digitalPersona®

Gold SDK - One Touch for Windows

Case by Case Migration Path

The intent of this document is to provide a case by case mapping of a Gold SDK implementation to a One Touch for Windows (OTW) SDK implementation. The code on the left (Gold API) can be implemented by the code on the right (OTW API). The key difference between the two APIs is the decision to use windows messages for device events in OTW in order to achieve seamless integration into Windows environments. Outside the API, the advantages of the OTW's new runtime engine provides increased performance in terms of matching speed, support for Citrix and MTS environments, 64-bit Vista and XP compatibility, and support for our latest hardware technologies such as the 4500 and 2500 readers.

Device Initialization

This section covers the preparation of an application to receive device specific events.

Gold SDK (dpfpfns.h)

Required calls:

FT_init() *Initialize the Gold SDK dpfpfns module.*
FT_CreateContext() *Create the context for current thread.*
FT_getNumDevices() *Check how many fingerprint readers are connected.*
FT_getDeviceList() *Retrieve list of connected devices.*
FT_connectDevice() *Connect to the fingerprint reader.*
FT_setupUILink() *Set up function pointers to receive events from the Gold . SDK. These events include fingerprint coming, fingerprint leaving, image acquired etc…*

Code would look something like:

```
//Client callback declarations – these should be defined somewhere in your code
FT_ACTION onAction (FT_STATUS_PT pStatus, void *pParam);
FT_RETCODE releaseDisplayBuf (FT_DISPLAY_BUF_PT, void *pParam);
FT_DISPLAY_BUF_PT getDisplayBuf (FT_FTR_TYPE, void *pParam);

//Variable declarations
FT_DEVICE_INFO* devInfo;   //Used to store device specific information
FT_HANDLE context;          //Used to track the particular thread context of an operation
FT_UI_LINK FtUILink;        //Used to setup the callbacks for device specific operations
int count;                  //Count of connected devices

//Executable code……

//Setup your callbacks and parameters for handling events
FtUILink.getAction = onAction;
FtUILink.releaseDisplayBuf = releaseDisplayBuf;
FtUILink.getDisplayBuf = getDisplayBuf;
FtUILink.numIntensityLevels = 32;
FtUILink.timeout = 2000;
FtUILink.extention = NULL;
FtUILink.displayOrientation = FT_PORTRAIT;
FtUILink.displayBufSize.width= 60;
```

One Touch for Windows C/C++ (DPDevClt.h)

Required calls:

DPFPInit() *Initialize the DPDevClt (dpfpapi) module*
DPFPEnumerateDevices() *Retrieve the number of connected devices and their device GUIDs .*
DPFPCreateAcquisition() *Connect a particular device to a window.*

Code would look something like:

```
//Variable declarations
ULONG count;     //Count of connected devices
GUID* DevUid;    //Used for device UID array
HDPOPERATION phOperation;  //Keeps track of an acquisition operation
HANDLE hWin;  //Handle to the window to send device specific events to

//Define a window's callback procedure.  This is where you handle fingerprint events.
int CALLBACK wndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
{
   switch (msg)
  {
          case FP_EVENT:
           switch (wparam)
           {
                 case WN_COMPLETED:
                       //Image successfully captured
                       break;
                 case WN_FINGER_TOUCHED:
                      //Finger touch event received
                       break;
           }
           break;
   }
  return 0;
}

//Executable code……
```

```
FtUILink.displayBufSize.height = 80;


//Initialize the toolkit
FT_init();

//Get the number of connected devices
FT_getNumDevices(&count);

//Get a list of device info structures describing the connected devics
FT_getDeviceList(count,&devInfo);

//Create a context for this particular thread
FT_createContext(&context);

//Attach a device – FtUILink callbacks will be called based on events
FT_connectDevice(context,(devInfo)->devId),FT_REGULAR_PRIORITY);

//Callbacks can now be made once an operation is begun (FT_verify, FT_register, etc)
FT_setupUILink(context,&FtUILink,NULL);
```

```
//Create your window (in this case a win32 API call)
 hWin=CreateWindowEx(0,WC_DIALOG,"DPWin",
WS_OVERLAPPEDWINDOW,200,200,440,200,NULL,NULL,NULL,NULL);

//Attach your window callback procedure to the window
SetWindowLong(hWin,DWL_DLGPROC,(long)wndProc);

//Initialize the toolkit and get a list of connected devices
DPFPInit();
DPFPEnumerateDevices(&count,&DevUid); //!!!Must call matching DPFPBufferFree()

//Register a custom message for your window to receive (in this case a win32 API call)
UINT FP_EVENT=RegisterWindowMessage("FP_EVENT");

//Attach a particular device to your window.
DPFPCreateAcquisition(DP_PRIORITY_NORMAL,*(DevUid),DP_SAMPLE_TYPE_IMAGE,
hWin,FP_EVENT,&phOperation);
```

Image Acquisition

This section covers how to capture\acquire an image (non-display) from the device.

Gold SDK (dpFpFns.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required calls:

FT_acquireImage() *Begins an image acquisition operation, FtUILink callbacks will be called based on device messages.*

Code would look something like:

```
//Variable declarations
FT_IMAGE_PT    image;   //Used to store non-displayable image
FT_RESULT      rslt;    //Result of image acquisition operation
unsigned int   usedLen; //Length of non-displayable image
FT_BYTE*       displayImage;   //Used to store displayable image
unsigned int   displayImageSize; //Displayable image size

//Executable code……

//Allocate image buffer
image= new unsigned char[300000];

//Begin image acquisition.  The image is copied into the buffer passed ('image' here).
FT_acquireImage(context, FT_VER_FTR,300000,image,&usedLen,&rslt);

//To get display (OPTIONAL)
   //Get image size (assuming devInfo already retrieved – see above section)
   displayImageSize=devInfo->imgSize.width*devInfo->imgSize.height;

   // To get display image call FT_getImagePixels
   FT_getImagePixels(image,displayImageSize,displayImage);
```

One Touch for Windows C/C++ (DPDevClt.h & dpFtrEx.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required calls:

DPFPStartAcquisition() *Begins the posting of messages to your window.*

Code would look something like:

```
//Variable declarations
DATA_BLOB* blob;         //DATA_BLOB declared in wincrypt.h.  Points to image.
unsigned char* displayImage;  //Used to store displayable image

//Define a window's callback procedure.  This is where you handle fingerprint events.
int CALLBACK wndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
{
   switch (msg)
  {
      case WM_SHOWWINDOW:
            //Start the posting of events to our window
            DPFPStartAcquisition(DP_PRIORITY_NORMAL,*(DevUid)
                        , DP_SAMPLE_TYPE_IMAGE,hWin,FP_EVENT,&phOperation);
             break;
      case FP_EVENT:
           switch (wparam)
           {
                case WN_COMPLETED:
                      //Get a reference to the image
                      Blob= (DATA_BLOB*)  lparam;

                      //Specify your image dimension requirements
                      FT_IMAGE_SIZE size;
                      size.height=110;
```

```
                    size.width=82;
                    displayImage=new unsigned char[size.height*size.width];

                     // (OPTIONAL) To get display image of 8-bit grey scale
                     // call FX_getDisplayImage
                     FX_getDisplayImage(fxContext,
                    blob->pbData,&size,false,256,image1);
                    break;
            }
            break;
  }
   return 0;
}
```

Feature Acquisition

This section covers feature acquisition.

Gold SDK (dpFpFns.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required calls:

FT_acquireFeatures() *Begins an image acquisition operation, FtUILink callbacks will be called based on device messages, and returns the extracted features.*

FT_getFeaturesLen() *Get possible feature lengths.*

Code would look something like:

```
//Variable declarations
int preRegLengthMax;        //Max length of pre registration features
int preRegLengthMin;        //Min length of pre registration features
int verLengthMax;           //Max length of verification features
int verLengthMin;           //Min length of verification features
unsigned char preRegFtrs;  //Used to store pre-registration features
unsigned char verFtrs;      //Used to store verification features
FT_IMG_QUALITY imgQuality;  //Quality of image
FT_FTR_QUALITY ftrQuality;    //Quality of extracted features
FT_BOOL rslt;                 //Result of extraction operation

//Executable code……

//Get feature lengths and allocate feature buffers
FT_getFeaturesLen(FT_PRE_REG_FTR,0,&preRegLengthMax,&preRegLengthMin);
FT_getFeaturesLen(FT_VER_FTR,0,&verLengthMax,&verLengthMin);
preRegFtrs = new unsigned char[preRegLengthMax];
verFtrs = new unsigned char[verLengthMax];

//Begin image acquisition and return the pre registration features
FT_acquireFeatures(context, FT_PRE_REG_FTR,preRegLengthMax
                        , preRegFtrs,&imgQuality,&ftrQualit,&rslt);
```

One Touch for Windows C/C++ (DPDevClt.h, dpFtrEx.h, DPMatch.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required Calls:

FX_extractFeatures() *Get extracted features.*

FX_getFeaturesLen() *Gets possible feature lengths.*

Code would look something like:

```
//Variable declarations
DATA_BLOB* blob;            //DATA_BLOB declared in wincrypt.h
int preRegLengthMax;        //Max length of pre registration features
int preRegLengthMin;        //Min length of pre registration features
int verLengthMax;           //Max length of verification features
int verLengthMin;           //Min length of verification features
unsigned char preRegFtrs;  //Used to store pre-registration features
unsigned char verFtrs;      //Used to store verification features
FT_IMG_QUALITY imgQuality;  //Quality of image
FT_FTR_QUALITY ftrQuality;    //Quality of extracted features
FT_BOOL rslt;                 //Result of extraction operation

//Define a window's callback procedure.  This is where you handle fingerprint events.
int CALLBACK wndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
{
   switch (msg)
  {
      case WM_SHOWWINDOW:
            //Start the posting of events to our window
            DPFPStartAcquisition(DP_PRIORITY_NORMAL,*(DevUid)
                        , DP_SAMPLE_TYPE_IMAGE,hWin,FP_EVENT,&phOperation);
             break;
      case FP_EVENT:
           switch (wparam)
          {
```

```
//Or begin image acquisition and return the verification features
FT_acquireFeatures(context, FT_VER_FTR,verLengthMax
                        , verFtrs,&imgQuality,&ftrQualit,&rslt);
```

```
                case WN_COMPLETED:
                    //Get a reference to the image
                    Blob= (DATA_BLOB*)  lparam;

                    //Extract pre-registration features
                FX_extractFeatures(fxContext,db->cbData,db->pbData
                                , FT_PRE_REG_FTR, preRegLengthMax
                                ,  preRegFtrs,&imgQuality,&ftrQuality,&rslt);

                    //Or extract verification features
                FX_extractFeatures(fxContext,db->cbData,db->pbData
                                , FT_VER_FTR, verLengthMax
                                ,  verFtrs,&imgQuality,&ftrQuality,&rslt);
                 break;
            }
           break;
 }
  return 0;
}

//Executable code……

//Get feature lengths and allocate feature buffers before feature extraction
FT_getFeaturesLen(FT_PRE_REG_FTR,0,&preRegLengthMax,&preRegLengthMin);
FT_getFeaturesLen(FT_VER_FTR,0,&verLengthMax,&verLengthMin);
preRegFtrs= new unsigned char[preRegLengthMax];
verFtrs= new unsigned char[verLengthMax];
```

Registration

This section covers registration of fingerprints

Gold SDK (dpFpFns.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required calls:

FT_register() *Begins the registration operation – acquires 4 images and creates a registration template.*

FT_getFeaturesLen() *Get possible feature lengths*

Code would look something like:

```
//Variable declarations
int regLengthMax;        //Max length of pre registration features
int regLengthMin;        //Min length of pre registration features

FT_REG_OPTIONS regOptions;
FT_BYTE* regFtrs=NULL;
FT_RESULT ftRslt;

//Executable code……

//Get feature lengths and allocate feature buffers
FT_getFeaturesLen(FT_REG_FTR, regOptions, &regLengthMax, &regLengthMin));
regFtrs= new unsigned char[regLengthMax];

//Start registration operation
FT_register(context,0,regLengthMax,regFtrs
        , NULL, &ftRslt);
```

One Touch for Windows C/C++ (DPDevClt.h & dpFtrEx.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required Calls:

FX_extractFeatures() *Get extracted features*

FX_getFeaturesLen() *Gets possible feature lengths*

MC_createContext() *Create matching context*

MC_generateRegFeatures() *Generate registration features*

Code would look something like:

```
//Variable declarations
DATA_BLOB* blob;            //DATA_BLOB declared in wincrypt.h
int preRegLengthMax;        //Max length of pre registration features
int preRegLengthMin;        //Min length of pre registration features
unsigned char* regFtrs;     //Used to store registration features
FT_BYTE* aryPreReg[4];     //Used to store the 4 required pre-registration features
FT_IMG_QUALITY imgQuality;  //Quality of image
FT_FTR_QUALITY ftrQuality;    //Quality of extracted features
FT_BOOL rslt;                 //Result of extraction operation
FT_HANDLE mcContext;
int fingerCount=0;

//Define a window's callback procedure.  This is where you handle fingerprint events.
int CALLBACK wndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
{
   switch (msg)
   {
      case WM_SHOWWINDOW:
            //Start the posting of events to our window
            DPFPStartAcquisition(DP_PRIORITY_NORMAL,*(DevUid)
                        , DP_SAMPLE_TYPE_IMAGE,hWin,FP_EVENT,&phOperation);
```

```
            //Create the matching context.  Make sure to call MC_closeContext later.
            MC_createContext(&mcContext);

            break;
      case FP_EVENT:
           switch (wparam)
           {
                case WN_COMPLETED:
                     //Get a reference to the image
                     Blob= (DATA_BLOB*)  lparam;

                      //Extract pre-registration features
                FX_extractFeatures(fxContext,db->cbData,db->pbData
                                   ,  FT_PRE_REG_FTR, preRegLengthMax
                                   ,  aryPreReg[fingerCount]
                                   ,  &imgQuality,&ftrQuality,&rslt);
                 If ( rslt == FT_OK )
                     ++fingerCount;

                //Reset result for next operation
                rslt = 0;

               //If we have four pre registration prints, generate registration features
                If ( fingerCount==4)
                     MC_generateRegFeatures ( mcContext, 0, 4, preRegLengthMax
                               , preReg, regLengthMax, regFtrs, NULL, &rslt);

                 If (rslt == FT_OK)
                     //Successful Registration
                  break;
           }
            break;
  }
   return 0;
}

//Executable code……

//Get feature lengths and allocate feature buffers before feature extraction
FT_getFeaturesLen(FT_PRE_REG_FTR,0,&preRegLengthMax,&preRegLengthMin);
FT_getFeaturesLen(FT_REG_FTR,0,&regLengthMax,&regLengthMin);
for ( int i=0;i<4;i++) aryPreReg[i]=new FT_BYTE[preRegLengthMax];
regFtrs = new unsigned char[regLengthMax];
```

Verification

This section covers how to verify fingerprints

Gold SDK (dpFpFns.h)

Must implement device initialization (see "Device Initialization" above) and registration before using this section.

Required calls:

FT_Verify() *Begins the registration operation – acquires 4 images and creates a registration template.*

Code would look something like:

```
//Variable declarations
FT_RESULT ftRslt;
FT_VER_SCORE score;
bool featuresChanged;


//Executable code……

//Start verification operation (context, regFtrs declared in previous sections)
FT_verify ( context, false, regFtrs, &featuresChanged,NULL,&score,&ftRslt);

if ( ftRslt == FT_SUCCESS )
   // Successful match
```

One Touch for Windows C/C++ (DPDevClt.h & dpFtrEx.h)

Must implement device initialization (see "Device Initialization" above) before using this section.

Required Calls:

FX_extractFeatures() *Get extracted features*
FX_getFeaturesLen() *Gets possible feature lengths*
MC_createContext() *Create matching context*
MC_verifyFeaturesEx() *Perform match*

Code would look something like:

```
//Variable declarations
DATA_BLOB* blob;            //DATA_BLOB declared in wincrypt.h
int verLengthMax;          //Max length of pre registration features
int verLengthMin;          //Min length of pre registration features
unsigned char* verFtrs;
FT_IMG_QUALITY imgQuality;  //Quality of image
FT_FTR_QUALITY ftrQuality;    //Quality of extracted features
FT_BOOL rslt;                      //Result of extraction operation
FT_HANDLE mcContext;


//Define a window's callback procedure.  This is where you handle fingerprint events.
int CALLBACK wndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)
{
   switch (msg)
  {
      case WM_SHOWWINDOW:
            //Start the posting of events to our window
            DPFPStartAcquisition(DP_PRIORITY_NORMAL,*(DevUid)
                        , DP_SAMPLE_TYPE_IMAGE,hWin,FP_EVENT,&phOperation);

            //Create the matching context.  Make sure to call MC_closeContext later.
            MC_createContext(&mcContext);
```

```
                break;
        case FP_EVENT:
                switch (wparam)
                {
                        case WN_COMPLETED:
                                //Get a reference to the image
                                Blob= (DATA_BLOB*)  lparam;

                                //To generate registration features see “Registration” in the above
                                //section.

                                //Extract verification features
                        FX_extractFeatures(fxContext,db->cbData,db->pbData
                                                , FT_VER_FTR, verLengthMax
                                                , verFtrs
                                                , &imgQuality,&ftrQuality,&rslt);

                        if ( rslt== FT_OK )
                        //Verify features, some parameters declared from previous sections
                        MC_verifyFeaturesEx( mcContext, regLengthMax, regFtrs
                                        , verLengthMax, verFtrs, false, NULL, NULL, NULL
                                        , NULL, &rslt);
                        if ( rslt == FT_OK )
                        // Verification successful (prints match)

                        break;
                }
                break;
  }
   return 0;
}

//Executable code……

//Get feature lengths and allocate feature buffers before feature extraction
FT_getFeaturesLen(FT_VER_FTR,0,&verLengthMax,&verLengthMin);
verFtrs = new unsigned char[verLengthMax];
```