yubico
the key to the cloud

Yubikey Configuration COM API

YubiKey device Windows configuration component

Version: 2.2

May 22, 2012

Introduction

Yubico is the leading provider of simple, open online identity protection. The company's flagship product, the YubiKey®, uniquely combines driverless USB hardware with open source software. More than a million users in 100 countries rely on YubiKey strong two-factor authentication for securing access to computers, mobile devices, networks and online services. Customers range from individual Internet users to e-governments and Fortune 500 companies. Founded in 2007, Yubico is privately held with offices in California, Sweden and UK.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Yubico shall have no liability for any error or damages of any kind resulting from the use of this document.

The Yubico Software referenced in this document is licensed to you under the terms and conditions accompanying the software or as otherwise agreed between you or the company that you are representing.

Trademarks

Yubico and YubiKey are trademarks of Yubico Inc.

Contact Information

Yubico Inc
228 Hamilton Avenue, 3rd Floor
Palo Alto, CA 94301
USA
info@yubico.com

Contents

© 2012 Yubico. All rights reserved.

1 Document Information

1.1 Purpose

The purpose of the device configuration component is to allow easy integration of Yubikey configuration functionality into third-party applications in Microsoft Windows environments.

The component is not intended as a “stand-alone” configuration utility and provided sample code is provided as templates only. In the case a configuration tool is needed, please refer to the Yubikey Configuration Utility.

1.2 Audience

Programmers and systems integrators.

1.3 Related documentation

- YubiKey Configuration Utility – The Configuration Tool for the YubiKey
- The YubiKey Manual – Usage, configuration and introduction of basic concepts
- Yubico online forum – http://forum.yubico.com

1.4 Document History

Date	Version	Author	Activity
2007-08-23	0.1	JE	First draft
2009-09-11	2.0	JE	Updated for YubiKey 2.0 support
2010-03-05	2.1	JE	Updated for Yubikey 2.1 support
2010-06-09	2.2	JE	Updated for Yubikey 2.2 support
2012-05-22	2.3	ZD	Changed document template

1.5 Definitions

Term	Definition
YubiKey device	Yubico’s authentication device for connection to the USB port
USB	Universal Serial Bus
HID	Human Interface Description. A specification of typical USB devices used for human interaction, such as keyboards, mice, joysticks etc.
API	Application Programming Interface
COM	Component Object Model – a component based programming model developed by Microsoft.
ActiveX	A definition on top of COM, primarily targeted for user interface extensions in a Web-browser.
Callback	Function in the User Program called by the API
AES	Advanced Encryption Standard. A NIST approved symmetric encryption algorithm.
OATH-HOTP	Initiative for Open Authentication (RFC 4226)
HMAC	Hash-based Message Authentication Code
SHA-1	Secure Hash Algorithm 1

2 Introduction

This document is intended as a reference for software developers who are integrating device personalization features with Yubico’s YubiKey.

Yubico provides a high-level device configuration component based on Microsoft’s COM/ActiveX technology. With this approach, a wide range of programming languages, scripting environments and software packages can perform device configuration operations using a single unified interface.

This document assumes a conceptual knowledge about the Yubico YubiKey, its functions and intended usage.

The document further assumes working knowledge of COM, and at least one programming language that supports COM components. Provided examples are developed with Microsoft Visual Studio .NET 2003.

The component is designed for the Microsoft Windows Win32 environment and works with Windows versions from Windows 2000 and onwards. Integration with Microsoft’s .NET programming model is straightforward. Refer to appropriate .NET documentation of integration of COM components.

2.1 Programming model

By using COM/ActiveX, most programming languages and third-party tools can interface to the YubiKey via the YubiKcom Component through a uniform interface with standard data representation. In other words, the component can be used by any programming language and development tool supporting COM/ActiveX. Examples include Visual C++, Visual Basic, Delphi, Microsoft Office (VBA) and Internet Explorer VB Script.

2.2 Yubikey 2.x support

The component has been upgraded to support Yubikey 2.x. In order to keep the binary interface, selection of configuration 1 or 2 is done by the means of a flag property rather than providing a new method or property for it.

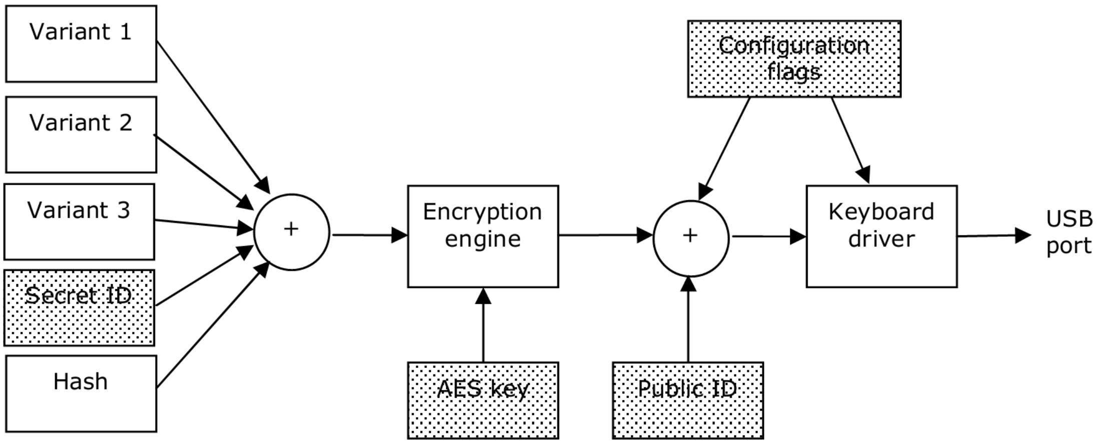
2.3 Limitations

The component is intended for trials and small-scale deployments, where devices are programmed one by one. The component is intended to work without requiring external hardware, which as such implies that attachment and removal of USB devices will be handled by the operating system. Enumeration of an inserted USB device takes some 2-3 seconds on a typical Windows XP installation. This time practically limits the overall throughput of a personalization process.

Consult Yubico for volume configuration, which can be performed at significantly higher speed with support of external Yubico proprietary hardware.

3 YubiKey concepts

The YubiKey is an authentication token capable of generating time-variant One Time Passwords (OTPs). The OTPs are made up of a number of fields where some are time-variant and sequenced in such a way that an OTP is guaranteed to be unique. Before the OTP is sent, it is encrypted with a 128-bit symmetric AES key.



The shadowed entities in the schematic denote fields that can be configured by the configuration component.

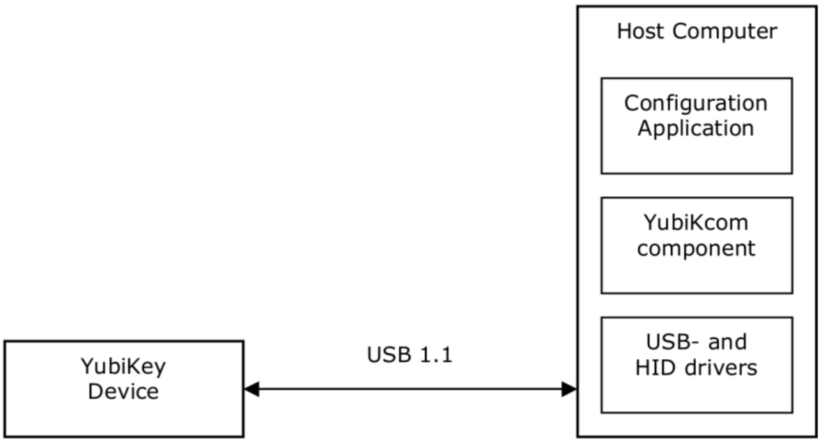
The unique approach with the YubiKey is that it identifies itself as a keyboard peripheral and that the encrypted OTP is transmitted as a series of emulated keystrokes. Thereby, the key can be used in any environment supporting USB/HID keyboards without the need of any client-side drivers.

The service validating an OTP has the same AES key and uses this to decrypt the received OTP. Fields in the decrypted OTP is then used to determine of the OTP is valid.

With the introduction of Yubikey 2.2, features for two-way communication, i.e. “challenge-response” have been introduced. This requires support by a client application. Yubico provides a COM component for such interaction in the Yubico Client API.

3.1 Configuration

Before a YubiKey device can be used, it needs to be configured. When manufactured, the device is in an un-configured state and it can therefore not generate any OTPs.



The configuration is performed by a keyboard back-channel, where configuration data is sent to the device via HID reports. The complexity of implementing this scheme is fully taken care of by YubiKcom component, where the required functionality is exposed as an Application Programming Interface (API).

3.2 Device password

In order to allow a configured device to be re-configured, a security mechanism is provided. Access to configuration properties of the device can be restricted by the means of a 48-bit password. If set, any operation involving update of the configuration image requires the correct 48-bit password to be supplied.

From a security standpoint, this approach is targeting destructive attacks as the interface does not provide any means of retrieving secret information. The configuration interface is write-only, and the term destructive attack refers to an opponent that wants to erase or modify the settings of a device.

By design, there is a “throttle” on update operations, which effectively limits the update throughput to around ten operations per second. Therefore, an exhaustive search for a static configuration password with 248 bits would require an effort in the region of 1013 seconds, which means some 890,000 years. By the singularity of the device itself, an exhaustive search cannot be parallelized.

These figures are however somewhat academic by nature, as even if such an exhaustive search would be successful, the security of the YubiKey scheme is not compromised. Again, a broken configuration password would only open up for a destructive operation that from a system’s point of view would turn the device useless.

3.3 Device public identity

The device could be assigned a static identity, which is a static binary string that is sent in clear text to uniquely identify the device. The service validating an OTP string from a device would typically extract this part to retrieve the appropriate encryption key for the particular YubiKey in question.

The public identity has no meaning for the device itself and if used, the sequencing and exact meaning of the public identity is defined by the device issuer.

3.4 Device encryption key

All devices have a 128-bit symmetric encryption key, which is used when OTPs are generated. Given the nature of symmetric keys, keeping the encryption key secret is the absolute key to keeping the scheme secure.

3.5 Device secret identity - UID

The device public identity is optional and as a static string can be intercepted and/or modified, an additional level of identity is provided. This “secret” identity, or UID, can only be retrieved by a party having the encryption key for a particular device.

The exact meaning of the secret identity is defined by the device issuer, which means that the secret identity does not need to be secret as such. It may be the same as the public identity. The UID should be unique although the scheme works even if it is not.

© 2012 Yubico. All rights reserved.

4 Using the API

The YubiKey configuration API is provided as a COM/ActiveX component, where methods and properties are exposed. Asynchronous notifications are provided by the means of events.

The component follows the data types defined by the COM Automation model which provides maximum flexibility and interoperability.

A COM component needs to be registered with the operating system in order to be used. This is typically done by an installation tool, where the Self registration function is used. The component can be explicitly registered with the regsvr32 utility: Type **regsvr32 yubikcom.dll** under the Start/Run menu.

Integration of COM components varies between different tools and languages, but the following steps describe the typical workflow of using the API of the YubiKcom Component:

1. Provide a reference to the component.
   The development tool needs access to the YubiKcom component's Type Library. It contains the interface description. The Type Library is embedded in the component itself; there is no separate .tlb file.
2. Instantiate the component
   The instantiation phase gives a “handle” to the YubiKcom component.
3. Set up and implement a callback/event interface (if required)
   If asynchronous notifications for when a device has been inserted or removed is needed, set up and implement the “sink” interface.

4.1 Methods

Function calls that do not return anything are implemented as methods

```
ykClear
```

4.2 Properties

Configuration data and function calls that have a return value are implemented as properties

```
ykIsInserted
ykIsConfigured
ykEnableNotifications
ykCurPWD
ykNewPWD
ykStaticID
ykUID
ykKey
ykFlagProperty
ykProgram
ykEnableDebug
```

4.3 Callbacks

Asynchronous events are implemented as methods.

```
ykInserted
ykRemoved
```

4.4 String representation

Binary values, such as the 128-bit AES key, are expressed as hexadecimal BSTR strings. A hexadecimal value of 0x1234abcd is therefore be provided as a string like “1234abcd”. Properties with a fixed length longer than a provided string will be padded with trailing zeroes on a nibble-by-nibble basis. For example, the UID field is six bytes (48 bits) in length and providing a string of “123” would yield an UID equal to “120300”

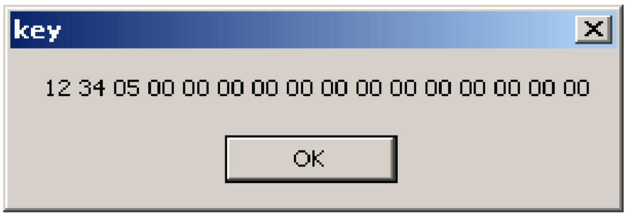
4.5 Yubikey 2.0 support

With the introduction of Yubikey 2.0, two independent configurations are available. In order to maintain binary compatibility with existing third-party code, support for the second configuration is provided by the means of a flag property rather than a separate property or method.

To select the second configuration, simply set the `ykFLAG_SECOND_CONFIG` to true.

4.6 Debug function

Under certain conditions, debugging of function calls can be difficult. Therefore, a debug function is provided which shows a message box for each function call. For example, setting ykKey to 12345 where the ykEnableDebug property is set yields the following message box:



12 34 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00

4.7 A quick example

The syntax and exact methodology varies between different environments, but a sample snippet setting up a YubiKey could look like this

```
obj = new IYubiKeyConfig

obj.ykKey = “12345678”
obj.ykUID = “47115812”
obj.ykStaticID = “123abc”
obj.ykFlagProperty(ykFLAG.ykFLAG_APPEND_CR) = True

If obj.ykProgram Then
      Error
Else
      Success
```

5 API Description

5.1 General

The YubiKcom component exposes functions and properties through the **`IYubiKeyConfig`** interface and callbacks through the **`_IYubiKeyConfigEvents`**, both via the **`YubiKeyConfig`** class.

The component follows the COM Automation model and therefore exposes a dual interface, i.e. a standard interface derived from the **`IDispatch`** interface. This typically makes interoperability in the Win32 environment straight-forward. Further, the .NET environment provides straight-forward interfacing to COM/Automation components.

The component is provided as an In-process server, i.e. it resides in a Dynamic Link Library (DLL), which has an embedded type library. Depending on the “container” environment used, the “wiring” or “wrapping” typically varies. The type library contains enough information for this wiring to be done automatically.

Consult your environment’s section of how to use external COM components.

5.2 Synopsis

Methods and properties:

`ykIsInserted`	Returns TRUE if a Yubikey device is inserted
`ykIsConfigured`	Returns TRUE for a configured device is inserted
`ykEnableNotifications`	Enable asynchronous notifications
`ykCurPWD`	Set current device password
`ykNewPWD`	Set new device password
`ykStaticID`	Set device static identity
`ykUID`	Set device UID
`ykKey`	Set the device AES key
`ykFlagProperty`	Set state of property flags
`ykClear`	Flush all settings
`ykProgram`	Execute programming
`ykEnableDebug`	Enable debug mode

Callbacks:

`ykInserted`	A YubiKey device has been inserted
`ykRemoved`	Device has been removed

Each entry shows the function (method, property etc) in two languages:

1. Visual Basic / VBA style
2. C++ / MIDL style

5.3 IYubiKeyConfig methods and properties

5.3.1 ykIsInserted - Returns TRUE if a Yubikey device is inserted

```
Property ykIsInserted As Long (Read only)
HRESULT ykIsInserted([out, retval] long* pVal);
```

Returns TRUE if one (and only one) YubiKey device is inserted in the USB port of the computer.

See also: `ykIsConfigured, ykEnableNotifications`

5.3.2 ykIsConfigured - Returns TRUE if a configured device is inserted

```
Property ykIsConfigured As Long (Read only)
HRESULT ykIsConfigured([out, retval] long* pVal);
```

Returns TRUE if one (and only one) YubiKey device is inserted in the USB port of the computer and that device is configured.

See also: `ykIsInserted, ykEnableNotifications`

5.3.3 ykEnableNotifications – Enable asynchronous notifications

```
Property ykEnableNotifications As Long (Write only)
HRESULT ykEnableNotifications([in] long rhs);
```

Enables or disables notifications on device insert- or removal events. The scanning is performed in a separate thread and the events are fired asynchronously. The application must implement the appropriate callback routines `ykInserted` and `ykRemoved` in order for this function to be meaningful.

See also: `ykIsInserted, ykInserted, ykRemoved`

5.3.4 ykCurPWD – Set current device password

```
Property ykCurPWD As BSTR (Write only)
HRESULT ykCurPWD([in] BSTR rhs);
```

Set the current password for a password protected device. Failing to correctly set the current password for a device will cause the `ykProgram` operation to fail. The default setting is that no password is used.

See also: `ykNewPWD`

5.3.5 ykNewPWD – Set new device password

```
Property ykNewPWD As BSTR (Write only)
HRESULT ykNewPWD([in] BSTR rhs);
```

Set the new password to be valid after the `ykProgram` operation has been completed. The default action is that no password is used and the device can be reprogrammed by anyone.

See also: `ykCurPWD`

5.3.6 ykStaticID – Set device static identity

```
Property ykStaticID As BSTR (Write only)
HRESULT ykStaticID([in] BSTR rhs);
```

Set the static (public) identity that will be sent in clear text. Setting this device to a blank string (default) means that no static identity is sent at all.

See also: `ykUID`

5.3.7 ykUID – Set device UID

```
Property ykUID As BSTR (Write only)
HRESULT ykUID([in] BSTR rhs);
```

Set the secret (private) identity that will be part of the encrypted OTP. Setting this device to a blank string (default) equals a UID of all zeroes.

See also: `ykStaticID`

5.3.8 ykKey – Set device encryption key

```
Property ykKey As BSTR (Write only)
HRESULT ykKey([in] BSTR rhs);
```

Set the key for encryption of OTPs. Setting this device to a blank string (default) equals a key of all zeroes.

5.3.9 ykFlagProperty – Set state of configuration property flags

```
Property ykFlagProperty(index As ykFLAG) As Long
HRESULT ykFlagProperty([in] ykFLAG index, [in] long rhs);
```

Set state of property flags, which are binary flags for configuration options. Please refer to related documentation for an in-depth description of these flags.

The default state of all flags is False. The flags are defined in the enumeration `ykFLAG` as follows:

`ykFLAG_TAB_FIRST`	Send TAB as first character
`ykFLAG_APPEND_TAB1`	Send TAB after first part
`ykFLAG_APPEND_TAB2`	Send TAB after second part
`ykFLAG_APPEND_DELAY1`	Append a delay after first part is sent
`ykFLAG_APPEND_DELAY2`	Append a delay after second part is sent
`ykFLAG_APPEND_CR`	Append a CR after last character
`ykFLAG_SEND_REF`	Send a reference string first
`ykFLAG_PACING_10MS`	Add 10 ms intra-character pacing
`ykFLAG_PACING_20MS`	Add 20 ms intra-character pacing

Yubikey 1 specific flags

`ykFLAG_TICKET_FIRST`	Send the OTP part first and then the ID part
`ykFLAG_ALLOW_HIDTRIG`	Allow trigger by CAPS and NUM
`ykFLAG_STATIC_TICKET`	The key will generate static tickets only

Yubikey 2 specific flags

`ykFLAG_SECOND_CONFIG`	Select second configuration
`ykFLAG_SHORT_TICKET`	Truncates the OTP output to 16 characters
`ykFLAG_STRONG_PW1`	Mixes upper- and lower case in OTP output
`ykFLAG_STRONG_PW2`	Replaces two characters with numerical digits
`ykFLAG_MAN_UPDATE`	Enables manual update of a static configuration
`ykFLAG_PROTECT_CFG2`	Enables protection of first/second configurations

Yubikey 2.1 specific flags

`ykFLAG_OATH_HOTP6`	Select 6-digit OATH-HOTP output
`ykFLAG_OATH_HOTP8`	Select 8-digit OATH-HOTP output
`ykFLAG_OATH_FIXED_MODHEX`	All public ID bytes Modhex
`ykFLAG_OATH_FIXED_MODHEX1`	First public ID byte Modhex
`ykFLAG_OATH_FIXED_MODHEX2`	First two public ID bytes Modhex

Yubikey 2.2 specific flags

`ykFLAG_OATH_SEED_VAL`	Set the OATH-HOTP initial seed value (numeric value rather than binary flag)
`ykFLAG_CHAL_YUBICO`	Enables OTP challenge-response mode
`ykFLAG_CHAL_HMAC`	Enables HMAC-SHA1 challenge-response
`ykFLAG_HMAC_LT64`	Sets HMAC-SHA1 data to be less than 64 bytes in length. If not set, the data length is fixed at 64 bytes
`ykFLAG_CHAL_KEY_TRIG`	Requires the Yubikey key to be pressed to complete the challenge-response transaction
`ykFLAG_SERIAL_BTN_VISIBLE`	Makes the device serial number visible by pressing the button at startup
`ykFLAG_SERIAL_USB_VISIBLE`	Makes the device serial number visible as a device serial number in the USB descriptor
`ykFLAG_SERIAL_API_VISIBLE`	Makes the device serial number visible as a device serial number in the USB descriptor

5.3.10 ykClear – Flush all settings

```
Sub ykClear
HRESULT ykClear();
```

Restores all properties to default which is not done automatically after `ykProgram` has been executed. No programming is done, only the properties set are cleared.

Configuration 1 is set as default.

5.3.11 ykProgram – Execute device programming

```
Property ykProgram As ykRETCODE
HRESULT ykProgram([out, retval] ykRETCODE* pVal);
```

Performs the actual program operation, where all settings are transferred and written to the non-volatile memory of the YubiKey device.

For Yubikey 2, setting `ykFLAG_SECOND_CONFIG` the to `TRUE` prior to the programming operation writes the setting to the second configuration.

The return codes are defined in the enumeration `ykRETCODE` as follows:

ykOK	The operation completed successfully
ykNO_DEVICE	No device was found
ykMORE_THAN_ONE	More than one device was found
ykREAD_ONLY	The device is read-only (invalid curPWD set)
ykREAD_ERROR	An error occurred when reading the device
ykWRITE_ERROR	An error occurred when writing the device

5.3.12 ykEnableDebug – Enable debug mode

```
Property ykEnableDebug As Long (Write only)
HRESULT ykEnableDebug([in] long rhs);
```

Enables or disables debug mode, where each function call causes a message box to be displayed. This feature is used to trace problems that may occur in environments where no adequate debugging support is at hand. See section 4.5.

5.4 _IYubiKeyEvents callbacks

5.4.1 ykInserted – A device has been inserted

```
Event ykInserted()
HRESULT ykInserted();
```

Called asynchronously when a device is inserted in the USB port. In order to receive this notification, the `ykEnableNotifications` property must be set to True.

See also: `ykEnableNotifications, ykRemoved`

5.4.2 ykRemoved – device has been removed

```
Event ykRemoved()
HRESULT ykRemoved();
```

Called asynchronously when device previously detected as inserted has been removed. In order to receive this notification, the `ykEnableNotifications` property must be set to True.

See also: `ykEnableNotifications, ykInserted`

6 Implementation cookbook

The following section shows how a User Application can be built using COM/ActiveX, and addresses some important issues. Yubico provides a set of example programs, which can be used as a “boilerplate” to develop User Applications.

Please ensure that the YubiKcom component is properly registered before proceeding. See section 4for details.

6.1 Microsoft VB and VBA (Visual Basic for Applications)

Although several differences exist between VB and VBA, the basic methodology to integrate the component is the same.

The typical workflow to create a minimal implementation consists of the following steps: (Assuming Visual Studio .NET 2003)

1. Create a VB project – Windows application. (In VBA applications, enter the Visual Basic editor)
2. Open Project/Add references… and double-click on the **YubiKcom 1.0 Type Library** under the **COM** tab. The component now shows up under **References** section as **YubiKcomLib** and can be conveniently browsed from there.
3. Specify the scope of the COM object variable and declare it:
   ```
   Dim WithEvents obj As
   YubiKcomLib.YubiKeyConfigClass
   ```
4. Determine an appropriate place to instantiate the COM component, for example Form.Load. Instantiate the component by:
   ```
   Set obj = New YubiKcomLib.YubiKeyConfig
   ```
5. To enable asynchronous notifications, add the following line after it:
   ```
   obj.ykEnableNotifications = True
   ```
6. Implement the method obj.ykInserted. Just add a message box to see that we get there:
   ```
   Private Sub obj_ykInserted() Handles
   obj.ykInserted
         MsgBox("Inserted")
   End Sub
   ```
7. Implement the method obj.ykRemoved. Just add a message box to see that we get there:
   ```
   Private Sub obj_ykRemoved() Handles obj.ykRemoved
         MsgBox("Removed")
   End Sub
   ```
8. Run the program. When inserting a device, the “Inserted” message box should appear in a few seconds. When removed the “Removed” message box should appear almost immediately.

Two example implementations for VB and VBA are included in the SAMPLES directory.

Example: VB Client
Environment: Microsoft Visual Basic .NET 2003

Example: Excel Client
Environment: Microsoft Excel 2003

6.2 Microsoft Visual C++

Integration of IPP functionality into a C++ program typically involves more work and deeper understanding of COM/ActiveX than a comparable implementation in VB. Further, depending on the supporting application development framework, the implementation complexity varies. As the `IYubiKeyConfig` interface is derived from `IDispatch`, a “dual interface” is provided, i.e. invocation can be made through direct function calls, or through dispatch invocation.

A further explanation of all aspects of COM programming in VC++ is out of scope of this document, but the typical workflow to create a minimal implementation in an MFC application consists of the following steps.

1. Select a CCmdTarget derived class to instantiate the COM component and to receive callbacks.
2. Import the Type library:
   ```
   #import <yubikcom.dll> no_namespace, named_guids
   ```
3. Declare a pointer to the object and a “cookie” (see 5. below) as a member variables in the class:
   ```
   IYubiKeyConfig *m_obj;
   DWORD m_cookie;
   ```
4. At a suitable point in the class code (OnInitDialog), instantiate the component:
   ```
   HRESULT hr =
   CoCreateInstance(CLSID_YubiKeyConfig, 0, CLSCTX_ALL,
   IID_IYubiKeyConfig, reinterpret_cast<void **>(&m_obj));
   ```
   (Ensure that COM is enabled. If not make a call to `AfxOleInit` or `CoCreateInstance` first).
5. Add appropriate error handling if the instantiation fails:
   ```
   if (FAILED(hr)) {
         . . .
   }
   ```
6. Setup the callback (Connection Point) interface:
   ```
   AfxConnectionAdvise (m_obj, DIID__IYubiKeyConfigEvents,
                        GetIDispatch(FALSE), FALSE,
                        &m_cookie)
   ```
7. Enable event notifications:
   ```
   m_obj->put_ykEnableNotifications(TRUE);
   ```
8. Make the class “COM enabled” by adding macros to the class definition:
   ```
               DECLARE_DISPATCH_MAP()
               DECLARE_INTERFACE_MAP()
   ```
9. Setup the event sink to catch incoming events:
   ```
   BEGIN_DISPATCH_MAP(CYKClientDlg, CDialog)
   //{{ AFX_DISPATCH_MAP(CYKClientDlg)
   //}} AFX_DISPATCH_MAP
   DISP_FUNCTION_ID (CYKClientDlg,
                     "ykInserted", 1,
                     ykInserted, VT_EMPTY,
                     VTS_NONE)
   DISP_FUNCTION_ID (CYKClientDlg,
                     "ykRemoved", 2,
                     ykRemoved, VT_EMPTY,
                     VTS_NONE)
   ```

```
END_DISPATCH_MAP()
BEGIN_INTERFACE_MAP (CYKClientDlg, CDialog)
      INTERFACE_PART (CYKClientDlg,
                        DIID__IYubiKeyConfigEvents,
                        Dispatch)
END_INTERFACE_MAP()
```

10. Declare and implement the member functions for the events:

```
void CYKClientDlg::ykInserted(void)
{
      AfxMessageBox("Inserted");
}
void CYKClientDlg::ykRemoved(void)
{
      AfxMessageBox("Removed");
}
```

11. Add clean-up functions to release the component when the application closes:

```
AfxConnectionUnadvise(m_obj,
            DIID__IYubiKeyConfigEvents,
GetIDispatch(FALSE),
            FALSE, m_cookie);
m_obj->Release();
```

12. Run the program. When inserting a device, the "Inserted" message box should appear in a few seconds. When removed the "Removed" message box should appear almost immediately.

Before `m_obj` goes out of scope, remember to call `AfxConnectionUnadvise`.

Example: MFC Client
Environment: Microsoft Visual C++ .NET 2003

6.3 Microsoft Internet Explorer / HTML scripting

Implementing client-side code as a script in a HTML page opens up exciting new ways to retrieve data remotely through the Web browser.

With a supporting server-side application, data can be captured remotely and transmitted over the Internet.

With the help of Visual Studio .NET, the typical workflow to create a minimal implementation consists of the following steps:

1. Create a new HTML page
2. Instantiate the object within the <BODY> section of the HTML code:

```
<object id="obj"
 classid="CLSID: 0840D787-A0FB-4039-A05E-89B98F99674D"
 viewastext></object>
```

3. Under "Client objects & events", add a script for window onload and enable notifications from there:

```
obj.ykEnableNotifications = True
```

4. Add script handlers for the ykInserted and ykRemoved functions:

```
<SCRIPT>
function obj_ykInserted() {
  alert("Inserted");
}
function obj_ykRemoved() {
  alert("Removed");
}
</SCRIPT>

<SCRIPT LANGUAGE="javascript" FOR="obj"
        EVENT="ykInserted">
 obj_ykInserted()
</SCRIPT>

<SCRIPT LANGUAGE="javascript" FOR="obj"
        EVENT="ykRemoved"> obj_ykRemoved()
</SCRIPT>
```

5. Open the page in Internet Explorer. When inserting a device, the “Inserted” message box should appear in a few seconds. When removed the “Removed” message box should appear almost immediately.

Example: HTML Client
Environment: Internet Explorer + Visual Studio .NET 2003

© 2012 Yubico. All rights reserved.