse("EGLL")
    ' Display in the listbox
    Me.lstArrival.Items.Clear()
    For Each rw As FSRunway In runways
        Me.lstArrival.Items.Add(rw.ToString())
    Next rw
    ' same for departure runways
    runways = AI.GetDepartureRunwaysInUse("EGLL")
    Me.lstDeparture.Items.Clear()
    For Each rw As FSRunway In runways
        Me.lstDeparture.Items.Add(rw.ToString())
    Next rw
End Sub
```

重写FSUIPC.INI中的AI交通设置

FSUIPC.INI文件包含了FSUIPC如何报告AI交通的设置。通常这可以让用户依照自己的意愿设置，你也可以重写这些设置。这并不会变更INI文件，它只是在你的应用运行时改变FSUIPC回应的方式。

下面的设置可以被重写：

- 空中AI交通的距离限制
- 当玩家在空中时地面的AI交通距离限制
- 当玩家在地面时空中的AI交通距离限制
- ATC标识符的格式。它可使用以下内容：
  - 注册号
  - 航班号
  - 飞机种类（Aircraft Type）
  - 飞机名称（Aircraft Title）
  - 飞机种类加注册号后三位
  - 飞机模型（Aircraft Model）

使用AITrafficServices中下面两种方法之一来变更INI设置。

```
OverrideAirborneTrafficINISettings
OverrideGroundTrafficINISettings
```

注意20秒后这些变更会被FSUIPC注销。要保证一直可用，20秒结束前必须再次调用这些方法。FSUIPC文档推荐每5秒一次。

向FSUIPC表写入AI交通（TCAS）信息

这个动态链接库还有一个向FSUIPC的AI交通表写入数据的功能。这不会在FS内创建AI飞机。事实上FS永远不会知道关于这些被创建的飞机的任何事情。

这个功能的主要用途是给其他使用FSUIPC的应用（或仪表）读取。最常见的情况是TCAS雷达/显示，也有可能是其他像进离场板类的应用。所以不要认为是动态链接库创建了一个真的AI交通，这指的是创建一个TCAS目标。

写入TCAS目标通过调用AITrafficServices类中的两个方法分两步完成。首先调用AddTCASTarget()方法来定义新飞机的信息。这个方法有很多参数。如果你不知道这些信息

可以将其值设定为0。比如你不知道升降率你可以发送0。FSUIPC对于大部分的值都不会做出任何处理，只是将它们发送到请求AI交通信息的应用中。

显然你写入的信息一定要对预定的接受方有所用处。比如你向一个TCAS仪表写入目标，你至少需要经纬度和高度。

FSUIPC需要的重要的参数是ID和ATC标识符。关于这个方法和它的参数的更多信息，参看参考手册或者Intellisense。

每次你需要写入飞机时，调用一次`AddTCASTarget()`。

你要将所有你需要的飞机写入时，第二步就是调用SendTCASTargets()将数据发送到FSUIPC。不需要调用Process()。

注意FSUIPC会在8至12秒后擦除这些飞机。也就是说在某段时间后需要重新发送数据（可能是至少5秒，当然也可以更频繁）。

每次你发送数据，必须要通过`AddTCASTarget()`和`SendTCASTargets()`这两步发送数据。动态链接库在上一次`SendTCASTargets()`调用后不会保存TCAS目标。