Traduction d'un chapitre du livre ***Building Recreational Flight Simulators****, édité par Mike's Flight Deck Books.*

www.mikesflightdeckbooks.com. Traduction par Jean-Paul Corbier, en juillet 2011.

L'interfaçage avec MSFS

Un simulateur de vol n'est qu'une pièce de musée inerte tant qu'elle a pas été amenée à la vie par une application de simulation effective. Ceci exige une connexion de haute qualité entre l'application et les divers systèmes qui composent votre simulateur de vol. Cette connexion a des composants logiciels et des composants matériels. Ce chapitre traite de la partie logicielle.

Déplacer les données dans les deux sens vers les applications de simulation de vol a été le plus gros travail dans la construction d'un simulateur de vol. Ce travail a été rendu beaucoup plus facile grâce à la mise sur le marché de MSFX. FSX Deluxe est livré avec une API, interface de programmation, appelée SimConnect. Parmi ses fonctions, SimConnect peut lire et écrire des données de FS.

L'essentiel de ce chapitre est dédié à SimConnect. Nous commencerons avec un survol des processus de lecture et d'écriture et des appels aux fonctions de SimConnect qui les permettent. Il y a une section détaillée qui explique comment mettre en œuvre l'édition gratuite de Microsoft Visual C++ express dans le développement d'applications d'entrée/sortie de SimConnect. Il y a deux exemples de programmes qui présentent la lecture et l'écriture de données de FS. Les deux programmes utilisent le port série com et fonctionne avec les projets matériels décrits dans les autres chapitres. Il y a une couverture brève d'autres options pour interfacer Flight Simulator sans utiliser SimConnect. Finalement, il y a un survol d'autres applications de simulation de vol qui ont des possibilités de programmation d'interfaces.

L'interface avec Flight Simulator

Le lancement de Flight Simulator X a été un événement important pour les amateurs de simulation de vol. FSX est la première version à intégrer l'API (*application program interface*) d'usage général appelée SimConnect . Bien que les amateurs ont été capables d'utiliser les API pour les tableaux de bord et les instruments et divers programmes d'autres fournisseurs, ces approches n'ont pas eu la versatilité ni le support officiel dont jouit SimConnect. Si vous voulez construire un simulateur de vol domestique basé sur FSX, alors SimConnect est la voie à suivre.

SIMCONNECT

SimConnect est un système de messagerie piloté par les événements qui supporte les communications dans les deux sens et vous permet d'interagir avec la plupart des aspects de FSX. L'API de SimConnect comprend plusieurs douzaines de fonctions. Par chance, nous avons seulement besoin d'une petite poignée de ces fonctions pour lire et écrire les données utilisées par FSX dans un simulateur domestique.

Un survol de la lecture des données

Il y a quatre étapes pour extraire des données de FSX :

1. établir une connexion avec SimConnect.
2. dire à SimConnect de quelles données on a besoin.
3. dire à SimConnect quand collecter les données et où les envoyer.
4. dire à SIMCONNECT d'envoyer effectivement les données.

A la lecture d'un exemple de programme qui utilise SIMCONNECT, il est facile de perdre la trace de ce qui se passe. Garder les étapes ci-dessus à l'esprit aide à s'y retrouver.

Etape 1 : connexion à SIMCONNECT

Avant que vous ne puissiez commencer une conversation sensée avec SIMCONNECT, vous devez attirer son attention. Vous faites ceci en appelant la fonction `SimConnect_Open()`. Cela établit une connexion et détermine comment, et si, votre programme sera informé de la disponibilité de données.

Avant d'appeler cette fonction vous devez créer une poignée vide (*empty Handle*) associée avec la connexion SIMCONNECT. Une poignée est une espèce particulière de pointeur que vous utiliserez pour faire référence à la connexion quand vous ferez les prochains appels aux fonctions SIMCONNECT. Quand vous appelez `SimConnect_Open()`, vous lui passez un pointeur vers cette poignée. SIMCONNECT charge la poignée avec un numéro de connexion valide si l'appel à la fonction a réussi ou bien un code d'erreur si elle échoue.

Vous avez le choix de configurer SIMCONNECT pour informer de manière active votre programme de certaines conditions comme par exemple la disponibilité de certaines données. SIMCONNECT peut signaler ceci en utilisant le système de messagerie Windows ou bien en créant un événement. Si vous choisissez d'utiliser le système de messagerie, vous devez fournir un code de message et la poignée vers une fenêtre pour recevoir ce code. Le code de message est simplement un nombre entier de votre choix. Si vous décidez d'utiliser un événement pour le signalement, vous devez fournir une poignée vers l'objet événement. Vous n'êtes pas tenu d'utiliser l'une ou l'autre de ces options. Vous pouvez simplement contrôler périodiquement avec SIMCONNECT si quelque chose d'intéressant est arrivé. Je reviendrai plus longuement là-dessus à l'étape quatre.

```
HRESULT SimConnect_Open(
   HANDLE* phSimConnect,  //pointeur vers une poignée qui identifie votre connexion.
   LPCSTR szName,      // nom de votre programme
   HWND hWnd,          // poignée vers votre fenêtre
   DWORD UserEventWin32,  // message à envoyer à votre fenêtre quand la donnée est disponible
   HANDLE hEventHandle,   // poignée de l'événement à établir quand la donnée est disponible
   DWORD ConfigIndex      // index dans votre fichier SimConnect.cfg
);
```

Étape 2 : sélection des variables de simulation spécifiques dont vous avez besoin

Vous devez définir un groupe de variables de simulation que SIMCONNECT aura à rapporter. Vous faites ceci en utilisant la fonction `SimConnect_AddToDataDefinition()`. Chaque appel à cette fonction ajoute une variable à la définition du groupe. Les noms de variables de simulation, les unités des variables et les types utilisés dans cette fonction sont listés dans le SDK de SIMCONNECT.

Vous pouvez définir plus d'un groupe. Chaque groupe a un identificateur unique que vous devez fournir. C'est simplement un index. Typiquement, vous utiliserez une énumération pour associer chaque index avec un texte significatif.

Vous avez le choix d'avoir les données de simulation qui vous soient rapportées périodiquement ou bien seulement en cas d'un changement minimum de leur valeur. Vous configurez cette option en entrant une valeur non nulle dans le paramètre `fEpsilon` et en positionnant des *flags* dans la fonction `SimConnect _RequestDataOnSimObject()`, présentée à l'étape trois.

```
HRESULT SimConnect_AddToDataDefinition(
   HANDLE hSimConnect,      // poignée vers votre connexion SimConnect
   SIMCONNECT_DATA_DEFINITION_ID DefineID,  // identificateur du groupe de données
```

```
   const char* DatumName,       // nom de la variable de simulation à ajouter au groupe
   const char* UnitsName,       // unité à utiliser pour la variable
   SIMCONNECT_DATATYPE DatumType,   // type de variable pour la donnée
   float fEpsilon,          // variation minimale de la variable avant rapport
   DWORD DatumID         // tag à utiliser si le rapport est en format Tagged
);
```

Étape 3 : démarrage de la réception des données

Utiliser la fonction `SimConnect_RequestDataOnSimObject()` quand vous être prêt à faire à rapporter les données par SIMCONNECT. Cet appel à la fonction fixera aussi la fréquence à laquelle les structures de données sont transmises. Vous pouvez faire en sorte que les données soit envoyées une seule fois ou bien synchroniser la transmission des données avec le *frame rate* de la simulation, le *visual frame rate* de la simulation ou encore une fois par seconde. Vous pouvez encore configurer SIMCONNECT pour sauter un certain nombre de périodes entre les transmissions. Par exemple, si vous avez besoin de données pour les instruments de vol 10 fois par seconde et que vous avez 30 FPS, vous pouvez synchroniser la transmission des données sur le *visual frame rate*, et sauter deux *visual frames* après chaque transmission de données.

Cette fonction reçoit aussi une paire de *flags* comme paramètres pour vous permettre de sauter des transmissions s'il n'y a pas de changement significatif dans les données. Vous pouvez vous arranger pour avoir toute la structure de données transmise seulement si une ou plus de ses variables change d'une certaine valeur `fEpsilon`. Par ailleurs vous pouvez configurer SIMCONNECT pour envoyer seulement la variable qui a changé. Si vous sélectionnez cette option, la donnée sera envoyée en format *Tagged* et en utilisant le *tag* que vous avez fourni quand vous avez établi la définition des données à l'étape précédente.

Configurer SIMCONNECT pour un envoi des données de manière périodique convient pour des variables qui changent continuellement, comme l'incidence et le roulis d'un avion. Les rapporter après un changement minimum est utile quand on suit une variable qui change lentement ou peu fréquemment comme la quantité de carburant.

```
HRESULT SimConnect_RequestDataOnSimObject(
   HANDLE SimConnect ,          // poignée de connexion
   SIMCONNECT_DATA_REQUEST_ID RequestID,   // identificateur de la demande de données
   SIMREQUEST_DATA_DEFINITION_ID DefineID, // le groupe de données demandé
   SIMCONNECT_OBJECT_ID ObjectID,      // de quoi traite la donnée
   SIMCONNECT_PERIOD Period,    // fréquence de collection des données
   SIMCONNECT_DATA_REQUEST_FLAG Flags, // options
   DWORD origin,            // quand démarrer la transmission des données
   DWORD interval,              // combien de périodes à sauter
   DWORD limit           // quand arrêter la transmission des données
) ;
```

Étape quatre : la demande des structures de données

SIMCONNECT ne délivre pas de données à votre programme tant que vous ne lui en demandez pas spécifiquement. SIMCONNECT vous enverra les données, mais elles seront tamponnées (*buffered*) jusqu'à ce que votre programme les reçoive réellement. Ceci vous permet de synchroniser l'arrivée des données avec leur traitement. Cela évite d'avoir de données en entrée qui changent mystérieusement quand vous êtes seulement à mi-chemin du travail à faire avec elles, une situation qui est rarement considérée comme agréable.

Une fois que vous appelez avez appelé SIMCONNECT `SimConnect_RequestDataOnSimObject`, SIMCONNECT construit la structure de données et la place dans une file d'attente de messages pour vous. Vous pouvez extraire cette structure de la file d'attente directement en appelant la fonction `SimConnect_GetNextDispatch()`, ou bien vous

pouvez faire en sorte que SIMCONNECT vous envoie les structures par une fonction en réponse à votre appel grâce à la fonction `SimConnect_CallDispatch()`.

Vous avez trois façons de déterminer quand une structure de données est disponible. La première est demander les données jusqu'à ce que vous les obteniez. Dans le jargon cela s'appelle du *polling* (scrutation). C'est du gaspillage de temps mais c'est facile à mettre en œuvre.

La plupart des exemples dans le SDK utilisèrent le *polling*. C'est réalisé par une boucle courte qui contient seulement un appel à `SimConnect_CallDispatch()` suivi d'une instruction `Sleep(1)`. Quand le système (l'*operating system*) accorde à cette portion de code sa tranche de temps, un appel à la fonction `SimConnect_CallDispatch()` est fait et le code libère le reste de la tranche de temps. Le paramètre `(1)` de `Sleep(1)` demande au système de ne pas le réveiller s'il reste moins d'une milliseconde à attendre. Si vous avez réglé SIMCONNECT pour envoyer des données à la fin de chaque *visual frame* et que vous avez 20 *frames* par seconde, vous faites 50 appels pour chaque structure de données envoyée.

Ceci signifie que vous faites 49 appels inutiles, et ceci gaspille des ressources du système, qui autrement pourrait atteindre des performances extrêmement élevées. En réalité, ce n'est pas vrai. Il n'y a pas dans ce cas un si grand gaspillage. Vous ne faites pas faire beaucoup de choses à SIMCONNECT. Vous avez simplement levé un *flag* qui dit à SIMCONNECT que vous êtes prêt à recevoir des données. Et encore, cette action se traduit pas un échange de contextes, et si vous avez beaucoup de connexions séparées actives vers SIMCONNECT, alors seulement vous pourrez commencer à avoir de sérieux gaspillages.

Les alternatives au *polling* sont que SIMCONNECT alerte votre programme grâce au système de messagerie de Windows ou bien que SIMCONNECT crée un événement.

Quand vous ouvrez initialement la connexion vers SIMCONNECT, vous avez l'option de lui passer un code de messages et une poignée vers une fenêtre. Si vous utilisez cette option, SIMCONNECT enverra le code de message à la fonction `winproc()` qui accompagne la fenêtre pointée par la poignée quand une structure de données devient disponible. Une fois que vous êtes notifié de cette disponibilité par ce message, vous demandez la donnée. Bien sûr, vous devez avoir une fenêtre pour utiliser le système de messages. Vous pouvez créer une fenêtre de base dans ce but, mais si vous n'utilisez pas une fenêtre par ailleurs, vous pouvez au contraire utiliser un événement à la place du système de messagerie de windows.

Un événement est un *flag* qui est levé par le programme et entretenu par le système. On y accède à travers une poignée et on peut le lire, l'écrire et le mettre à zéro. Il est utilisé pour synchroniser l'exécution entre les différents *threads* ou programmes. Vous créez un événement en utilisant l'appel à `CreateEvent()`. Si vous passez la poignée de l'événement à SIMCONNECT quand vous ouvrez pour la première fois la connexion à SIMCONNECT, SIMCONNECT créera l'événement quand la donnée sera disponible. Vous pouvez utiliser la fonction `WaitForSingleObject()` dans votre programme pour synchroniser son exécution avec la disponibilité de cette donnée.

Le choix de la meilleure des trois options dépend de circonstances particulières. Si vous avez déjà une structure de programmes qui utilise la messagerie de Windows, alors ajouter un nouveau message de SIMCONNECT semble un choix évident. Si vous avez un programme plus simple qui traite très fréquemment des structures de données, la surcharge ajoutée par le *polling* est faible et probablement tout à fait tolérable. Si vous utilisez la messagerie de Windows et que les données arrivent de manière peu fréquente, ou si vous voulez diminuer le temps de latence et la surcharge, alors le meilleur choix est d'utiliser un événement pour signaler la disponibilité de vos données.

`SimConnect_CallDispatch(`

```
   HANDLE hSimConnect,       // poignée vers votre connexion à SIMCONNECT
   DispatchProc pfcnDispatch,   // nom de votre fonction de rappel de traitement de donnéesVoid * pContext,
       // pointeur que SIMCONNECT retourne à la fonction d'appel
) ;

SimConnect_GetNextDispatch(
   HANDLE hSimConnect,       // poignée vers votre connexion à SIMCONNECT
   SIMCONNECT_Recv** pData,  // pointeur vers un pointeur de la structure de données
   DWORD pcbData         // pointeur vers la taille de la structure de données
) ;
```

Un survol de l'écriture des données

Il y a trois étapes clés pour écrire des données dans FSX :

1. établir une connexion avec SIMCONNECT.
2. donner à SIMCONNECT le nom des variables qui devront être écrites.
3. dire à SIMCONNECT d'écrire les données.

Puisque vous avez lu ce qui concerne la lecture des données, vous avez déjà une idée de SIMCONNECT. Écrire est un peu plus simple que lire. Il y a seulement trois étapes, et vous en avez déjà couvert une. Néanmoins, ce survol va rendre la lecture de l'exemple de code un peu plus facile.

Étape 1 : connexion à SIMCONNECT

La connexion à SIMCONNECT pour écrire des données est exactement la même que pour lire des données. En fait, la chance est que vous utiliserez la même connexion pour à la fois lire et écrire.

Étape 2 : dire SIMCONNECT quoi attendre

La plupart des variables de simulation sont écrites en notifiant SIMCONNECT d'un événement du client. Ceci apparaît comme une suite des « *key events* » utilisés dans le SDK des tableaux de bord et des instruments. Les entrées du clavier et les clics de la souris sont des événements qui déclenchent une réponse du simulateur. Bien que se débarrasser des claviers soit un des buts majeurs des constructeurs de simulateur domestique, nous continuerons à utiliser les *key events* de l'interface logicielle pour changer des variables dans FSX. Les variables qui ne peuvent pas être changées de cette manière sont listées dans la partie du SDK consacrée aux événements.

Vous pouvez écrire directement à seulement un petit nombre des variables listées dans les tables « *Simulation Variables* » du SDK. Ce sont les tables que vous utilisez pour trouver les noms de variables pour la lecture des données. Si la variable n'a pas un « *Y* » dans la dernière colonne des tables (colonne « *Settable* »), vous ne pouvez pas les écrire directement.

Dans les deux cas il est nécessaire d'abord d'informer SIMCONNECT de quelle variable va être écrite. Pour les variables listées dans les tables d'événements, vous faites ceci en établissant une liste d'identifiants uniques des événements. Le meilleur moyen de le faire est avec une énumération. L'énumération vous permet d'associer une chaîne de caractères significative avec ce qui est juste un index (nombre entier). Avec votre identificateur d'événements, vous pouvez faire un appel à `SimConnect_Map_ClientEvent()` pour chaque variable que vous prévoyez d'écrire.

Pour les quelques variables listées dans les tables des variables de simulation du SDK, vous construisez des tables de définition de variables exactement comme vous avez fait pour lire ces variables avec SIMCONNECT. Cependant comme vous n'écrirez généralement qu'une variable à la fois, la plupart de vos groupes de définition de variables n'aura qu'un seul membre.

```
SimConnect_Map_ClientEvent(
   HANDLE hSimConnect,      // poignée vers votre connexion SIMCONNECT
   SIMCONNECT_CLIENT_EVENT_ID EventID, // votre nom pour votre événement
   Const char* EventName,   // nom de la variable de simulation du SDK
) ;

SimConnect_AddToDataDefinition(
   HANDLE hSimConnect,      // poignée vers votre connexion SIMCONNECT
   SIMCONNECT_DATA_DEFINITION_ID DefineID, // identificateur du groupe de données
   const char* DatumName,       // nom de la variable à ajouter au groupe
   const char* UnitsName,       // unités à utiliser avec la variable
   SIMCONNECT_DATATYPE DatumType, // type de variable pour la donnée
   float fEpsilon,          // variation minimale de la variable avant rapport
   DWORD DatumID         // tag à utiliser si vous utilisez un format tagged
) ;
```

Étape 3 : demander l'écriture des données

L'écriture est directe. Appelez `SimConnect_TransmitClientEvent()` pour les variables dans la liste des identificateurs d'événements. Appelez `SimConnect_SetDataOnSimObject()` si la variable est une des quelques variables listées dans les variables de simulation.

Un message d'événements clients est seulement envoyé aux clients (y-inclus SFX) qui ont une priorité plus faible que la priorité du groupe qui est mentionnée dans l'appel à `SimConnect_TransmitClientEvent()`. Vous assurez que l'appel de votre événement est activé à en mettant la priorité haute.

Une partie délicate de cette étape est de vous assurer que vous passez les données avec le format correct. Le SDK est utile mais il n'est pas toujours strictement précis. Par exemple le format de donnée pour régler les fréquences de navigation NAVCOM est donné comme « BCD HZ », décimal codé binaire en Hertz. Pour une fréquence de 121,5 MHz, ceci donnerait « `0X121500000` », (où 0X indique un nombre en hexadécimal). En réalité, la valeur correcte est « `0X12150` ». Le descripteur correct du format devrait être « BCD settable digits ».

```
SimConnect_TransmitClientEvent(
   HANDLE hSimConnect,      // poignée vers votre connexion SIMCONNECT
   SIMCONNECT_OBJECT_ID,    // l'object auquel la donnée se réfère
   SIMCONNECT_CLIENT_EVENT_ID EventID, // votre nom pour votre événement
   DWORD dwData          // valeur à écrire dans la variable
   SIMCONNECT_NOTIFICATION_GROUP_ID GroupID, // groupe de notification
) ;

SimConnect_SetDataOnSimObject(
   HANDLE hSimConnect,      // poignée vers votre connexion SIMCONNECT
   SIMCONNECT_DATA_DEFINITION_ID DefineID, // identificateur du groupe de données
   SIMCONNECT_OBJECT_ID,    // l'object auquel la donnée se réfère
   SIMCONNECT_DATA_SET_FLAGS Flags, // utilisé pour indiquer un format tagged
   DWORD ArrayCount,        // nombre d'éléments de données
   DWORD cbUnitSize,        // nombre d'octets dans l'élément de données
   void * pDataSet          // pointeur vers la donnée
) ;
```

Mise en place de l'environnement de développement

SIMCONNECT est livré avec la version deluxe de FSX. Cependant il ne s'installe pas tout seul quand vous installez FSX. Vous devez utiliser l'explorateur Windows pour retrouver l'installateur de SIMCONNECT sur le DVD et le lancer manuellement. Une fois que vous l'avez installé, vérifiez les possibles mises à jour sur le site Internet Microsoft FS Insider (www.fsinsider.com).

SIMCONNECT peut être appelé depuis une variété de langages incluant C, C++, C# et Visual Basic. Ce ne sont pas nécessairement des produits de Microsoft. Les développeurs ont utilisé avec succès Delphi et Borland Turbo C++. Comme l'essentiel des échantillons de code fournis avec la documentation de SIMCONNECT est en C++, et comme SIMCONNECT est un produit de Microsoft, je resterai avec Microsoft C++ pour la discussion qui suit.

La documentation de SIMCONNECT indique spécifiquement que vous pouvez utiliser Microsoft Visual C++ 2005 Express pour construire des *add-ons* SIMCONNECT. Ce qui n'est pas mentionné c'est que cette édition Express toute seule n'est pas suffisante. L'édition de 2005 est configurée pour développer des applications .NET, mais il lui manque les bibliothèques et les en-têtes pour développer des applications Windows. L'utilisation de C++ 2005 Express exige de télécharger et d'installer la plate-forme SDK de Windows.

Par chance, Visual C++ 2008 Express inclut les bibliothèques et en-têtes de Win32. Cette version fonctionne bien avec SIMCONNECT. Donc la chose la plus simple à faire est de télécharger et installer cette version.

Lancer les exemples de code de SIMCONNECT

La documentation du SDK de Fligh Simulator inclut de nombreux exemples de code qui illustrent les méthodes de SIMCONNECT. Les faire tourner aide à explorer leurs fonctionnalités, et sert de test pour la configuration de votre système. En particulier, compiler et faire tourner les exemples « `openandclose` » vous assure d'avoir un environnement de développement correctement mis en place et que le SIMCONNECT fonctionne bien avec votre copie de FSX.

NDT : ce qui suit ne s'invente pas ! A suivre scrupuleusement.

1. Recherchez le fichier `simconnect.ini`. Il doit se trouver dans ces `C:\Program Files\Microsoft Games\Microsoft Flight Simulator X SDK\SDK\Core Utilitities Kit\SimConnect SDK\config` si vous avez utilisé les fichiers par défaut de SDK. Placez une copie de ce fichier dans votre de dossiers `My Documents\Flight Simulator X`. SIMCONNECT fonctionnera sans ce fichier, cependant sa présence permettra à SIMCONNECT d'ouvrir une fenêtre de diagnostic très utile.

2. Démarrez C++.

3. Sélectionnez **Fichier\Nouveau\Projet**. Ceci ouvre la fenêtre de dialogue de nouveaux projets.

4. Sélectionnez « Application Console Win32», et entrez « `openandtest` » comme nom en bas de la fenêtre. Il doit y avoir une marque cochée juste à côté de « Créer un répertoire pour la solution ».

5. Cliquez sur **ok**. Ceci ouvre la fenêtre de l'assistant Application Win32.

6. Cliquez sur « **Suivant>** ». Ceci accède aux paramètres de l'application. Il faut sélectionner **Application Console**. Cochez l'option **Projet vide**.

7. Cliquez sur **Terminer**. Visual Studio a mis en place l'espace de travail pour votre projet de test.

8. Comme le programme de test que vous allez créer devra communiquer avec SIMCONNECT, le *linker* de Visual Studio C++ doit être informé de l'endroit où se trouve la bibliothèque SIMCONNECT. Cliquez sur **Projet** dans la barre d'outils de Visual Studio et sélectionnez **Propriétés** pour ouvrir la boîte de dialogue des pages de **Propriétés**. Dans la partie gauche de la boîte de dialogue, déployez **Propriétés de configuration**, puis **Editeur de liens**, et sélectionnez **Général**. Dans la fenêtre de droite, sélectionnez la zone de texte à droite de **Répertoires de bibliothèques supplémentaires**. Cliquez sur le petit bouton qui apparaît à la droite de la zone de texte. Ceci ouvrira la boîte de dialogue de **Répertoires de bibliothèques supplémentaires**. Cl sur l'icône de gauche en haut à gauche. Vous accédez à une ligne avec un bouton marqué "…" tout à gauche. Il donne accès à une boîte de sélection de répertoire. Explorer jusqu'à trouver l'endroit du dossier contenant `simconnect.lib`. Si vous avez utilisé l'emplacement d'installation par défaut, ce sera : `C:\Program Files\Microsoft Games\Microsoft Flight Simulator X SDK\SDK\Core Utilitities Kit\SimConnect SDK\lib`. La fenêtre d'exploration va se fermer et le chemin vers `simconnect.lib` se retrouvera dans la boîte de dialogue des répertoires de bibliothèques additionnelles. Cliquez sur **ok**.

9. Vous devez aussi dire à Visual Studio d'utiliser la bibliothèque SIMCONNECT. Dans la boîte de dialogue des pages de proprpriétés qui est encore ouverte, sélectionnez **Ligne de commande**. Sous **Options additionnelles** ajoutez le nom de fichier `simconnect.lib`. Cliquez sur **ok**.

10. Cliquez avec le bouton droit dans la fenêtre d'exploration de solution de Visual Studio. Sélectionnez **Ajoutez\Nouvel élément**. Ceci ouvre une boîte de dialogue **Ajouter un nouvel élément**. Sélectionner **Code** dans les **Catégories** et ensuite sélectionnez **Fichier d'en-tête** (.h) dans les modèles installés. Tapez `simconnect. h` dans la boîte de nom et cliquez sur **Ajouter**.

11. En utilisant Notepad, ouvrir le fichier `simconnect.h` trouvé dans `C:\Program Files\Microsoft Games\Microsoft Flight Simulator X SDK\SDK\Core Utilitities Kit\SimConnect SDK\inc`. Copiez son contenu et collez-le dans la fenêtre `simconnect.h` que vous avez créée à l'étape précédente. Sauvegarder le fichier. A ce point, vous avez mis en place tout ce dont vous avez besoin pour réaliser un programme de développement d'un add-on avec SIMCONNECT. Pour être sûr que tout est correct, vous allez maintenant ajouter un programme de tests simples à partir de la documentation de SDK.

12. Cliquez avec le bouton droit dans la fenêtre de l'explorateur de solutions de Visual Studio. Ceci ouvre la boîte de dialogue **Ajouter un nouvel élément**. Sélectionnez **Fichier C++ (.cpp)** dans les modèles installés. Tapez « `opentest` » dans la boîte de nom et cliquez sur **Ajouter**.

13. Copiez les exemples de code de *open and close* depuis la documentation du SDK dans la fenêtre vide `opentest.cpp`. Sauvegardez le fichier. Vous pouvez aussi utiliser le programme présenté à la fin de ce document, qui est une variante d'un exemple du SDK, mais qui a été testé et qui marche.

14. Vous pouvez maintenant construire le projet et exécuter votre programme. Dans la barre d'outils de Visual Studio, cliquez sur construire la solution. Vous pouvez lancer le programme en sélectionnant **Debug** dans la barre d'outils et en cliquant sur **démarrer sans débug**.

Lecture des données : SimOuput

SimOutput est un exemple qui montre comment lire des données de FSX en utilisant SIMCONNECT. Il explique plusieurs choses :

- Ouvrir une connexion vers SIMCONNECT
- Utiliser la synchronisation d'objets événements partagés
- Construire des groupes d'événements

- Construire des groupes de définition de données
- Recevoir une notification d'événements
- Recevoir des données périodiques
- Recevoir des données seulement quand elles changent
- Mettre en place une connexion avec le port série
- Transmettre des données à travers le port série

Ce programme a été construit en utilisant l'édition Express de Microsoft C++.Net avec la plate-forme SDK de Windows installée. SimOutput demande des données à FSX à travers SIMCONNECT pour plusieurs variables, formatte les données reçues et les émet à travers le port série. Les données sont formattées pour piloter le moteur pas à pas de l'instrument VSI et l'instrument à LEDs qui est décrit dans le chapitre des instruments à faire soi-même.
Ceci est d'abord un programme de démonstration. Il inclut une série de contrôles d'erreurs qui est destinée à vous aider à localiser les problèmes quand vous faites des expériences avec SIMCONNECT et le port série. Si vous prévoyez d'utiliser des programmes d'entrées-sorties et de les partager, il faut prévoir de bien maîtriser ces conditions d'erreur auparavant.
NDT : ce programme a été testé et fonctionne.

Fichier stdafx.h à inclure au projet

```
// stdafx.h : fichier Include pour les fichiers Include système standard,
// ou les fichiers Include spécifiques aux projets qui sont utilisés fréquemment,
// et sont rarement modifiés

#pragma once

#include "targetver.h"
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <strsafe.h>
#include <conio.h>

// TODO: faites référence ici aux en-têtes supplémentaires nécessaires au programme
```

Fichier du programme SimOutput

```
//--------------------------------------------------------------------------------------
//
// SimOutput : exporte des données par le port série en utilisant SimConnect
//
// Après le chargement d'un vol, on envoie sur COM1 affiche l'altitude, l'incidence,
// l'inclinaison, la vitesse indiquée et la quantité de carburant, le tout formatté.
// En même temps, on affiche dans une fenêtre DOS les mêmes données.
//
// Le programme démarre effectivement au lancement de FSX. Donc si FSX est déjà en fonctionnement quand on
lance
// le programme, il faut simuler le chargement d'un vol. Pour ça, il faut aller dans Options, Paramètres,
Commandes,
// et faire OK. Ça démarre.
//
//--------------------------------------------------------------------------------------

#include "StdAfx.h"
#include "SimConnect.h"

int quit = 0;
HANDLE hSimConnect = NULL;

struct StreamedData
```

```
    {
    double vertspeed;
    double altitude;
    double pitch;
    double bank;
    double ias;
    };

struct ChangedData
    {
    double fuel_left;
    double fuel_right;
    };

static enum EVENT_ID
    {
        EVENT_SIM_START,
        EVENT_SIM_PAUSE
    };

static enum DATA_DEFINE_ID
    {
    STREAMING_DATA,
    CHANGING_DATA
    };

static enum DATA_REQUEST_ID
    {
    STREAM_REQ,
    CHANGED_REQ
    };

// prototype for callback function
void CALLBACK MyDispatchProcRD(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext);

int __cdecl _tmain(int argc, _TCHAR* argv[])
    {
    // Crée l'objet-événement Windows pour la synchronisation avec SimConnect
    // create the Windows event object for synchronizing with SimConnect
    HANDLE hEventHandle= ::CreateEvent(NULL,FALSE, FALSE, NULL);

    if (hEventHandle==NULL)
    {
        printf("Erreur : impossible de créer un objet événement");
    }

    HRESULT hr;

    // Ouvre une connexion avec SimConnect et lui passe la poignée pour l'objet-événement. "SimOutput" est
le nom du programme.
    // open a connection to SimConnect and pass it the handle to the event object
    if (SUCCEEDED(SimConnect_Open(&hSimConnect,"SimOutput", NULL, 0,hEventHandle,0)))
        {
        printf("\nConnecté à Flight Simulator.");

        // Définit les données pour les variables à mises à jour fréquentes
        hr = SimConnect_AddToDataDefinition(hSimConnect, STREAMING_DATA,"Vertical Speed", "feet");
        hr = SimConnect_AddToDataDefinition(hSimConnect, STREAMING_DATA,"Plane Altitude", "feet");
        hr = SimConnect_AddToDataDefinition(hSimConnect, STREAMING_DATA,"Attitude Indicator Pitch Degrees",
"degrees");
        hr = SimConnect_AddToDataDefinition(hSimConnect, STREAMING_DATA,"PLANE BANK DEGREES", "degrees");
        hr = SimConnect_AddToDataDefinition(hSimConnect, STREAMING_DATA,"AIRSPEED INDICATED", "knots");

        // Définit les données pour les variables à mises à jour peu fréquentes
        hr = SimConnect_AddToDataDefinition(hSimConnect, CHANGING_DATA,"Fuel Tank Right Main Quantity",
"gallons", SIMCONNECT_DATATYPE_FLOAT64,.01f);
```

```
        hr = SimConnect_AddToDataDefinition(hSimConnect, CHANGING_DATA,"Fuel Tank Left Main Quantity",
"gallons", SIMCONNECT_DATATYPE_FLOAT64,.01f);

        // Demande un événement quand la simulation démarre / Request an event when the simulation starts
        hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START,"SimStart");

        // Abonne le prog au système d'événements / Subscribe to system events
         hr = SimConnect_SubscribeToSystemEvent (hSimConnect,EVENT_SIM_PAUSE, "PAUSE");

         // démarre la notification de l'événement "en/hors pause" du système d'événements (OFF pour
l'arrêter)
         hr = SimConnect_SetSystemEventState(hSimConnect,EVENT_SIM_PAUSE, SIMCONNECT_STATE_ON);

        while( 0 == quit)
        {
            ::WaitForSingleObject(hEventHandle, INFINITE);              // This event synchronizes us with
            SimConnect_CallDispatch(hSimConnect, MyDispatchProcRD, NULL);   // SimConnect data availability
        }
        hr = SimConnect_Close(hSimConnect);
        printf("\nConnexion à Flight Simulator fermée.");      // ajouté par JPC 17/07/2011
        }
    else    // ajouté par JPC 17/07/2011
        {
            printf("\nEchec de la connexion à Flight Simulator.");
            printf("\nTapez une touche pour continuer.");
            char Touche;
            Touche = _getch();
        }
    return 0;
    }

void CALLBACK MyDispatchProcRD(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
    {
    HRESULT hr;
    DWORD ObjectID = 0;
    static HANDLE hComm;         // these must be static so we don't lose
    static DCB dcb;                 // access to the com port when the callback
    static COMMTIMEOUTS timeouts;   // function goes out of scope
    char byOutput[18];
    DWORD BytesWritten;
    DWORD dwLastErrorCode;
    int idigit1 = 0;
    int idigit2 = 0;
    int idigit3 = 0;
    int idigit4 = 0;
    int ifuel = 0;
    double dVSI = 0;
    int iVSI = 0;
    int iAltSteps = 0;
    bool EnPause = true;

    switch(pData->dwID)
    {
        case SIMCONNECT_RECV_ID_EVENT:
        {
            SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData;
            switch(evt->uEventID)
                {
                    case EVENT_SIM_PAUSE :
                        printf("\rEn pause");
                        break;
                    case EVENT_SIM_START : // sim has started, send data requests and open comp port
                        // this is the request for continuing data updates
                        hr=SimConnect_RequestDataOnSimObject(hSimConnect,STREAM_REQ,
                        STREAMING_DATA,
                        SIMCONNECT_OBJECT_ID_USER,
                        SIMCONNECT_PERIOD_VISUAL_FRAME,
```

```
                        0,0,0);
                        printf("\nDébut de la simulation");

                        //This requests data only when at least one data in the group changes
                        hr=SimConnect_RequestDataOnSimObject(hSimConnect,
                        CHANGED_REQ,
                        CHANGING_DATA,
                        SIMCONNECT_OBJECT_ID_USER,
                        SIMCONNECT_PERIOD_VISUAL_FRAME,
                        SIMCONNECT_DATA_REQUEST_FLAG_CHANGED,
                        0,14);

                        //Open the serail com port
                        hComm = CreateFile(TEXT("COM1:"), GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,
NULL);
                        if (hComm==INVALID_HANDLE_VALUE) //basic error check : did port open ?
                            {
                                MessageBox(NULL,TEXT("Impossible d'ouvrir Com Port 1"),TEXT("ERREUR"),0);
                            }
                        else
                        {
                            printf("\nPort Comm1 ouvert");    // ajouté par JPC 17/07/2011
                        }

                        if (!GetCommState(hComm,&dcb))    // establish reference to the device control block
                            {
                                // report error if we can't
                                MessageBox(NULL,TEXT("Impossible d'obtenir le Device Control block
courant"),TEXT("ERREUR"),0);
                                CloseHandle(hComm);
                            }
                        else
                            {
                                printf("\nDevice Control block OK");    // ajouté par JPC 17/07/2011
                            }
                        dcb.BaudRate= 38400;     // this is the device control Block structure
                        dcb.fParity = false;  // where we configure the COM port parameters
                        dcb.ByteSize = 8;
                        dcb.Parity = NOPARITY;
                        dcb.StopBits = TWOSTOPBITS;

                        if (!SetCommState(hComm,&dcb))  // try updating the device control block
                            {
                                // report error if unable
                                MessageBox(NULL,TEXT("Impossible de mettre à jour le Device Control
block"),TEXT("ERREUR"),0);
                                CloseHandle(hComm);
                            }
                        else
                            {
                                printf("\nDevice Control block mis à jour");    // ajouté par JPC 17/07/2011
                            }
                        timeouts.ReadIntervalTimeout = 50 ;          // the timeout structure determines
                        timeouts.ReadTotalTimeoutMultiplier = 50 ;   // how long a read or a write to the
                        timeouts.ReadTotalTimeoutConstant = 500 ;    // com port can hang in the calling
                        timeouts.WriteTotalTimeoutMultiplier = 10 ;  // function
                        timeouts.WriteTotalTimeoutConstant = 100 ;

                        if (!SetCommTimeouts(hComm,&timeouts))  // try updating the timeouts
                            {
                                // report error if unable
                                MessageBox(NULL,TEXT("Impossible de mettre à jour le
CommTimeouts"),TEXT("ERREUR"),0);
                                CloseHandle(hComm);
                            }
                        else
                            {
```

```
                            printf("\nComm TimeOuts mis à jour");    // ajouté par JPC 17/07/2011
                        }
                    break;
            default:
                break;
        }
        break;
    }

    case SIMCONNECT_RECV_ID_SIMOBJECT_DATA : // SimConnect has sent data
        {
            SIMCONNECT_RECV_SIMOBJECT_DATA *pObjData =(SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE*)pData;
            switch(pObjData->dwRequestID)
                {
                    case STREAM_REQ :   // "STREAM_REQ" is the ID we supplied to SimConnect to
                        {              // request continuing transmission of rapidly changing data
                            ObjectID = pObjData->dwObjectID;
                            StreamedData *pS=(StreamedData*)&pObjData->dwData;

                            printf("\rObjectID = %d VS=%.0f Alt=%.0f Pitch=%.1f Bank=%.1f IAS=%.1f",
                                 ObjectID,pS->vertspeed,pS->altitude,pS->pitch,pS->bank,pS->ias);

                            byOutput[0]='C';   //"CLST1" est l'en-tête des données à transmettre
                            byOutput[1]='L';   //aux instruments de vol exigeant
                            byOutput[2]='S';   //des mises à jour très fréquentes
                            byOutput[3]='T';   //
                            byOutput[4]='1';   //
                            byOutput[5]=(char)2; // 2 indique que nous faisons un "frame transfer"

                            dVSI =(pS->vertspeed)/.150;         // format vertical speed
                            if(dVSI>400.) dVSI=400;             // for stepping motor VSI
                            if(dVSI<-400.) dVSI=-400;           // project
                            iVSI = static_cast<int>(dVSI+400.); // NOTE : hasn't been
                            byOutput[6] = (char)(iVSI%256);     // mapped to the non linear
                            byOutput[7] = (char)(iVSI/256);     // scale markings

                            iAltSteps=(int)(pS->altitude * .8); // format altitude for a stepping
                            byOutput[8] = (char)(iAltSteps%256);   // motor based altimeter having
                            iAltSteps=iAltSteps/256;        // 800 steps = 1000 pieds
                            byOutput[9] = (char)(iAltSteps%256);
                            iAltSteps=iAltSteps/256;
                            byOutput[10] = (char)(iAltSteps%256);

                            byOutput[11] = (char)0;      // continue formatting for
                            byOutput[12] = (char)0;      // your particular instruments

                            // A WriteFile() to the com port handle sends the data to the com port
                            if(!WriteFile(hComm,byOutput,13,&BytesWritten,NULL))
                                {
                                    dwLastErrorCode = GetLastError();
                                    if (dwLastErrorCode != ERROR_IO_PENDING)  // did it work ?
                                        {
                                            MessageBox(NULL, TEXT("Erreur en écriture"), TEXT("ERREUR
D'ECRITURE"), MB_OK);
                                        }
                                }
                        break;
                        }
                    case CHANGED_REQ:    // "CHANGE_REQ" is the ID we supplied to
                        {               // SimConnect to request conditionnal transmission of slowly changing
data
                            ObjectID = pObjData->dwObjectID;
                            ChangedData *pS = (ChangedData*)&pObjData->dwData;
                            printf("\rObjectID = %d Left tank quantity=%.2f Right tank quantity=%.2f\n",
                                 ObjectID,pS->fuel_left,pS->fuel_right);
                            byOutput[0]='C';   //"CLST0" is the header of the instruments
                            byOutput[1]='L';   //that are updated infrequently
```

```
                        byOutput[2]='S';
                        byOutput[3]='T';
                        byOutput[4]='0';
                        byOutput[5]=(char)2;  // 2 indicates we are doing a frame transfer

                        byOutput[6]=(char)(((int)(pS->fuel_left))+128); // arc size and decimal
                        ifuel = (int)(100. * (pS->fuel_left));
                        idigit1=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit2=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit3=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit4=ifuel%10;
                        ifuel = ifuel / 10;
                        byOutput[8]=(char)(idigit1+16*idigit2);
                        byOutput[7]=(char)(idigit3+16*idigit4);

                        byOutput[9]=(char)(((int)(pS->fuel_right))+128); // arc size and decimal
                        ifuel = (int)(100. * (pS->fuel_right));
                        idigit1=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit2=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit3=ifuel%10;
                        ifuel = ifuel / 10;
                        idigit4=ifuel%10;
                        ifuel = ifuel / 10;
                        byOutput[10]=(char)(idigit1+16*idigit2);
                        byOutput[11]=(char)(idigit3+16*idigit4);

                        // write to the serial port, check for success or lack thereof
                        if(!WriteFile(hComm,byOutput,12,&BytesWritten,NULL))
                            {
                                dwLastErrorCode = GetLastError();
                                if (dwLastErrorCode != ERROR_IO_PENDING)  // did it work ?
                                    {
                                        MessageBox(NULL, TEXT("Erreur en écriture"), TEXT("ERREUR EN
ECRITURE"), MB_OK);
                                    }
                            }
                        break;
                    }
                case SIMCONNECT_RECV_ID_QUIT:  // sim has quit
                    {
                        quit = 1;
                        CloseHandle(hComm);
                        break;
                    }
                default:
                    printf("Received:%d,pData->dwID\r");
                    break;
            }
        }

    }
}
// ------------------------------------------------------------------------------
```

Ecriture des données dans FSX

SimInput est un exemple qui montre comment écrire des données de FSX en utilisant SIMCONNECT. Il explique comment :

- Associer des identificateurs d'événements avec des événements définis par l'utilisateur

- Construire des groupes de définition de données
- Ecrire en transmettant un événement client
- Ecrire en positionnant des données dans un objet simulé

Comme dans l'exemple précédent une partie du code sert à illustrer quelques points au sujet de la communication avec FSX à travers SIMCONNECT. Il effectue quelques des contrôles d'erreur, ce que votre code devra aussi faire en abondance.
NDT : ce programme a été testé et fonctionne.

Fichier du programme SimInput

```
//--------------------------------------------------------------------
//
// SimInput : exemple d'écriture de données dans FSX avec SimConnect
//
// Description : attend qu'un vol soit démarré, puis entre dans une boucle
// où les fréquences de COM1 et NAV1 et le code transpodeur sont changés
// et la manette des gaz déplacée. Ceci se fait en alternant les valeurs
// toutes les 3 secondes.
//--------------------------------------------------------------------

#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <strsafe.h>

#include "SimConnect.h"
bool bSimGoing = false ;
HANDLE hSimConnect = NULL ;

static enum EVENT_ID{
    EVENT_SIM_START,
    EVENT_SET_COM1_FREQ,
    EVENT_SET_NAV1_FREQ,
    EVENT_SET_XPONDER_CODE
} ;
static enum SET_DATA_DEFINITION_IDS{
    SET_THROTTLE_ID1,
    SET_THROTTLE_ID2
} ;

int quit = 0 ;

void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext);

int __cdecl _tmain()
{
    HRESULT hr ;
    DWORD Com1Freq = 0x12350 ;
    DWORD Nav1Freq = 0x11150 ;
    DWORD XPonderCode = 0x1200 ;
    double ThrottlePosition1 = 100. ;
    double ThrottlePosition2 = 100. ;
    bool bFlipFlop = true ;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect,"SimInput", NULL, 0,0,0)))
        {
        printf("\nConnecté à Flight Simulator.");
    // request a simulation started event
    hr = SimConnect_SubscribeToSystemEvent(hSimConnect, EVENT_SIM_START, "SimStart") ;

    // set up a data definition for throttle
    hr = SimConnect_AddToDataDefinition(hSimConnect, SET_THROTTLE_ID1,"GENERAL ENG THROTTLE LEVER
POSITION:1","percent") ;
```

```
    hr = SimConnect_AddToDataDefinition(hSimConnect, SET_THROTTLE_ID2,"GENERAL ENG THROTTLE LEVER
POSITION:2","percent") ;

    // map our events to simulator key events
    hr = SimConnect_MapClientEventToSimEvent(hSimConnect,EVENT_SET_COM1_FREQ,"COM_RADIO_SET") ;
    hr = SimConnect_MapClientEventToSimEvent(hSimConnect,EVENT_SET_NAV1_FREQ,"NAV1_RADIO_SET") ;
    hr = SimConnect_MapClientEventToSimEvent(hSimConnect,EVENT_SET_XPONDER_CODE,"XPNDR_SET") ;

    while (!bSimGoing)   // attend qu'un vol soit chargé (wait until a flight is loaded)
    // si FSX est déjà lancé, il faut simuler un démarrage en appelant le menu
    // Options, Paramètres, Commandes, et là, cliquer sur OK.
    {
        SimConnect_CallDispatch(hSimConnect,MyDispatchProc, NULL) ;
        Sleep(10) ;
    }

    while (bSimGoing)
    {    // ici, nous utilisons les événements pour régler les fréquences de NavCom et le code du
transpondeur.
        SimConnect_TransmitClientEvent(hSimConnect,0,
            EVENT_SET_COM1_FREQ, Com1Freq,
            SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY) ;
        SimConnect_TransmitClientEvent(hSimConnect,0,
            EVENT_SET_NAV1_FREQ, Nav1Freq,
            SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY) ;
        SimConnect_TransmitClientEvent(hSimConnect,0,
            EVENT_SET_XPONDER_CODE, XPonderCode,
            SIMCONNECT_GROUP_PRIORITY_HIGHEST,
            SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY) ;
    // ici, nous utilisons SimConnect_SetDataOnSimObject() pour déplacer le throttle
        hr= SimConnect_SetDataOnSimObject(hSimConnect, SET_THROTTLE_ID1,
            SIMCONNECT_OBJECT_ID_USER,0,0,sizeof(ThrottlePosition1),
            &ThrottlePosition1) ;
        hr= SimConnect_SetDataOnSimObject(hSimConnect, SET_THROTTLE_ID2,
            SIMCONNECT_OBJECT_ID_USER,0,0,sizeof(ThrottlePosition2),
            &ThrottlePosition2) ;

        if (bFlipFlop)    // alterne entre les réglages pour voir l'effet sur la simulation
        {
            Com1Freq = 0x12350 ;   // notez le format hexadécimal pour les variables BCD
            Nav1Freq = 0x11125 ;
            XPonderCode = 0x1200 ;
            ThrottlePosition1 = 100. ;
            ThrottlePosition2 = 80. ;
            bFlipFlop = false;
        }
        else
        {
            Com1Freq = 0x12825 ;
            Nav1Freq = 0x11500 ;
            XPonderCode = 0x1771 ;
            ThrottlePosition1 = 80. ;
            ThrottlePosition2 = 100. ;
            bFlipFlop = true ;
        }
        Sleep(3000) ; // attente 3 secondes
    }
    hr = SimConnect_Close(hSimConnect) ;
}  // while
return 0 ;
}

void CALLBACK MyDispatchProc(SIMCONNECT_RECV* pData, DWORD cbData, void *pContext)
{
    switch(pData->dwID)
```

```
    {
        case SIMCONNECT_RECV_ID_EVENT :
        {
        SIMCONNECT_RECV_EVENT *evt = (SIMCONNECT_RECV_EVENT*)pData ;
        switch(evt->uEventID)
            {
            case EVENT_SIM_START :
                {
                bSimGoing = true ;
                }
                break ;
            default :
                break ;
            }
            break ;
        }
        case SIMCONNECT_RECV_ID_QUIT :
        {
            bSimGoing = false;
            break ;
        }
        default :
            printf("\nReceived :%d",pData->dwID) ;
            break ;
    }
}
// ------------------------------------------------------
```

Programme de test de connexion à Flight Simulator

NDT : ce programme a été testé et fonctionne.

```
//---------------------------------------------------------
// Programme de test de connexion à Flight Simulator
//                par SimConnect
// JP Corbier - juillet 2011
// opentest.cpp
//---------------------------------------------------------

#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <strsafe.h>
#include <conio.h>

#include "SimConnect.h"

int __cdecl _tmain()
{
    int Touche;
    HRESULT hr;
    HANDLE hSimConnect = NULL;

    if (SUCCEEDED(SimConnect_Open(&hSimConnect, "opentest", NULL, 0, 0, 0)))
    {
        printf("\nConnecté à Flight Simulator!");
       printf("\n------------------------------");
        Touche = _getch();// attente d'une frappe au clavier
       hr = SimConnect_Close(hSimConnect);

        printf("\nDéconnecté de Flight Simulator");
       Touche = _getch(); // attente d'une frappe au clavier
    } else
       printf("\nImpossible de se connecter à Flight Simulator");
       Touche = _getch(); // attente d'une frappe au clavier
```

```
}
// ------------------------------------------------------------
```