U3 Platform 1.0 SDK

Developer Guide

Version 1.0
September 2005
Revision 1.0

DOCUMENT CONTROL INFORMATION

Document No.: 04-UM-0605-10

	Title	Name	Date
Issued by:	U3 Chief Architect	Daniel Goodman	September 2005
Edited by:			

Table of Contents

1 Introduction

This guide describes how to use the U3 Platform SDK to develop and deploy U3 applications. This SDK provides directions on how to develop applications for the three different types of U3 applications.

The document also provides instructions for the Device API (DAPI), especially with regards to the event model and includes examples that illustrate session management, working with cookies and protected areas.

- Section 1 introduces the U3 Platform SDK, including the U3 Platform SDK components, application types, how to deploy and launch applications, and provides an overview of U3 devices.
- Section 2 gives an overview of the DAPI, including the DAPI programming model and the DAPI functions.
- Section 3 discusses the DAPI application development environment.
- Section 4 details the DAPI programming event model.
- Section 5 includes a DAPI programming sample.
- Section 6 describes how to write cookies to a U3 device.
- Section 7 explains how to use DAPI to access private or protected areas of a device.

1.1 U3 SDK Components

The **U3 SDK** provides the tools to assist you in developing and deploying U3 applications and includes the following components:

- **Application Deployment Guide** – This document describes how to create and package U3 Launchpad applications that are installed and run from U3 smart devices.
- **DAPI Package** – The Device API (DAPI) is used to create U3 Aware applications and can also be used to extend Launchpad applications to work directly with the U3 device. The DAPI package including the following components:
    - **u3dapi10.dll** – DAPI DLL containing the DAPI functions
    - **u3dapi10.lib** – Library file required to link to the DLL
    - **u3dapi10.h** – DAPI header file
    - **u3dapi10.def** – Definition file listing all DAPI functions and their ordinals
- **API Reference Guide** – This reference document includes a detailed description of the U3 DAPI functions.
- **Developer Guide** – This document describes how to use the U3 SDK to develop and deploy applications.
- **Release Notes** – These notes include the latest information available at release.

The following kits are separate from the SDK and are not covered in this document.

- **U3 Hardware Developer Kit (HDK)** – contains the U3 device services specification and tools for hardware manufacturers to develop U3 compliant devices. For availability and information regarding the HDK, please contact U3 at the email or address provided at the end of this document.
- **U3 smart Logo Compliance Kit** – in order to ensure that software applications can be deployed on U3 smart drives, U3 LLC has developed the ***U3 smart Application Compliance Program***. This includes a set of test criteria, which must be followed before a software application can be certified and before the right to display the U3 smart logo can be granted. The kit is available from http://www.u3.com/developers/.

1.2 Overview of U3 Applications

There are three types of U3 applications that can be developed and deployed.

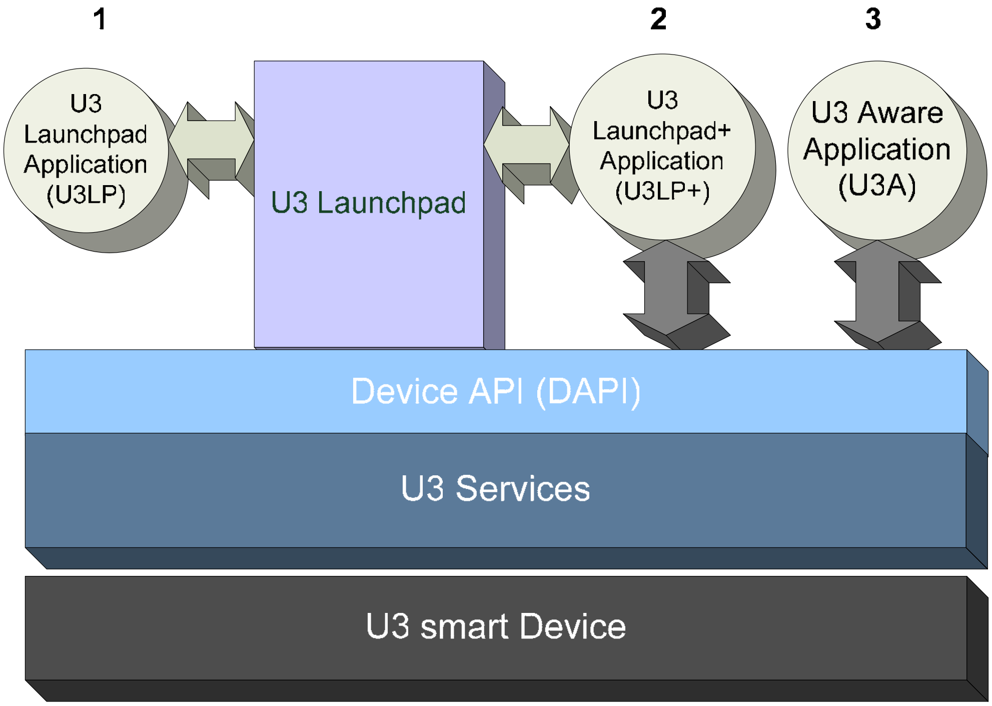
1
2
3
U3 Launchpad Application (U3LP)
U3 Launchpad
U3 Launchpad+ Application (U3LP+)
U3 Aware Application (U3A)
Device API (DAPI)
U3 Services
U3 smart Device

Figure 1: U3 Application Types and SDK Architecture

- **U3 Launchpad (U3LP) Application** – A Windows application that is resident on the U3 smart device and whose lifecycle is managed by U3 Launchpad. Because the application is installed onto the device, all configurations, user preferences and files travel with the device. For further information and a description of the U3 Launchpad, refer to the *Application Deployment Guide*.
- **U3 Launchpad+ (U3LP+) Application** – A U3 Launchpad application that ships with the DAPI DLL allowing it to communicate directly with the U3 device and leverage the advanced device features. This is necessary for example, to work with cookies, or to establish the

device unique ID. For further information, refer to the *Application Deployment Guide*, *SDK Developer Guide* and the *DAPI Reference Guide*.

- **U3 Aware (U3A) Application** – A host-based application that uses the DAPI DLL to detect and communicate with U3 Devices as they are inserted into the host. A U3A application is a standard Windows application, with a standard installation and startup procedure, which has been extended to support U3 devices. For further information, refer to the SDK *Developer Guide* and the *DAPI Reference Guide*. Examples of U3 Aware applications are:
    - A Windows service, such as a Login service, that uses U3 smart device as tokens.
    - A Windows application, such as a Backup tool, which has U3 device support, enabling data to be securely backed up to a password protected private area on the device.

1.3 Deploying and Launching U3 Applications

Table 1 describes the components of the SDK required to develop, deploy and launch the various types of U3 applications.

Table 1: SDK Component Usage

Application Type	Relevant Documentation	Required Components	Installation Location
U3 Launchpad	Application Deployment Guide		U3 Launchpad
U3 Launchpad+	Application Deployment Guide SDK Developer Guide DAPI Reference Guide	DAPI Package*	U3 Launchpad
U3 Aware	SDK Developer Guide DAPI Reference Guide	DAPI Package*	Host

* U3LP+ and U3A Applications require the DAPI package for development since it contains the libraries for interacting with the device. These applications must ship with the DAPI DLL.

Note: The *U3 smart Application Certification Self-Test Criteria* document, included in the U3 Logo Compliance Kit, should be consulted before designing and deploying a U3 application.

1.3.1 Deploying and Launching U3 Applications

The U3 applications are deployed and launched as follows:

- **U3LP and U3LP+ Applications** – These applications are resident on the device and must be deployed using the procedures described in the Application Deployment Guide. The applications are launched and managed by the Launchpad
- **U3A Applications** – These applications are installed on the host and are deployed and launched like any other Windows host based application.

1.4 Overview of U3 Devices

A U3 smart device is a configurable USB flash drive (UFD) that currently supports up to two independent memory domains (support for additional domains will be provided in future releases). Each memory domain can be configured as one of the following:

- Persistent read-only domain that emulates a CD-ROM
- Removable domain (removable disk)

1.4.1 Persistent Read-Only Domains (Virtual CD-ROM)

A persistent read-only domain is a domain that emulates CD-ROM functionality. The domain provides full CD-ROM functionality, including auto-run. The CD-ROM domain can be used to store pre-defined data on a U3 device that cannot be changed by a user.

The virtual CD-ROM domain is created, configured and managed using the U3 Tool. The tool allows ISO images to be burnt to the virtual CD-ROM domain. This domain can also be locked with a key that must be provided to reconfigure the domain or burn a new image. For information about handling the CD domain, refer to the *Hardware Development Kit* documentation.

1.4.2 Removable Domain Areas

A removable domain can be configured to contain a public and/or private area. A public area acts like a standard USB flash device. The private area implements password-based access control and it is not accessible until a successful login has occurred with the correct password. The login and logout functions are accessible through DAPI.

Users can view either the public or the private area on a removable domain at one time, as follows:

- If the domain contains only a public area, it becomes visible immediately when the device is inserted into the USB port.
- If the domain contains only a private area, it becomes visible only after logging in through DAPI. Until then, it behaves like a device with no media inserted.
- If the domain contains both public and private areas, the public area is initially visible, for example, on drive E. When users log into the private area, it replaces the public area and only the private area is visible on drive E. When users log out of the private area, the public area again becomes visible on drive E, replacing the private area.

1.4.3 Sample Configurations

U3 smart devices support the following configurations:

- Removable (all public) + CDROM
- Removable (all private) + CDROM

1.4.4 Cookies

Cookies are used to store data in a special hidden data area of the device. They are accessible only by DAPI. Applications can store private data in this hidden area. Cookies are addressed by a unique section and entry pair, in the same way that entries in Windows INI files are addressed.

2 Overview of the Device API

This section presents an overview of the Device API (DAPI). DAPI is only required by those applications that interact with the device, that is U3 LaunchPad+ and U3 Aware applications.

Note: Most U3 applications are standard U3 Launchpad applications and do not require the use of DAPI.

The Device API allows applications to communicate with U3 compatible devices and also query information about U3 compatible devices, leveraging advanced device features.

The DAPI DLL is a C-based DLL that can be used by all programming languages that support Unicode. The dll can be implicitly or explicitly loaded.

2.1 DAPI Programming Model

This section provides a high-level description of the DAPI programming model.

2.1.1 Event Model

DAPI is event-driven based upon device insertion, update and removal events. A detailed description of the DAPI event model is presented in Section 4. The main points of the event model are as follows.

- DAPI notifies the application when device events occur through a callback function that the developer must implement.
- Device events are: device inserted (connect), device removed (disconnect) and U3 smart device specific update and configuration change events.
- The different U3 smart device update events occur when an operation has been performed on the device or when Windows has updated information about the device configuration. For example, the update event occurs when Windows has assigned a drive letter to a domain on the device. Typically when a multi-domain device is inserted, each domain is assigned a drive letter one domain at a time.
- The device handle is included with all DAPI events and is passed in the callback to the application. DAPI functions use this handle to interact with the device.

2.1.2 Device Support

DAPI recognizes both U3 and non-U3 compliant devices. Not all functions are available for non-U3 compliant devices.

Applications can specify filters so that only events for specific devices are received. For more information, see Section 3.3, dapiRegisterCallback of the *DAPI Reference Guide*.

2.2 Error Codes

DAPI uses the Windows HRESULT system to return error and success codes. DAPI uses the FACILITY_ITF feature to define additional DAPI-specific errors that are defined in u3dapi.h and that are described in Section 7.1 of the *DAPI Reference Guide*.

2.3 DAPI Functions

The DAPI functions allow applications to interact with a device. They are divided into the following usage groups:

- **Session Management Functions** – Allow you to set up and terminate a DAPI session, register callback functions and obtain a device handle.
- **Device Functions** – Used to query basic information about the device such as device capability, device identification parameters and device domain configuration for all USB devices.
- **Private Area Functions** – Gets private area configuration information, logs in and out of the private area and sets a private area password.
- **Cookie Functions** – Reads and writes binary and text cookies.

For more information about the DAPI functions, see the *U3 DAPI Reference Guide.*

2.4 Privileged Operating Modes

DAPI supports administrator and non-administrator (user) privilege modes. When in non-administrator mode, also referred to as user mode, some DAPI features are not available. The HRESULT code DAPI_E_CMD_NOT_SUPPORTED_IN_NON_ADMIN_MODE is returned denoting that these features are not currently available when these functions are called in user mode. DAPI will automatically switch the device to user mode when required.

3 Building DAPI Applications

3.1 SDK Development Environment

The DAPI DLL uses C-based function calls. Any Win32 based language and development environment that supports Unicode strings can be used to develop DAPI applications including, but not limited to:

- Microsoft® Visual Studio® Version 6, .Net 2003 and newer versions
- Borland® C and Delphi

3.2 Shipping the DAPI DLL

All applications that use DAPI (U3LP+, U3A) must include the DAPI DLL together with the application. You also must ensure that the specific DAPI DLL is used by the specifying full path to the DLL. That will ensure that your applications will work with the specific version of DAPI.

4 Introduction to the DAPI Programming Event Model

This section describes the DAPI event model for a U3 device.

4.1 Device Events

The following events are generated by DAPI for all supported USB devices:

Table 2: Callback Events Generated for all Devices

Event Name	Value	Description
DAPI_EVENT_DEVICE_CONNECT	0x01	A supported USB device has been connected.
DAPI_EVENT_DEVICE_DISCONNECT	0x02	A device has been disconnected. The device handle is no longer valid.

The following events are generated for U3 smart devices:

Table 3: Callback Events Generated for U3 smart Devices

Event Name	Value	Description
DAPI_EVENT_DEVICE_UPDATE	0x03	Information about a domain on the device has been updated. A domain has been assigned a drive letter.
DAPI_EVENT_DEVICE_LOGIN	0x04	A private area on the device has been successfully logged into.
DAPI_EVENT_DEVICE_LOGOUT	0x05	A private area on the device has been successfully logged out of.
DAPI_EVENT_DEVICE_WRITE_PROTECT_ON	0x06	Write protect has been enabled on a removable domain and the domain is now read only.
DAPI_EVENT_DEVICE_WRITE_PROTECT_OFF	0x07	Write protect has been disabled on a removable domain.
DAPI_EVENT_DEVICE_RECONNECT	0x08	The device has been refreshed by Windows. The current device configuration is no longer valid.
DAPI_EVENT_DEVICE_NEW_CONFIG	0x09	The device has been reconfigured. The current device configuration is no longer valid.

4.2 Multiple Domain Initialization Process

Windows processes domains on a device sequentially, that is each domain is assigned a drive letter and its device information is updated in the host. When a device with multiple domains is inserted into the machine, events are received in the following sequence: initially, a connect event will be received when the first domain is initialized and assigned a drive letter. Subsequently, an update event will be received as each additional domain is initialized and assigned a drive letter.

Until the device is completely initialized the dapiQueryDomainInformation (see *DAPI Reference Guide*, Section 4.3) function will return a S_FALSE value. The DAPI_DI_NOT_READY flag will be set for each domain that has not yet been assigned a drive letter.

Note that if a session or callback is unregistered, a disconnect event is not sent for associated devices.

4.3 Callback Function Event Information

The SDK uses a callback function to provide information to the application about the occurrence of events described in the previous section; the occurrence of an event causes the callback function to be called. The event that occurred for a device and the device handle are passed as parameters of the callback function.

When an event occurs the following data is sent to the callback function:

- Device handle: This handle references the device. It can be used with all other DAPI functions to perform actions and query information about the device. If the handle is used after the device is removed, an E_HANDLE error will be returned.
- eventType: Inserted, removed or U3 specific update event. See Section *Device Events*, for more information.
- pEx: When the callback is registered, a pointer to arbitrary data can also be provided by the developer. This pointer is sent to the callback every time an event occurs that is associated with the registered callback instance. The pointer is provided as a void * and must be cast to the appropriate data type.

The callback function has the following form:

```
void _stdcall dapiCallback( HDEVICE hDev,
                            DWORD   eventType,
                            void *  pEx)
```

4.3.1 Registering for Events

You register for a device event by registering a callback function (using dapiRegisterCallback, see Section 3.3 of the *DAPI Reference Guide*). You can register a callback function for all devices or for a specific device by specifying the pszConnectionString parameter as NULL or a specific device serial number, respectively.

4.3.2 Obtaining the Connection String for a Device

For U3LP+ applications, the Launchpad provides the connection string for the device from which the application is running. The connection string resides in the U3 Environment Variable U3_DAPI_CONNECT_STRING (see Section 9.4.4 in *U3 Runtime Variables* of the *Application Deployment Guide*.) Alternatively, you can use the dapiQueryDeviceInformation function to obtain the serial number string of a device, which can be used as the connection string parameter for a new callback registration.

4.4 Multiple Registration

You can register a callback function more than once with the same or different connection string. Each time a callback is successfully registered a handle is provided to reference that instance of the callback in the future. The handle is used to unregister the callback function.

The pEx parameter can be used when performing multiple registrations to aid in tracking each instance or associating a callback with a class instance or other data.

5 A Simple DAPI Program

This section presents a simple example that includes the basic steps in creating a DAPI application.

5.1 Getting the Device Information

This simple MFC Dialog application uses DAPI to display information about device that is inserted. It detects the first device inserted and prints out information about that device. This application shows how to register a callback function, get the device handle and use it to get basic information about the device. For more information about the DAPI functions used in the application, see Chapter 3 of the *DAPI Reference Guide*.

The steps of the application are:

1. Create a MFC Simple Dialog application, called DAPIMFC. Set the project type to be Unicode.
2. Include `u3dapiXX.h` in the code and include `u3dapiXX.lib` in the project.
3. Create a class to hold the DAPI functions. We will call it DapiBase. Change the application’s dialog class to subclass DapiBase as well as CDialog

```
class CDAPIMFCDlg : public CDialog, CDapiBase
```

4. Implement the callback function that you will register to receive DAPI events. As DAPI functions should not be called from within the callback, this function will send a Windows message back to the application. We will use the pEx value to pass the handle of the application’s main window.

```
void _stdcall CDapiBase::Callback(HDEVICE hDev, DWORD eventType, void* pEx)
{
   if(NULL != pEx)
          PostMessage((HWND)pEx, WM_USER + eventType, hDev, 0);
}
```

5. In the DapiBase define a function that creates a DAPI session. A DAPI application must begin by creating a DAPI session (using dapiCreateSession) that enables access to the DAPI functions and initiates listening for device events. Once the session is created, register the callback function, passing in the hWnd parameter as pEx.

```
HRESULT CDapiBase::RegisterForEvents(HWND hWnd, wchar_t* pszConnctionString)
{
   HRESULT res = S_FALSE;
          // make sure that the session or callback does not exist
   if(ILLEGAL_HSESSION != m_hSession || ILLEGAL_HCALLBACK != m_hCallBack)
          return res;
          //create a session
   if((res = dapiCreateSession(&m_hSession)) == S_OK)
   {      //register a callback on our session
          if(S_OK != (res =  dapiRegisterCallback(m_hSession, pszConnctionString,
                                    (DAPI_CALLBACK)(Callback),(void *) hWnd,
                                    &m_hCallBack)))
                 {  UnregisterForEvents();  }
     }
   return res;
}
```

6. When a device is inserted a message will be sent to the main window via the callback function. Create a message handler that will call dapiQueryDeviceInformation using the device handle passed by the callback function. This will call the DAPI function outside of the callback function.

```
BEGIN_MESSAGE_MAP(CDAPIMFCDlg, CDialog)
      . . .
   ON_MESSAGE(WM_UFD_CONNECTED, OnDeviceConnected)
   ON_MESSAGE(WM_UFD_DISCONNECTED, OnDeviceDisconnected)
   ON_MESSAGE(WM_UFD_UPDATED, OnDeviceUpdated)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

. . .

LRESULT CDAPIMFCDlg::OnDeviceConnected(WPARAM hDevice, LPARAM unused)
{
GetDlgItem(IDC_STATIC_STATUS)->SetWindowText(_T("Device Connected"));

devInfo info;

   //get the device information
if(dapiQueryDeviceInformation((HDEVICE)hDevice, &info) == S_OK)
   {
      CString strUnique; // turn UniqueID binary array into a string
      for (size_t i = 0; i < sizeof(info.uniqueID); ++i )
      {
         CString tmpStr;
         tmpStr.Format(_T("%02x "), (int)info.uniqueID[i]) ;
         strUnique += tmpStr;
      }
   GetDlgItem(IDC_EDIT_UNIQUE)->SetWindowText((LPCTSTR)strUnique);
   GetDlgItem(IDC_EDIT_SERIAL)->SetWindowText((LPCTSTR)info.serialNumber);
   GetDlgItem(IDC_EDIT_FW_VER)->SetWindowText((LPCTSTR)info.FWVersion);
   GetDlgItem(IDC_EDIT_VENDOR_STR)->SetWindowText((LPCTSTR)info.vendorString);
   GetDlgItem(IDC_EDIT_PRODUCT_STR)->SetWindowText((LPCTSTR)info.productString);
   }

return 0;
}
```

7. In the OnInitDialog() function, register the callback function. Use the U3 Launchpad Environment variable U3_DAPI_CONNECT_STRING to determine the connection string for the device.

```
typedef CStringT< TCHAR, StrTraitMFC< TCHAR > > CMyString;
CMyString connectionString;
connectionString.GetEnvironmentVariable(_T("U3_DAPI_CONNECT_STRING"));

// Register the callback, using the safe windows handle and the
// connection string. The CS will be NULL if not defined.
// This calls the parent class CDapiBase::RegisterForEvents(...)

RegisterForEvents(GetSafeHwnd(),connectionString.GetBuffer(256));
```

8. Destroy the DAPI session. A DAPI application should always be terminated by destroying the session (dapiDestroySession). All callback and device handles associated with the DAPI session

are also destroyed and invalidated.

```
void CDapiBase::UnregisterForEvents()
    {
    if(ILLEGAL_HCALLBACK != m_hCallBack)
            dapiUnregisterCallback(m_hCallBack);

    m_hCallBack = ILLEGAL_HCALLBACK;

    if(ILLEGAL_HSESSION != m_hSession)
            dapiDestroySession(m_hSession);

    m_hSession = ILLEGAL_HSESSION;

    }
```

5.2 Get the list of domains on a device

The following is a code example showing how to query the domain information for a U3 device. This function will not act until all domains have been assigned drive letters. See the `CDAPIMFCDlg::UpdateDrives()` function for the full example. See Section 4.3 in the *DAPI Reference Guide*, dapiQueryDomainInformation.

1. U3 smart devices support a variable number of domains. To determine the number of domains on the device call the dapiQueryDomainInformation function with a NULL domainInfo pointer and an nCount variable with a value of 0.

```
WORD nCount = 0;
// Read the number of drives to allocate the array
if(dapiQueryDomainInformation(hDevice, NULL, &nCount) == S_OK)
```

2. If the function returns S_OK, the use the update nCount value to create an array of domainInfo structures. Below is an example, using a STL vector template.

```
//Allocate domains vector
typedef std::vector<domainInfo> domain_arr;
domain_arr vec(wDomains);
```

3. Call the function again with a domainInfo array. If the function returns S_OK then it has succeeded and all domains have been assigned drive letters. If the function returns S_FALSE, then not all the domains have been assigned drive letters. Wait until all drives have been assigned a drive letter, i.e. wait for S_OK value

```
if(dapiQueryDomainInformation(hDevice, &(vec[0]), &wDomains) == S_OK)

        {
```

4. Loop through each domain displaying the required information about each domain, or example a list of drive letters

```
        CString strDrives;
        for(size_t n=0; n <vec.size();++n)
                {
                strDrives += vec[n].szPath;
                strDrives += _T(" ");

                }
```

6 Working with Cookies

This section describes how to write cookies to a U3 device. Cookies are used to store data in a special hidden data area of the device. They are accessible only by DAPI. Applications can store private data in this hidden area.

Cookies are addressed by a unique section and entry pair, in the same way that entries in Windows INI files are addressed.

See the *Standard Cookie Functions* in Chapter 6 of the DAPI Reference Guide for limits on cookie-related values.

The *U3 smart Application Certification Self Test Criteria* document, part of the U3 Logo Compliance Kit, describes the acceptable use of cookies on U3 smart devices.

The following sections show how to use DAPI to write text cookies to a device.

6.1 Reading and Writing an Text Cookie

The following example writes a text cookie to a device.

The steps of the application are:

1. Test that the device supports cookies by querying the device capabilities using the dapiQueryDeviceCapability(hDev, DAPI_CAP_U3_COOKIE) call.
2. Determine the size of the cookie data and if the cookie exists. Call dapiReadTextCookie with a NULL szValue entry and an nLength value = 0. The function will either return S_OK or DAPI_E_BUFFER_TOO_SMALL.

```
DWORD nCookieLen = 0;
HRESULT res = dapiReadTextCookie(hDevice,pszSection,pszEntry, NULL,
                                 &nCookieLen);
switch(res)
      {
      case DAPI_E_BUFFER_TOO_SMALL:
      case S_OK:

      {
```

3. Use the returned length value to allocate a string and call dapiReadTextCookie with updated nLength value an szValue buffer.

```
                  // OK read cookie data
                  // nCookieLen includes final \0 char
            wchar_t * pszTextBuffer = new wchar_t[nCookieLen];
            wmemset(pszTextBuffer,_TCHAR("\0"),nCookieLen);
            if(S_OK == dapiReadTextCookie(hDevice,pszSection,pszEntry,
                                          pszTextBuffer, &nCookieLen))
                  {
                  . . .
                  }
            delete [] pszTextBuffer;
            break;

            }
```

4. Alternatively the cookie may not exist, so an alternate value should be generated

```
            case DAPI_E_DATA_NOT_FOUND:
                  // cookie does not exist
                  // create alternate or default data
                  . . .
                  break;
            default:
                  // general error reading cookie
                  // check and analyze return error code
                  . . .
                  break;

            }
```

5. Write updated cookie using dapiWriteTextCookie.

```
      dapiWriteTextCookie(hDevice,pszSection,pszEntry,
                             (LPCTSTR)GenerateCookieData());
```

Note: A cookie can be written immediately after getting device handle. There is no need to wait for drive letters to be assigned to domains.

6.2 Implementation

```
       // This function checks for a cookie on the device called
       // [AppName]
       //    LastSeen=
       //
       // NOTE According to self test criteria this cookie must be
       // removed from the device when the application is uninstalled!!!
       // Implement in deviceUninstall a method to delete this cookie.

void CheckandSetCookie(HDEVICE hDevice)
       {

             // check that the device supports cookies
       if(S_OK != dapiQueryDeviceCapability(hDevice,DAPI_CAP_U3_COOKIE))
             {
             SetWindowText(_T("Device does not support cookies"));
             return;
             }


       wchar_t * pszSection = _T("AppName");
       wchar_t * pszEntry   = _T("LastSeen");

       // test to see if cookie exists
       DWORD nCookieLen = 0;
       HRESULT res = dapiReadTextCookie(hDevice,pszSection,pszEntry,
                                        NULL,
                                        &nCookieLen);
       switch(res)
           {
           case DAPI_E_BUFFER_TOO_SMALL:
           case S_OK:
                {
                // OK read cookie data
                // nCookieLen includes final NULL char
                wchar_t * pszTextBuffer = new wchar_t[nCookieLen];
```

```
         // clear the buffer
         wmemset(pszTextBuffer,_TCHAR("\0"),nCookieLen);

         if(S_OK == dapiReadTextCookie(hDevice,pszSection,pszEntry,
                                              pszTextBuffer,
                                              &nCookieLen))
             {
             SetWindowText(pszTextBuffer);
             }
         delete [] pszTextBuffer;
         break;
         }

    case DAPI_E_DATA_NOT_FOUND:
         // cookie does not exist
         SetWindowText(_T("This is the first time we have seen the device"));
         break;

    default:
         // general error reading cookie
         SetWindowText(_T("Error reading cookie data"));
         break;
    }

// now set the cookie with the latest time string
CTime time(CTime::GetCurrentTime());
dapiWriteTextCookie(hDevice,pszSection,pszEntry,
                   (LPCTSTR)time.FormatGmt(_T("%#c")));

}
```

7 Working with Private Areas

This section shows how to use DAPI to access private or protected areas of a device.

The protected or private area is a secure area within a removable domain that supports password-based access control. It is accessible only by login through DAPI

The private area functions are available to be used with devices configured using the U3 Hardware Development Kit (HDK)

Note: U3 Launchpad+ and U3 Aware applications working in conjunction with LaunchPad-managed devices must not call private area functions. This may result in data loss or application instability.

7.1 Reading and Writing to a Private Area

The following example demonstrates how to log into a private area of a specific device. The user can read or write to it and then logs out again. The example also shows how to handle a login error and to detect when the protected area has been permanently locked because the maximum number of accesses, MaxNOA, was exceeded. In this application we use a serial number of a specific device as the pszConnectionString to receive connection events only from that device. For more information, see Chapter 5 of the *DAPI Reference Guide*.

The example will also show how to query the domain information and test when the device is not yet fully initialized.

Note that the initial steps of the application, up to and including implementing and registering the callback function and the last step of destroying the session are omitted in this example.

The steps of the application are:

1. Test that the device supports private areas
   dapiQueryDeviceCapability (hDev, API_CAPI_U3_PRIVATE_AREA).
2. Call dapiQueryDomainInformation to determine the path to removable domains and check that the DAPI_DI_REMOVABLE_PRIVATE flag is set. This indicates that the domain has a private area configured.
   While the typeMask for the domain includes the DAPI_DI_NOT_READY flag, wait until the next device update event. When the flag is no longer set, log into the private area.
3. Use dapiGetPrivateAreaInfo to get the password hint and test that the domain is not locked out (the domain is locked when CurrNOA = MaxNOA). Store the currNOA (number of attempts) value for the domain. This will be used later to handle any errors.
4. Get the path to the domain and the password. Display the password hint, if needed.
5. Call dapiLoginPrivateArea to login to private area.

   If the login failed with a DAPI_E_COMM_FAIL, test the updated CurrNOA value.

   - If CurrNOA has increased, the password was incorrect.
   - If CurrNOA is now equal to Max NOA, the domain is permanently locked and a reset password (area format) must be performed (using dapiSetPrivateAreaPassword, see Section 5.4 of the *DAPI Reference Guide*).

- If CurrNOA has not increased, the communications with the device have failed.
- After performing the appropriate corrective action, call dapiLoginPrivateArea again.

6. On user command, call dapiLogoutPrivateArea to logout from the private area.

U3 Platform 1.0 SDK, Developer Guide

U3
303 Twin Dolphin Drive, 6th Floor
Redwood City, CA 94065
1-800-837-3654, info@u3.com

For more information, visit www.u3.com