M851 WristApp GUI Plug-in Design Guide

Timex Corporation
March 31, 2003

DOCUMENT REVISION HISTORY

REVISION: 1.0	DATE: 5/06/2003	AUTHOR: Brigham W. Thorp

AFFECTED PAGES	DESCRIPTION
All	Created document.

TABLE OF CONTENTS

1 Introduction

The Timex Data Link USB watch supports downloading of programs called WristApps that add additional functionality to the device. Unlike built-in applications that are present in the device ROM, WristApps have code that use some watch memory. In addition, a WristApp may or may not have a database associated with it. For example, the World Time WristApp contains a database while the Pulse Wrist App does not. The database associated with the WristApp gives the user the ability to add different data to the mode. The WristApp GUI Plug-in gives users the ability to change data within a WristApp database.

This document serves as a guide for developing a WristApp GUI Plug-in for the PIM (Personal Information Manager) software.

1.1 Applicable Documents

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

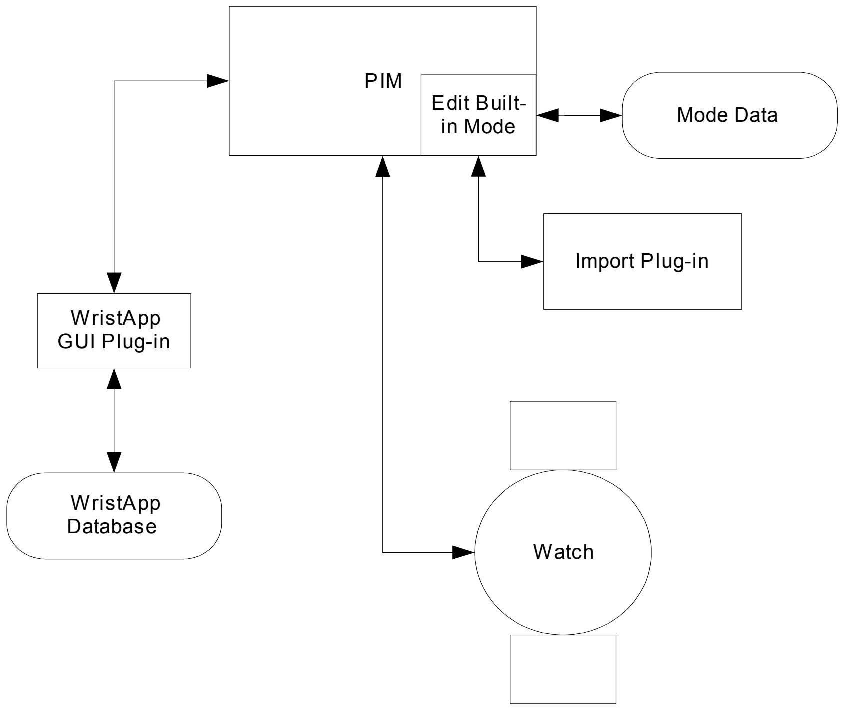
- M851 WristApp API Reference Guide

Please visit http://www.timex.com/developer for more information.

2 System Overview

This part of the documentation describes how the system is integrated as a whole. The PIM is the main application that handles all of the different parts of the system. When you double click on an application in the mode list, the PIM either launches the built-in mode editor if the mode is an internal application, or it launches the WristApp GUI Plug-in if the mode is a WristApp and has a GUI Plug-in associated with it. The PIM also handles calling the module that handles communicating data with the watch. Finally, the PIM contains all of the data editors for each of the built-in modes. The data-editors themselves handle import plug-ins that allow you to import data from different sources such as CSV files or schedules over the Internet. The simple diagram shown below represents the Timex Data Link USB system.

Timex Data Link USB PC Software Architecture



3 Developing the Application

This section defines how to create a WristApp GUI Plug-in Application. This document assumes that a WristApp has been developed in accordance with the WristApp Design Guide.

3.1 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.

3.2 Creating the application

There are two functions that must be defined for the PIM to call so that the application can show it's configuration screen, and the application can create the database on the fly (which could handle getting data from the Web, for example, just before a download occurs, without having to enter the data manually again).

There are two functions that the PIM software may call within the WristApp GUI Plug-in. These two functions are *Show* and *ProcessData*. Both functions are described below.

3.2.1 Show Function

extern "C" BOOL FAR PASCAL EXPORT Show(LPCTSTR lpszDataPath, LPVOID pData);

This function is used to show a GUI to the user so that they can modify data using a user interface. This function is called by the PIM when the user double-clicks on the WristApp entry in the mode list. Typically, this function will instantiate a class (such as a dialog) that is then shown.

When the user has finished with the dialog, the updated database should be saved. For consitency between applications, Timex recommends using a dialog based class and saving data on the OK button press. Closing of the dialog should occur on the OK button, Cancel button, or 'X' button press.

Parameters:
lpszDataPath – This is the directory where the data file (WristApp database) is located
pData – NULL will be passed. This value is only used for the internal application dialogs.

3.2.2 ProcessData Function

extern "C" BOOL FAR PASCAL EXPORT ProcessData(LPCTSTR lpszDataPath, LPVOID pData);

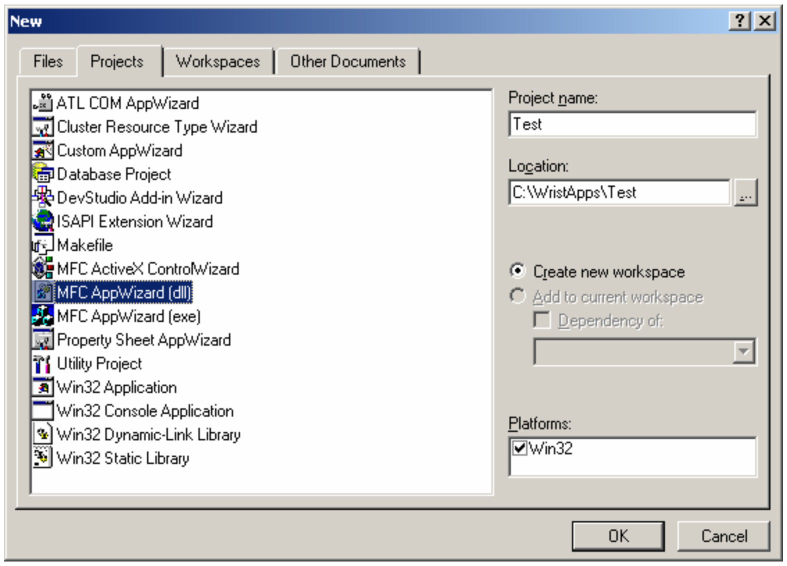
This function is used to update the WristApp database with new data without having input from the user. The function is called when the user clicks on the Send and Refresh buttons, as well as if the WristApp is setup for Auto Import; all database data is regenerated. Everything from reading the old data, gathering the new data, and creating the database is done in this function.

Parameters:
lpszDataPath – This is the directory where the data file (WristApp database) is located
pData – NULL will be passed. This value is only used for the internal application dialogs.

3.2.3 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



- ATL COM AppWizard
- Cluster Resource Type Wizard
- Custom AppWizard
- Database Project
- DevStudio Add-in Wizard
- ISAPI Extension Wizard
- Makefile
- MFC ActiveX ControlWizard
- MFC AppWizard (dll) [✓]
- MFC AppWizard (exe)
- Property Sheet AppWizard
- Utility Project
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Static Library

Project name: Test

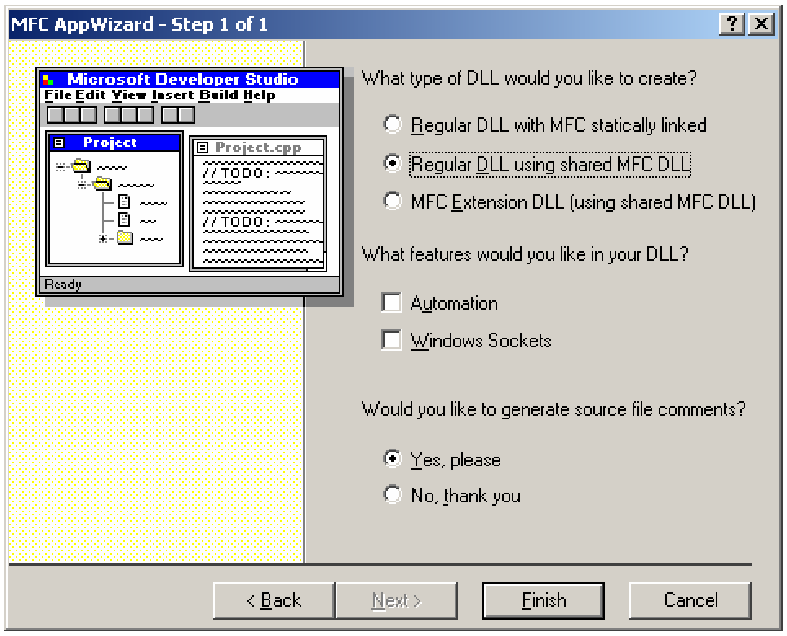
Location: C:\WristApps\Test

[✓ Create new workspace] Add to current workspace

Dependency of:

Platforms: [✓] Win32

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


MFC AppWizard - Step 1 of 1
Microsoft Developer Studio
File Edit View Insert Build Help
Project
Project.cpp
// TODO:
// TODO:
Ready
< Back
Next >
Finish
Cancel

What type of DLL would you like to create?

Regular DLL with MFC statically linked [✓ Regular DLL using shared MFC DLL] MFC Extension DLL (using shared MFC DLL)

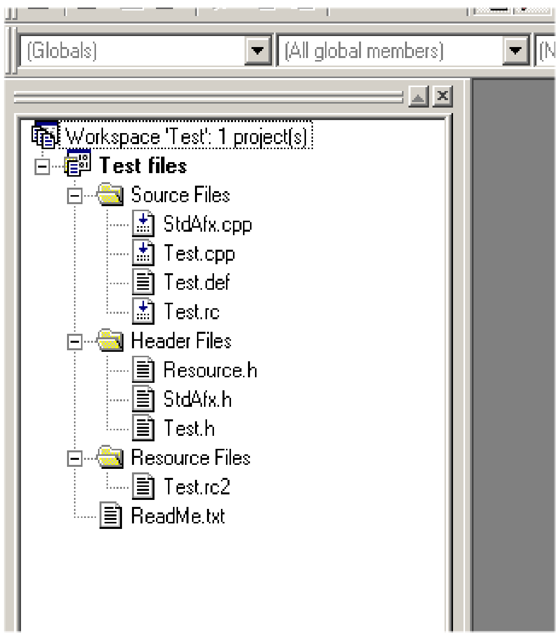
What features would you like in your DLL?

[ ] Automation
[ ] Windows Sockets

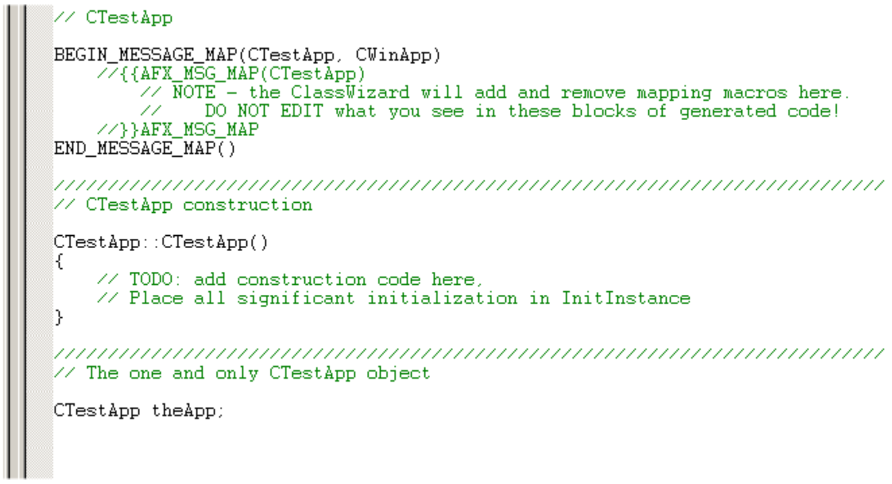
Would you like to generate source file comments?

[✓ Yes, please] No, thank you

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.



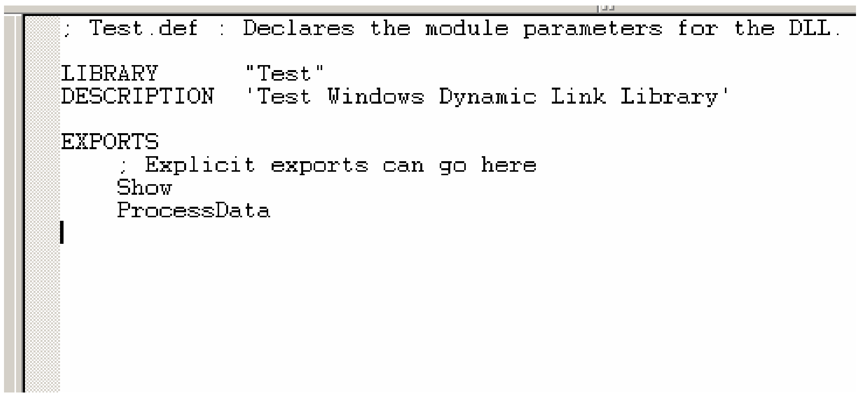
```
// CTestApp

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 WorldTime WristApp:

```
extern "C" BOOL FAR PASCAL EXPORT Show(LPCTSTR lpszDataPath, LPVOID pData)
{
    //Need to use this macro whenever we access resources from MFC DLL
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    CString sINIFile = lpszDataPath;
    sINIFile += "App\\TestApp.ini";

    CTestApp dlg;
    dlg.m_sIniFileName = lpszDataPath;
    dlg.m_sIniFileName += "App\\TestApp.ini";

    return dlg.DoModal();
}

extern "C" BOOL FAR PASCAL EXPORT ProcessData(LPCTSTR lpszDataPath, LPVOID pData)
{
    //Need to use this macro whenever we access resources from MFC DLL
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    CString sINIFile = lpszDataPath;
    sINIFile += "TestApp.ini";

    theApp.CreateDatabase(sINIFile);

    return TRUE;
}
```

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



```
; Test.def : Declares the module parameters for the DLL.

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

EXPORTS
    ; Explicit exports can go here
    Show
    ProcessData
```

3.2.4 Example application – World Time

The World Time WristApp that comes with the Timex Data Link USB has a user interface that allows you to configure the different time zones within the WristApp.

As described above, the Show and ProcessData functions are called by the PIM to launch the GUI Plug-in. In the World Time WristApp Plug-in, created using Microsoft's Visual C++ with MFC libraries, there is a dialog class that handles such messages as WM_INITDIALOG, BN_CLICKED for the OK and Cancel buttons, as well as WM_DESTROY. In addition to the dialog class, there is an app class based on the CWinApp base class which has routines for saving the World Time database.

The pseudo-code below shows how the World Time GUI Plug-in works. Please refer to the included source code for the actual functionality:

```
Show
{
        Instantiate the app class
        Initialize the character mapping for the watch
        Initialize the World Time dialog
        {
                Load the World Time cities from the INI file
                Fill the list view with the city data
                Enter message loop (exit on OK, Canel, or Close events)
                OK button press
                {
                        Retrieve dialog data
                        Save changes to INI file
                        Create the Wrist App database
                }
        }
}

ProcessData
{
        Instantiate the app class
        Initialize the character mapping for the watch
        Load the World Time cities from the INI file
        Create the Wrist App database
}
```

3.3 WristApp Information

3.3.1 Distributing the WristApp

The following files generated by the WristApp developer will be used for distribution of the wristapp. Application_name may be any text supported by Windows, as long as the total length of the name (with extension) does not exceed 256 characters.

Filename	Description
application_name.**APP**	*Information file required by PIM.*
application_name.**TXT (or .HTML)**	*Description of the wristapp and its operation. This may also be a .HTML file. If there are graphics with the HTML, they must be added to this directory as well*
application_name.**_PAR_018.BIN**	*Parameter file required by M851 OS to initialize the wristapp in the system.*
application_name.**_CODE_018.BIN**	*WristApp code.*
application_name.**_DBASE_018.BIN**	*WristApp database file – this file is created by the WristApp PC interface.*
application_name.**DLL**	*WristApp PC interface – this is the file that will be created by following this documentation. This is only required if changing data is allowed*

Please refer to the WristApp Design Guide on the contents of the WristApp configuration files. Please note that the DLL that is created is present in the same directory as the other WristApp files (the App subdirectory where the PIM is installed) . However, the developer is free to place any additional subdirectories into the WristApp directory to be used for data storage.

3.3.2 Changed WristApp Data

When the database changes between downloads, the packager (TUCP.DLL) checks the size of the database and the contents versus the last data sent to the watch. If the size of the new data is larger than the previously allocated space, then a full download will occur. There is no call that needs to be done in order for this to occur. The packager handles this for you. All that is required is that the database differs in some way versus the previously downloaded database. If the database is the same size, yet the contents are different, then the database will be re-downloaded, but the code will not have to be downloaded again. This applies to all applications with database data associated with them.

3.3.3 Reading WristApp Data

If the WristApp has been modified in the watch, it will be uploaded back to the PC when the next read occurs, which happens when the watch is connected to the PC. The packager DLL reads the files and puts it into the DB directory in the PIM's installation directory.

Each wristapp that has been uploaded back to the PC will have the name upwristappdbX.bin where X is the instance of the application. When downloading, the PIM will place the instance number in the WristApp's associated .INI file (which is created if it doesn't exist). The file shown below describes how the INI file will be created:

```
[System]
Instance=5
```

In the example above, the WristApp has an instance of 5, so the WristApp's file will be upwristappdb5.bin. When the user double-clicks on your WristApp in the PIM's mode list, you can read this file before showing the user interface and show the changes to the user.

If the above Instance value was NULL (no value to the right of the equals sign), then that means no data was read for the WristApp. In this case, you should provide default data for the user when the UI launches.

This file should be the same format as the database that was downloaded, and its up to the WristApp code to make sure that the database stays the same.

4 Trademarks

TIMEX is a registered trademark and service mark of Timex Corporation.
TIMEX DATA LINK and WristApp are trademarks of Timex Corporation in the U.S. and other countries.
Night-Mode is a registered trademark of Timex Corporation.
INDIGLO is a registered trademark of Indiglo Corporation.