Timex Data Link USB
Import Plug-In Design Guide

Timex Corporation
May 19, 2003

DOCUMENT REVISION HISTORY

REVISION: 1.0	DATE: 3/31/2003	AUTHOR: David Gray

AFFECTED PAGES	DESCRIPTION
All	Created document.

REVISION: 1.1	DATE: 5/19/2003	AUTHOR: David Gray

AFFECTED PAGES	DESCRIPTION
2 – 12	Added additional information on memory allocation for import and auto-import.

TABLE OF CONTENTS

1 Introduction

The Timex Data Link USB software supports the addition of Plug-In modules within the application. The Plug-In modules can be used to import data to certain Modes within the software, which can be downloaded to the watch. The Plug-Ins are detected by the software if they are placed in the proper directory and conform to the conventions described in this document.

This document serves as a guide for developing a Plug-Ins for the PIM software.

1.1 Applicable Documents

The following documents serves as detailed reference in the creation of this document.

- WristApp API Reference Guide

2 Developing the Application

This section defines how to create an Import Plug-In Application.

2.1 Required Tools

To develop the application, you must have a supported 32-bit Windows development platform. This could be Microsoft Visual Studio (Visual Basic and/or C++), Borland Delphi, or any other development tools that create 32-bit Windows applications in the form of a Dynamic Link Library.

2.2 Mode Support for Plug-Ins

Several modes support the use of Plug-In modules within the software. When Importing data from outside sources, these modes detect the existence of Plug-Ins and give the user the choice to utilize them. The Modes that support Plug-Ins are:

Alarm
Appointment
Note
Contact
Occasion
Schedule

Each of these modes use different data types to store and pass information. These data types, or structures, are described below and are documented in the WristApp API Reference Guide.

2.3 Creating the application

There are four functions that must be defined for the PIM to call so that the Plug-In conforms to the coding conventions.

2.3.1 SupportsImport Function

__declspec(dllexport) SupportsImport(int AppIndex) ;

2.3.1.1 Overview

All structures and character strings created in this function should be created on the free store or heap using the new allocator. The memory is is deallocated within the PIM software after it has been imported. This is illustrated in the sample code located in section 2.3.5 of this document.

This function is informs the PIM if the Plug-In supports the importation of data based on an AppIndex. The AppIndex is an indicator of a particular mode, as described below.

2.3.1.2 Function Details

Parameters:

AppIndex - This value indicates an enumeration that represents a Mode type within the PIM. These values are defined with Tucp.h which should be included within your project. The possible values are listed below.

AppIndex Enum. Name	AppIndex Enum Value	Mode Name
ALARM_APP	5	Alarm
APPT_APP	6	Appointment
NOTE_APP	7	Note
OCCASION_APP	9	Occasion
CONTACT_APP	10	Contact
SCHEDULE_APP	11	Schedule

Return Value:
Boolean indiciating whether the Plug-In supports the import.

2.3.2 GetImportNameAndDescription Function

__declspec(dllexport) GetImportNameAndDescription(int AppIndex,
char* lpszName,
int iNameMax,
char* lpszDescription,
int iDescriptionMax);

2.3.2.1 Overview

This function is used to retrieve the name and description for the Plug-In. The PIM presents this information to the user when opted to choose the import source. The return values are based on the AppIndex parameter which describes the mode that the request is for. This function will not be called if the call to SupportsImport returns a false for the given AppIndex.

2.3.2.2 Function Details

Parameters:
AppIndex - This value indicates an enumeration that represents a Mode type within the PIM for wich the name and description should be returned for. These values are defined with Tucp.h which should be included within your project. The possible values are listed below.

AppIndex Enum. Name	AppIndex Enum Value	Mode Name
ALARM_APP	5	Alarm
APPT_APP	6	Appointment
NOTE_APP	7	Note
OCCASION_APP	9	Occasion
CONTACT_APP	10	Contact
SCHEDULE_APP	11	Schedule

lpszName - This value is a null terminated string containing the name of the Plug-In.
iNameMax - This value contains the length of lpszName
lpszDescription – This value is a null terminated string containing the description of the Plug-In import.
iDescriptionMax – This value contains the length of lpszDescription

Return Value:
Boolean indiciating whether the function call succeeded.

2.3.3 Import Function

__declspec(dllexport) Import(int AppIndex,

void pData,**
LPINT iNumImports,
LPBOOL bUseAsAutoImportSource,
LPINT iModeInstance);

2.3.3.1 Overview

This function is used to retrieve the import data. The AppIndex represents the Mode for which the data is being retrieved. The data returned to the PIM will be in various forms depending on the AppIndex. Each mode uses it's own data type or structure for passing and storing information. These various types are described below, and should be used accurately.
The PIM is designed so that the modes that are enabled for Import can perform an Automatic Import at certain times, or when the user requests by clicking on the Refresh button. The Automatic Import serves the purpose of updating the mode data based on an outside wource, without user intervention. The automatic import is performed when the PIM is initialized, when the Mode instance is loaded (By editing it's data), or when the user clicks the Refresh Button. The Boolean, bUseAsAutoImportSource, is returned from the Plug-In indicating if this is to be the sole Auto Import source for the particular mode.
Within the PIM it is possible to have multiple instances of the same mode. The iModeInstance informs the Plug-In which specific instance is querying for data. This gives the Plug-In flexibility in the case that difference preferences or sets of data may be returned if desired.

2.3.3.2 Memory Allocation

All structures and character strings created in this function should be created on the free store or heap using the new allocator. The memory is is deallocated within the PIM software after it has been imported. This is illustrated in the sample code located in section 2.3.5 of this document.

2.3.3.3 Function Details

Parameters:

AppIndex - This value indicates an enumeration that represents a Mode type within the PIM that is requesting the data. These values are defined with Tucp.h which should be included within your project. The possible values are listed below.

AppIndex Enum. Name	AppIndex Enum Value	Mode Name
ALARM_APP	5	Alarm
APPT_APP	6	Appointment
NOTE_APP	7	Note
OCCASION_APP	9	Occasion
CONTACT_APP	10	Contact
SCHEDULE_APP	11	Schedule

pData - This value is a pointer to a pointer of the data returned to the PIM. The data is initialized within the Plug-In based on the size of the data, or number of imports returned. The data type or structure will vary from Mode to Mode, which is determined by the AppIndex. These data types are defined with Tucp.h which should be included within your project. These definitions are also defined at the end of this document in detail. The possible types are listed below:

AppIndex Enum. Name	AppIndex Enum Value	Data Type
ALARM_APP	5	ALARM_ITEM
APPT_APP	6	APPT_IMPORT_ITEM
NOTE_APP	7	NOTE_IMPORT_ITEM
OCCASION_APP	9	OCCASION_ITEM
CONTACT_APP	10	PHONE_IMPORT_ITEM
SCHEDULE_APP	11	SCHEDULE_IMPORT_ITEM

iNumImports - This value indicates the number of import items returned to the PIM.
bUseAsAutoImportSource – This value indicates if the Plug-In should be the source of the Automatic Import for the particular mode instance.
iModeInstance – This value indicates the particular Mode Instance that is requesting the import data.

Return Value:
Boolean indiciating whether the function call succeeded.

2.3.4 AutomaticImport Function

__declspec(dllexport) AutomaticImport(int AppIndex, void pData, LPINT iNumImports, LPINT iModeInstance);**

2.3.4.1 Overview

This function is used to retrieve the import data automatically without user intervention. This function should not display any dialogs or ask the user for choices.
The AppIndex represents the Mode for which the data is being retrieved. The data returned to the PIM will be in various forms depending on the AppIndex. Each mode uses it's own data type or structure for passing and storing information. These various types are described below, and should be used accurately.
The PIM is designed so that the modes, that are enabled for Import, can perform an Automatic Import at certain times, or when the user requests by clicking on the Refresh button. The Automatic Import serves the purpose of updating the mode data based on an outside source, without user intervention. The automatic import is performed when the PIM is initialized, when the Mode instance is loaded (By editing it's data), or when the user clicks the Refresh Button. The Boolean, bUseAsAutoImportSource, is returned from the Plug-In indicating if this is to be the sole Auto Import source for the particular mode.
Within the PIM it is possible to have multiple instances of the same mode. The iModeInstance informs the Plug-In which specific instance is querying for data. This gives the Plug-In flexibility in the case that difference preferences or sets of data may be returned if desired.

2.3.4.2 Memory Allocation

All structures and character strings created in this function should be created on the free store or heap using the new allocator. The memory is is deallocated within the PIM software after it has been imported. This is illustrated in the sample code located in section 2.3.5 of this document.

2.3.4.3 Function Details

Parameters:

AppIndex - This value indicates an enumeration that represents a Mode type within the PIM that is requesting the data. These values are defined with Tucp.h which should be included within your project. The possible values are listed below.

AppIndex Enum. Name	AppIndex Enum Value	Mode Name
ALARM_APP	5	Alarm
APPT_APP	6	Appointment
NOTE_APP	7	Note
OCCASION_APP	9	Occasion
CONTACT_APP	10	Contact
SCHEDULE_APP	11	Schedule

pData - This value is a pointer to a pointer of the data returned to the PIM. The data is initialized within the Plug-In based on the size of the data, or number of imports returned. The data type or structure will vary from Mode to Mode, which is determined by the AppIndex. These data types are defined with Tucp.h which should be included within your project. These definitions are also defined at the end of this document in detail. The possible types are listed below:

AppIndex Enum. Name	AppIndex Enum Value	Data Type
ALARM_APP	5	ALARM_ITEM
APPT_APP	6	APPT_IMPORT_ITEM
NOTE_APP	7	NOTE_IMPORT_ITEM
OCCASION_APP	9	OCCASION_ITEM
CONTACT_APP	10	PHONE_IMPORT_ITEM
SCHEDULE_APP	11	SCHEDULE_IMPORT_ITEM

iNumImports - This value indicates the number of import items returned to the PIM.
iModeInstance – This value indicates the particular Mode Instance that is requesting the import data.

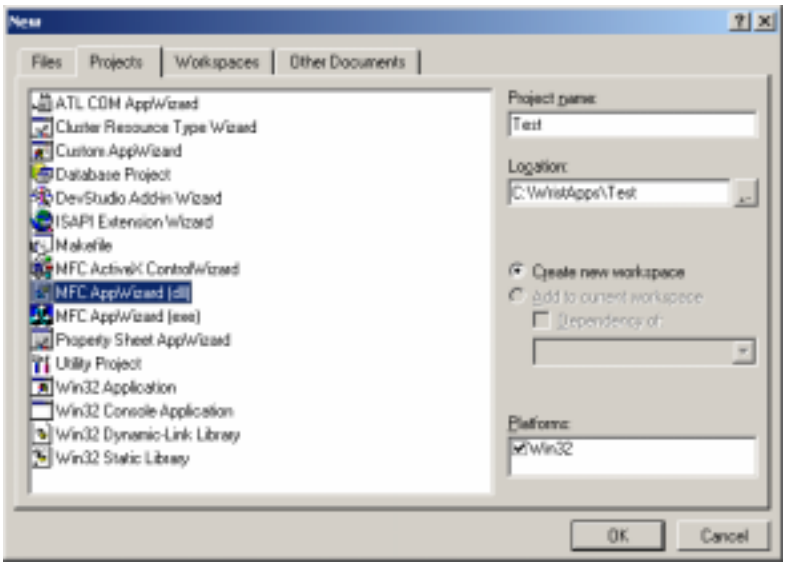
Return Value:

Boolean indiciating whether the function call succeeded.

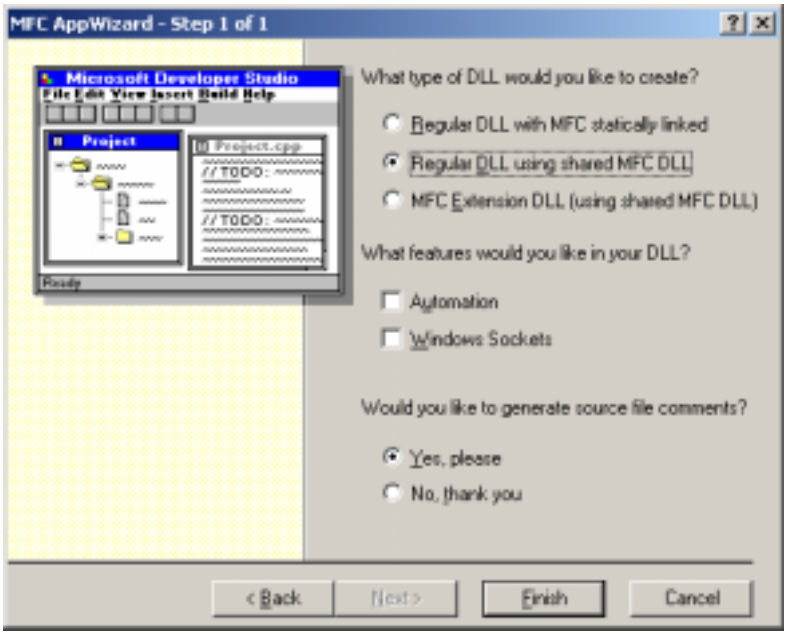
2.3.5 Creating the DLL

The following screenshots show how to create the application using Microsoft's Visual C++ 6.0.

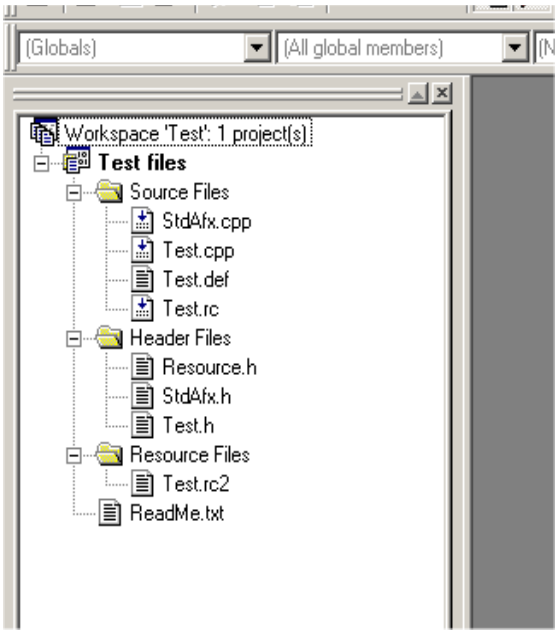
Step 1: Launch Microsoft Visual C++. Select File…New



Step 2: Select MFC AppWizard (dll) , enter the name of the project, as well as the project location., then click on OK.



Step 3: Choose Regular DLL using shared MFC DLL, and leave the other settings alone. Click on Finish.



Step 4: The project will have the basic files present for you to have a DLL. At this point, you need to define the entry functions.

```
// CTestApp

BEGIN_MESSAGE_MAP(CTestApp, CWinApp)
    //{{AFX_MSG_MAP(CTestApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestApp construction

CTestApp::CTestApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CTestApp object

CTestApp theApp;
```

Step 5: Open the class associated with the application. After **CTestAPP theApp** definition shown above, place the definitions from the two entry points above. Here is the example code for the Outlook Add-In provided with the software:

```
declspec(dllexport) SupportsImport(int AppIndex)
{

        //Need to use this macro whenever we access resources from MFC DLL
        AFX_MANAGE_STATE(AfxGetStaticModuleState());

        BOOL bretval = FALSE;
        switch (AppIndex)
        {
        case ALARM_APP:
                bretval = FALSE;
                break;
        case APPT_APP:
                bretval = TRUE;
                break;
        case NOTE_APP:
                bretval = TRUE;
                break;
        case CONTACT_APP:
                bretval = TRUE;
                break;
        case OCCASION_APP:
                bretval = TRUE;
                break;
        case SCHEDULE_APP:
                bretval = FALSE;
                break;
        default:
                bretval = FALSE;
                break;
        }

        return bretval;
}
```

```
declspec(dllexport) GetImportNameAndDescription(int AppIndex,
                                                char* lpszName,
                                                int iNameMax,
                                                char* lpszDescription,
                                                int iDescriptionMax)
{

        //Need to use this macro whenever we access resources from MFC DLL
        AFX_MANAGE_STATE(AfxGetStaticModuleState());

        CString strName, strDescription;
        BOOL bretval = FALSE;

        ////////////////////////////////////
        //  Copy the Name (Same for all)
        ////////////////////////////////////
        strName.LoadString(IDS_NAME + OLData.m_iLanguageOffset);
        strncpy(lpszName, strName, iNameMax);

        ////////////////////////////////////
        //  Copy the Description based on the AppIndex
        ////////////////////////////////////
        switch (AppIndex)
        {
        case ALARM_APP:
                strDescription.LoadString(IDS_ALARM_APP_IMPORT_DESCRIPTION);
                strncpy(lpszDescription, strDescription, iDescriptionMax);
                bretval = TRUE;
                break;

          ...
          ...
                //…Do same for other Modes
          ...
          ...

        default:
                bretval = FALSE;
                break;
        }

        return bretval;
}
```

```
declspec(dllexport) EXPORT Import(int AppIndex, void** pData, LPINT iNumImports, LPBOOL
                                        bUseAsAutoImportSource, LPINT iModeInstance)
{
       // Need to use this macro whenever we access resources from MFC DLL
       AFX_MANAGE_STATE(AfxGetStaticModuleState());
       BOOL bretval = FALSE;
       CString strValue;

       switch (AppIndex)
       {
       case ALARM_APP:
              {
                     //  create the data based on size
                     LPALARM_ITEM pAlmData = new ALARM_ITEM[2];
                     ////////////////////////////////////////
                     //  Fill in the two Alarm Items with data
                     ////////////////////////////////////////
                     //  Allocate memory for the Alarm Text on the Heap
                     strValue = "New Imported Alarm";
                     char* pAlarmText = new char[strValue.GetLength() +1];
                     strcpy(pAlarmText, strValue);
                     pAlmData[0].pszText = pAlarmText;
                     pAlmData[0].cchTextMax = strValue.GetLength();
                     //  Fill in the rest of the Information
                     pAlmData[0].bHour = 10;
                     pAlmData[0].bMinute = 15;
                     pAlmData[0].bModified = FALSE;
                     pAlmData[0].nFreq = (AlarmFrequency)0;
                     pAlmData[0].nStatus = (AlarmStatus)1;
                     pAlmData[0].wItemID = 0;

                     //  Allocate memory for the Alarm Text(2) on the Heap
                     strValue = " Another Imported Alarm ";
                     char* pAlarmText2 = new char[strValue.GetLength() +1];
                     strcpy(pAlarmText2, strValue);
                     pAlmData[1].pszText = pAlarmText2;
                     pAlmData[1].cchTextMax = strValue.GetLength();
                     //  Fill in the rest of the Information
                     pAlmData[1].bHour = 14;
                     pAlmData[1].bMinute = 45;
                     pAlmData[1].bModified = FALSE;
                     pAlmData[1].nFreq = (AlarmFrequency)3;
                     pAlmData[1].nStatus = (AlarmStatus)3;
                     pAlmData[1].wItemID = 0;

                     //  Set the pointer to our local data
                     *pData = (void*)pAlmData;

                     //  Set the number of imports we are returning
                     *iNumImports = 2;

                     //  This wont be the Automatic Import Source, so set to false
                     bUseAsAutoImportSource = FALSE;

                     //  The call is successful
                     bretval = TRUE;
                     break;
              }
         ...
         ...
              //…Do same for other Modes
         ...
         ...
       default:
              //  call failed, we don’t support this type
              bretval = FALSE;
              break;
       }

       return bretval;
}
```

```
declspec(dllexport) AutomaticImport(int AppIndex, void** pData, LPINT iNumImports,
                                                    LPINT iModeInstance)
{

        //Need to use this macro whenever we access resources from MFC DLL
        AFX_MANAGE_STATE(AfxGetStaticModuleState());
        BOOL bretval = FALSE;
        switch (AppIndex)
        {
        case ALARM_APP:
                {
                        //  create the data based on size
                        LPALARM_ITEM pAlmData = new ALARM_ITEM[2];

                        ////////////////////////////////////////
                        //  Fill in the two Alarm Items with data
                        ////////////////////////////////////////
                        //  Allocate memory for the Alarm Text on the Heap
                        strValue = "New Imported Alarm";
                        char* pAlarmText = new char[strValue.GetLength() +1];
                        strcpy(pAlarmText, strValue);
                        pAlmData[0].pszText = pAlarmText;
                        pAlmData[0].cchTextMax = strValue.GetLength();
                        //  Fill in the rest of the Information
                        pAlmData[0].bHour = 10;
                        pAlmData[0].bMinute = 15;
                        pAlmData[0].bModified = FALSE;
                        pAlmData[0].nFreq = (AlarmFrequency)0;
                        pAlmData[0].nStatus = (AlarmStatus)1;
                        pAlmData[0].wItemID = 0;

                        //  Allocate memory for the Alarm Text(2) on the Heap
                        strValue = " Another Imported Alarm ";
                         char* pAlarmText2 = new char[strValue.GetLength() +1];
                        strcpy(pAlarmText2, strValue);
                        pAlmData[1].pszText = pAlarmText2;
                        pAlmData[1].cchTextMax = strValue.GetLength();
                        //  Fill in the rest of the Information
                        pAlmData[1].bHour = 14;
                        pAlmData[1].bMinute = 45;
                        pAlmData[1].bModified = FALSE;
                        pAlmData[1].nFreq = (AlarmFrequency)3;
                        pAlmData[1].nStatus = (AlarmStatus)3;
                        pAlmData[1].wItemID = 0;

                        //  Set the pointer to our local data
                        *pData = (void*)pAlmData;

                        //  Set the number of imports we are returning
                        *iNumImports = 2;

                        //  The call is successful
                        bretval = TRUE;
                        break;
                }
          ...
          ...
                //…Do same for other Modes
          ...
          ...
        default:
                //  call failed, we don't support this type
                bretval = FALSE;
                break;
        }

        return bretval;
}
```

Please note that the CreateDatabase routine defined above is a generic function that could be called to create the actual WristApp database.

After adding these four functions, you must then add the function names to the DEFinition file. Open TEST.DEF and add the four functions as shown:

```
LIBRARY      "Test"
DESCRIPTION  Test Windows Dynamic Link Library'

EXPORTS
        SupportsImport
        GetImportNameAndDescription
        Import
        AutomaticImport
```

2.4 Installing the Application

2.4.1 Distributing the PlugIn

Once compiles the Plug-In should be in the form of a Dynamic Link Library. The file should be placed within the AddIns directory of the application folder. The PIM software will automatically detect it's presence.

3 Trademarks

4 Structure Definitions

The following definitions are found within the Tucp.h file and are presented here for reference.

```
enum AppIndex
{
   TOD_APP              = 0,
   COMM_APP             = 1,
   CHRONO_APP           = 2,
   TIMER_APP            = 3,
   INTTIMER_APP         = 4,
   ALARM_APP            = 5,
   APPT_APP             = 6,
   NOTE_APP             = 7,
   OPTION_APP           = 8,
   OCCASION_APP         = 9,
   CONTACT_APP          = 10,
   SCHEDULE_APP         = 11,
   SYNCHRO_APP          = 12,
   COUNTER_APP          = 13,
   SOUND_APP            = 16,
   WRISTAPP_APP         = 17
};

typedef struct _ALARM_ITEM
{
   WORD wItemID;                  // must be a unique index of the item to which this structure refers
   LPTSTR  pszText;               // pointer to a null-terminated string that contains the item text
   int  cchTextMax;               // number of characters in the buffer pointed to by pszText
   AlarmStatus nStatus;           // alarm status: 0=unused, 1=used/disarmed, 3=used/armed
   AlarmFrequency nFreq;          // frequency (or occurence) of the alarm
   BYTE bHour;                    // alarm hour
   BYTE bMinute;                  // alarm minute
   DATE dtReserved1;              // not used
   BYTE bModified;         // modified or not
} ALARM_ITEM, *LPALARM_ITEM;


typedef struct _APPT_IMPORT_ITEM
{
   WORD wItemID;                  // must be a unique index of the item to which this structure refers
   LPTSTR  pszText;               // pointer to a null-terminated string that contains the item text
   int  cchTextMax;               // number of characters in the buffer pointed to by pszText
   DATE dtStartDateTime;          // the date and time of the first occurrence of the appt
   ApptStatus nStatus;            // appointment status: 0=unused, 1=disarmed, 3=armed
   Frequency nFreq;               // frequency (or occurence) of the appt
   Reminder nReminder;            // reminder offset (used to calculate prenotification time)
   DATE dtReserved1;              // not used
   DATE dtReserved2;              // not used
   BYTE bModified;                // modified or not
   LPTSTR  pszImportIDText;       // pointer to a null-terminated string that contains the ID of the Import Item, as it
                                  // pertains to the Add-In
   int  cchImportIDTextMax;       // number of characters in the buffer pointed to by pszImportIDText
} APPT_IMPORT_ITEM, *LPAPPT_IMPORT_ITEM;
```

```
enum ApptStatus
{
   APPT_UNUSED   = 0,
   APPT_DISARMED = 1,
   APPT_ARMED    = 3
};

enum Frequency
{
   APPTFREQ_1DAY    = 0, // 1-day (one-time) occurance
   APPTFREQ_DAILY   = 1,
   APPTFREQ_WEEKDAY = 2,
   APPTFREQ_WEEKEND = 3,
   APPTFREQ_WEEKLY  = 11,
   APPTFREQ_MONTHLY = 12,
   APPTFREQ_YEARLY  = 13
};

enum Reminder
{
   REMINDER_0MINS  = 0,
   REMINDER_5MINS  = 1,
   REMINDER_10MINS = 2,
   REMINDER_15MINS = 3,
   REMINDER_30MINS = 4,
   REMINDER_60MINS = 5,
   REMINDER_2HRS   = 6,
   REMINDER_3HRS   = 7,
   REMINDER_4HRS   = 8,
   REMINDER_5HRS   = 9,
   REMINDER_6HRS   = 10,
   REMINDER_8HRS   = 11,
   REMINDER_10HRS  = 12,
   REMINDER_12HRS  = 13,
   REMINDER_24HRS  = 14,
   REMINDER_48HRS  = 15
};

typedef struct _NOTE_IMPORT_ITEM
{
   WORD wItemID;                 // must be a unique index of the item to which this structure refers
   LPTSTR  pszText;              // pointer to a null-terminated string that contains the item text
   int  cchTextMax;              // number of characters in the buffer pointed to by pszText
   BYTE bStatus;                 // note status: 0=unused, 1=used
   BYTE bModified;               // modified or not
   LPTSTR  pszImportIDText;      // pointer to a null-terminated string that contains the ID of the Import Item, as it
                                 // pertains to the Add-In
   int  cchImportIDTextMax;      // number of characters in the buffer pointed to by pszImportIDText
} NOTE_IMPORT_ITEM, *LPNOTE_IMPORT_ITEM;
```

```
enum OccasionType
{
   OCCASION_NONE     = 0,
   OCCASION_BDAY     = 1,
   OCCASION_ANNIV    = 2,
   OCCASION_HOLIDAY  = 3,
   OCCASION_VACATION = 4
};

typedef struct _OCCASION_ITEM
{
   WORD wItemID;                // must be a unique index of the item to which this structure refers
   LPTSTR  pszText;             // pointer to a null-terminated string that contains the item text
   int  cchTextMax;             // number of characters in the buffer pointed to by pszText
   DATE dtStartDateTime;        // the date and time of the original occurrence of the occasion
   BYTE bStatus;                // occasion status: 0=non-recurring, 1=recurring
   OccasionType nType;          // type of the occasion
   BOOL bIgnoreYear;            // true if you want the occasion to not have the year specified
   DATE dtReserved1;            // not used
} OCCASION_ITEM, *LPOCCASION_ITEM;

typedef struct _PHONE_IMPORT_ITEM
{
        LPTSTR  pszNameText;            // pointer to a null-terminated string that contains the name text
        int  cchNameTextMax;            // number of characters in the buffer pointed to by pszNameText
        LPTSTR  pszPhone1Text;          // pointer to a null-terminated string that contains the phone1 text
        int  cchPhone1TextMax;          // number of characters in the buffer pointed to by pszPhone1Text
        LPTSTR  pszPhone1TypeText;      // pointer to a null-terminated string that contains the phone type
        int  cchPhone1TypeTextMax;      // number of characters in the buffer pointed to by pszPhone1TypeText
        LPTSTR  pszPhone2Text;          // pointer to a null-terminated string that contains the phone2 text
        int  cchPhone2TextMax;          // number of characters in the buffer pointed to by pszPhone2Text
        LPTSTR  pszPhone2TypeText;      // pointer to a null-terminated string that contains the phone type
        int  cchPhone2TypeTextMax;      // number of characters in the buffer pointed to by pszPhone2TypeText
        LPTSTR  pszPhone3Text;          // pointer to a null-terminated string that contains the phone3 text
        int  cchPhone3TextMax;          // number of characters in the buffer pointed to by pszPhone3Text
        LPTSTR  pszPhone3TypeText;      // pointer to a null-terminated string that contains the phone type
        int  cchPhone3TypeTextMax;      // number of characters in the buffer pointed to by pszPhone3Text
        LPTSTR  pszPhone4Text;          // pointer to a null-terminated string that contains the phone4 text
        int  cchPhone4TextMax;          // number of characters in the buffer pointed to by pszPhone4Text
        LPTSTR  pszPhone4TypeText;      // pointer to a null-terminated string that contains the phone type
        int  cchPhone4TypeTextMax;      // number of characters in the buffer pointed to by pszPhone4Text
        LPTSTR  pszAddressText;         // pointer to a null-terminated string that contains the address text
        int  cchAdressTextMax;          // number of characters in the buffer pointed to by pszAddressText
        LPTSTR  pszEmailText;           // pointer to a null-terminated string that contains the Email text
        int  cchEmailTextMax;           // number of characters in the buffer pointed to by pszEmailText
        LPTSTR  pszImportIDText;        // pointer to a null-terminated string that contains the ID of the Import
                                        // Item, as it pertains to the Add-In
        int  cchImportIDTextMax;        // number of characters in the buffer pointed to by pszImportIDText
        int         iNumDupContacts;    // number indicating the number structures being imported related to
                                        // this particular contact (Done when contact has more than 4 phone
                                        // numbers), otherwise 1
        int  iImportIndex;              // number indicating the current index of the contact import related to
                                        // iNumDupContacts
} PHONE_IMPORT_ITEM, *LPPHONE_IMPORT_ITEM;
```

```
typedef struct _SCHEDULE_IMPORT_ITEM
{
   WORD wGroupID;              // Should be 0 for imports
   LPTSTR  pszScheduleName;    // pointer to a null-terminated string that contains the schedule name
   int  cchScheduleNameMax;    // number of characters in the buffer pointed to by pszScheduleName
   LPTSTR  pszGroupText;       // pointer to a null-terminated string that contains the group text
   int  cchGroupTextMax;       // number of characters in the buffer pointed to by pszGroupText
   LPTSTR  pszGroupLabel;      // pointer to a null-terminated string that contains the group label
   int  cchGroupLabelMax;      // number of characters in the buffer pointed to by pszGroupLabel
   ScheduleType nType;         // schedule type according to the enum
   WORD wItemID;               // must be a unique index of the item to which this structure refers
   LPTSTR  pszText;            // pointer to a null-terminated string that contains the item text
   int  cchTextMax;            // number of characters in the buffer pointed to by pszText
   DATE dtDateTime;            // the date and time of the schedule item
   int nDayOfWeek;             // day of the week for schedule type DOW_TIME; 0=sun, 1=mon,... 6=sat
} SCHEDULE_IMPORT_ITEM, *LPSCHEDULE_IMPORT_ITEM;
```