NISTIR 7929r1

Incorporating Biometric Software Development Kits into the Development Process

Karen Marshall
Ross J. Micheals
Kevin Mangold
Kayee Kwong

http://dx.doi.org/10.6028/NIST.IR.7929r1

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7929r1

Incorporating Biometric Software Development Kits into the Development Process

Karen Marshall
Ross J. Micheals
Kevin Mangold
Kayee Kwong
Information Access Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.7929r1

July 2014

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Acting Under Secretary of Commerce for Standards and Technology and Acting Director

National Institute of Standards and Technology Interagency or Internal Report 7929r1

34 pages

1. Overview

Purpose

The use of biometric devices has become critical to implementing various forms of security in a large number of organizations worldwide. The current state of biometric sensor integration is labor intensive and prone to interoperability issues because of proprietary hardware and software, and a lack of standards in the Software Development Kit (SDK) installation process. Sensor integration incorporates the device hardware installation and intricate patchwork necessary to facilitate full communication between the device's software and the application that will ultimately command and control the target sensor.

This document describes a process used to achieve more flexible and reliable integration of biometric sensor SDKs into the application development process. It is intended for two audiences. First, for *developers*, it provides an analysis of the typical biometric SDK installation process and guidelines on how to increase reliability and repeatability of the SDK installation process, with an emphasis on separating the compile-time and run-time resources. Second, for biometric sensor *providers*,[1] it suggests a specific set of resources to be distributed in SDKs that allow developers to access SDK components as well as obtain details pertinent to the use and maintenance of their biometric sensor solutions. Both audiences are encouraged to use the document as informative guidance to help develop and improve their own processes; readers are encouraged to adapt, modify, or evolve content to suit their own context of use.

Introduction/Background

As development efforts increasingly include the use of biometric devices, device-SDK-integration needs to be better assimilated into the development process with practices that include the use of proven industry tools. The increasing use of biometric devices in the enterprise is accompanied by large-scale development efforts. One way industry manages and promotes increased production is by facilitating automation. Build servers are one of the standard tools used to implement automation (and maintain code integrity) in the software development arena. The benefits of using build servers are well established; however, most biometric SDK installation packages in the current marketplace (late 2012) do not easily lend themselves for use within a build server.

A build server inherently *requires* the ability to compile software on one machine and execute the build on other machines. Build servers and execution hosts do not necessarily share a common operating environment. Cross-compilation facilitates the ability to install, uninstall, and use SDK compile-time and run-time components *independent* of each other. The biometric device SDKs used in developing this document did not explicitly provide this capability "out of the box."

The installation process of biometric SDKs is often unpredictable and proprietary. Therefore, most of the improvements discussed in this document focus on restructuring the installation phase of SDK integration. A provider's installer, which we refer to as an *out-of-box* installer, is often provided to execute the SDK installation. It is typically "wizard-style" software initiated by a developer on the machine that will host the biometric device, host the development environment, *and* host the compiled code that integrates the sensor.

[1] We use the term *provider* to refer to the organization or party responsible for the development, ownership, maintenance, and support of a biometric SDK. Biometric sensor vendors, manufacturers, companies, or integrators are examples of potential *providers*.

Automated installers (e.g. out-of-box installers) are not used by all sensor providers. When an installer is absent, installation may be carried out in a variety of ways. It may require manual placement of the dependencies into appropriate application and operating system directories. In other cases, the host operating systems may have device support built in, such as with "class level" drivers and libraries. There may even be several wizards that must be orchestrated in a specific order to accomplish a single installation. The important point is that *SDKs vary widely* both in the tools provided and in the *effectiveness and completeness* of accompanying documentation. This document represents the first step towards guiding providers toward a uniform set of SDK resources.

When an out-of-box installer is distributed in SDK packages, it may not allow easy access to its internal resources. In addition, out-of-box installers typically install (or uninstall) dependencies with one broad stroke. An installation engine itself may further cloak dependencies hidden from the developer. In these cases, it becomes extremely challenging to automate the determination of whether or not a system is in a state that satisfies the SDK prerequisites. In other words, when using out-of-box installers, there is no clear identification or access to these dependencies.

Throughout this document we classify dependencies into the following categories:

1. *Compile-time* dependencies are anything needed to affect/support *compilation* of a software application (i.e., development tools, SDK libraries). Development tools can include a full integrated development tool (IDE) or command line tools.
2. *Run-time* dependencies are anything needed to affect/support *execution* of principal software. These may include, but are certainly not limited to, device drivers, configuration files, or static and dynamic software libraries such as dynamic link libraries (DLLs).
3. *Secondary* dependencies are any other dependencies that are unique to the provider's SDK, but are neither run-time nor compile-time. For example, some SDK installers will not run without libraries specific to the installation engine.

Frequently, a single requirement can manifest itself within more than one dependency category. The most common type of "dual" membership would be a software library used during both compile-time and run-time. A run-time file dependency might become a secondary dependency if the installation script checks for the presence of the same file. Nevertheless, for the sake of cross-compilation, it is useful to distinguish between run-time and compile-time dependencies.

This document identifies two common secondary dependencies that severely inconvenienced the development process.

1. **Some installers would run only within certain operating systems.** That is, the installer (not the SDK itself) has a built-in mechanism that checks the operating system on which the installation executable is running. If the operating system is not one considered "supported" by the SDK provider, the installer will halt the installation. This halting of the installation process implies that the provider has not tested and will not guarantee the successful operation of its SDK in that environment. If the installer is halted, the use of a sensor in an unsupported environment could be precluded.
2. **The coupling between resources and installation engines is tight.** Provider installer packages typically contain scripts or executables that orchestrate the installation process, but do not directly satisfy any of the core compile- or run-time dependencies. In many cases, attempting to use SDK components independent of the installation architecture requires considerable workaround. For example, the purpose of a Microsoft Installer (MSI) file is to independently install SDK components. However, if an MSI file is configured to launch only when triggered in a particular manner (such as by a setup.exe file), reconfiguring the MSI file for independent use is necessary.

Secondary dependencies are particularly frustrating because they can add a level of artificial complexity to what may already be an obtuse process.

As an alternative to out-of-box installers, (or the absence of any installers), this document will discuss the use of custom installers. Installers created by developers that perform automated installation of compile-time and run-time dependencies independently. These installers do not have the complex installer architecture or the restrictions found in many out-of-box installers.

2. About this Document

The objective of this document is to assist developers and providers in developing a process that enhances and brings consistency to a provider's SDK. This process can be used to

- validate a repeatable and automatable cross-compilation process
- clearly define SDK functionality and components
- identify and isolate the causes of baseline integration errors.

The resulting automated process reduces development time and improves the quality and reliability of the installation process.

The remainder of the document is organized as follows. Section 3 provides (as a result of this work) recommendations for sensor providers that highlight a uniform set of resources that, if provided to developers, will lessen the challenges inherent in the existing integration process and promote increased industry acceptance. Section 0 defines and discusses *custom installers*—automated (in as much as possible) scripts that install and uninstall the SDK resources. Section 5 describes a series of tests that may be used to vet those installers. Finally, Section 6 provides an overview of a set of test *templates* that may be used as a framework to generate the specific tests in Section 5. A glossary is included in Section 9 and an appendix of test templates in Section 10.

3. Recommendations

Biometric SDK installation materials vary widely from provider to provider. These variations and the absence of an automated integration methodology often propagate further challenges into the application development process. The process described in this document seeks to decrease the uncertainty and the development time involved with biometric SDK installation and integration.

Therefore, to respond to challenges discovered by the authors and to provide developers with the flexibility, clarity, and access needed for a more reliable installation process, the authors recommend that biometric SDK providers consistently adopt the following recommendations. Many of these may seem obvious. However, each of these recommendations follows directly from author experiences.

Recommendation 1: Provide a list of device models for which the SDK can be used

A list of devices/device models for which the SDK can be used and the germane information pertaining to their use is helpful in the documentation. SDKs are usually requested and provided with the initial acquisition of a sensor. They often support more than one model of a device. Provided there is no licensing restriction, an SDK may support multiple devices on one system using the same model or possibly different models. Sometimes an SDK does not support more than one device (of the same model or distinct models) on the same machine. Having this information clearly stated saves time and assists with development planning.

Recommendation 2: Provide specifics on the environment on which the SDK has been tested

SDK documentation sometimes includes a list of supported operating systems on which the provider has successfully tested their software. This serves as the provider's assurance that the SDK will work in these environments. In research and development, a wider array of operating environments and architectures may be used than is routinely implemented; for example, consider an integration into a beta or recently released operating system. When out-of-box installers are programmed to abort installation in an environment whose operating system is not on the provider's list, this explicitly precludes the opportunity to use the SDK in an environment that might successfully host the SDK but is not "officially" supported. Therefore, providers are encouraged to not exclude unsupported but functional environments. This is ultimately to the benefit of both the integrator and the provider, since integrators can experiment with their development in novel environments, and providers have an opportunity for feedback on configurations that they may not have previously considered.

Recommendation 3: Include demonstration application executable and source code

Demonstration applications are an invaluable tool in verifying the operation of a sensor in an environment. At a minimum, an executable should always be included in the SDK to show that full communication exists between the operating system and the device and to exhibit at a minimum the basic functionality of the device when installed correctly. Including source code provides the developer a first look at how the sensor API is integrated with application code. This insight could be applied to other programming code and aid in the integration of a sensor into custom application code. Having both a working executable and working source code provides the best case scenario.

Source code provided without a precompiled working executable can present time-consuming problems, the most important of which is the extra time required to get the code to run. In some instances, code may have indecipherable comments, or code may be nothing more than a remnant of what was originally used by the provider. In either case this is discovered only by trial and error. It is critical that the code work to realize the benefits of having a demonstration application. A disadvantage of having code and no working executable is that there must be a compilation tool installed on the machine hosting the sensor for the demonstration application to work.

Recommendation 4: Provide an option to install compile-time and run-time dependencies separately

Currently, an out-of-box installer typically provides the option only for either installing or uninstalling all components at once on a single machine. However, the need to compile on one machine and execute on another is often required. Because of this limitation in biometric sensor installers as currently supplied by providers, the ability to compile and execute on separate machines as separate and distinct functions is not provided. This could be an opportunity for providers to satisfy a potentially increasing need.

As the number of businesses using biometric sensors increases, the process for installing and integrating biometric sensors can be expedited if providers offer the ability to compile applications separate from their run-time environments, unlike the installers typically provided. This process can be accomplished as described in this paper and would allow for the use of a build server to more efficiently manage the code. Alternatively, cross-compilation would be possible if providers supplied installers with the option to install or uninstall all general dependencies, compile-time dependencies, and run-time dependencies as independent actions.

In the absence of that option, there is a less elegant way of providing access to dependencies separately. Such a provision can be done by clearly and consistently listing in the documentation of all SDKs those dependencies needed for compilation and those needed for execution and by making them available unrestricted by the installer architecture. Implementation time would be greatly reduced absent restrictions or the workarounds required for extraneous requirements and the separation of dependencies required for compilation and execution.

Recommendation 5: State instructions for tethering the device

SDK documentation should explicitly state how the physical and logical connection for the device must be handled. The documentation should state when and how the device should be connected to a system. Some devices have a specific order in which tasks must be performed. If the order is not followed, the device will not be recognized by the operating system or initialize. The other discrepancy that can cause an initialization anomaly is not configuring the correct settings on the device (e.g., camera) prior to connection. For example, a digital camera may require a set of specific camera settings before it is connected to a host. If the correct order is not observed, the camera will not initialize. Explicitly identifying this in the documentation would avoid excessive time spent on something that should be a simple task.

Recommendation 6: Provide instructions for SDK updates

SDK updates can be helpful and important to a sensor or system. They can enhance the functionality of a sensor or include support for later models of a sensor. Documentation should provide instructions that include a vehicle by which providers notify and instruct a user of all pertinent information concerning receiving and implementing SDK updates. Most are received via the Internet, but some must still be received by postal mail. Among many providers, there has been no consistent method for receiving notification when an update is available. The SDK consumer has the responsibility to randomly check from time to time with a provider for updates. In addition, when updates are received, they often have no instructions on how the updates should be deployed. If the sensor is part of a system, performing the update could produce unknown effects or disable it completely. This discourages updating sensors.

Recommendation 7: If necessary, note elevated privileges to install/run

Elevated privileges should not be necessary to install/run components of an SDK. However, if they are, then that should be noted. Enterprise systems permissions are often structured in a proprietary design and access can vary widely among departments. The mission should not be to discover experimentally the minimum necessary requirements for using the SDKs but rather to install and use them quickly and effectively. Providing developers with explicit permission information avoids the trial-and-error time spent resolving permission restrictions during SDK implementation.

Recommendation 8: Provide a way to identify SDK components that are dependent on target architecture

As discussed throughout this document, there is wide variety in how SDKs store components—within an automated installer, as a store of files, a series of "wizards," or other combinations thereof. Often these components have a direct dependency on the host architecture—libraries and executables being the most typical. If these dependencies are either not satisfied or are confused (that is, the files are not matched with the correct architecture) the project and the device communication will likely fail. Therefore, a precise way to identify and access the right resource for a specific purpose can expedite the development process.

The following real-world example motivated this particular recommendation. A team member was performing development in a 64-bit environment, but targeting the final application for 32-bit. The project compiled on the host environment, but would not run correctly on the desired destination. Additionally, the error received had little to do with the actual problem. During troubleshooting, the developer verified the presence of the correct files; however, since the 64-bit and 32-bit versions had the same filename, the architectural dependency was still hidden from the developer. If the name of the library (or executable) includes an architecture mnemonic, then the dependency can be consistently and robustly identified regardless of its location.

4. Custom Installers

A *custom installer* is defined as software that automates the installation of the provider's SDK. Ideally, such installers

- provide consistent, successful, installations
- can satisfy compile-time and run-time dependencies *independently*
- require no more user interaction than is absolutely necessary
- are initiated by a single action, such as single command or mouse action
- behave uniformly
- are tested across multiple environments
- can be used to easily upgrade previously installed versions of the SDK.

Custom installers can produce a reliable and debugged configuration upon which developers can build because they are easy-to-use tools that quickly and successfully accomplish bug-free installations of specific SDK components. Typically, a custom installer is a combination of scripts or batch files. They may orchestrate a provider's installer, implement documented requirements, or may even be independent recreations of the provider's installer. That is, if an installer is not provided, then a custom installer must be constructed once the compile-time and run-time requirements are identified and understood.

Custom installers can require significant work to create. Upon first acquisition from a provider, the set of SDK resources provided are unfamiliar. If an out-of-box installer is provided, it needs to be disassembled to gain access to its resources. The SDK dependencies are then reconstructed into custom installers. Secondary dependencies must be isolated and overcome; for example, as mentioned previously, an installer might have a restriction that would allow it to be launched only when called in a specific manner.

The testing process described in the remainder of this document presumes that a development team either is using (or wants to use) some sort of custom installers—even if they are simple wrappers around the provider's out-of-box installer. Although the end goal is an installer with the above characteristics, the tests that follow also provide a framework within which integrators can iteratively develop, debug, and isolate problems and verify the custom installers. In other words, it is expected tests and installers will be developed iteratively and simultaneously, even though the document's structure may suggest otherwise.

The information in the next section presumes that the custom installers will comprise a device driver installer (if applicable) and an SDK installer that can install the run-time and compile-time components independently. The precise form of the installers is not relevant because different biometric SDKs may require significantly different approaches to automated installation. However, in the authors' experience, distinguishing among drivers, the run-time components of the SDK, and the compile-time components of the SDK is a widely applicable, and therefore useful, categorization.

5. Custom Installer Tests

Custom installer tests are designed to verify that custom installers provide the equivalent functionality to that resulting from the installation of the provider's SDK. The basic strategy is to verify the functionality as provided and to test the recreated custom installers piecemeal. Compile-time and run-time dependencies can more easily be "teased" apart through the isolation gained through repeatable tests.

Tests must be performed sequentially as listed to get meaningful results. They assist with the incorporation of SDKs into the development process by verifying specific functionality in an incremental way as well as by vetting any SDK-related bugs. This debugging occurs separately from the larger software development process so as to localize issues relating directly to the SDK installation and sensor operation.

The SDK installation and operation discussed here is sensor integration up to, but not including, calling source code. At the conclusion of testing, there is increased user confidence in the successful and expected outcome from using the custom installers. Performing these tests also helps to determine whether an existing SDK can or cannot work in an updated (or newly released) environment in the absence of provider documentation.

The six sensor integration tests, the custom installers they exercise, and page numbers for their test templates are as follows:

Test Name	Installers Exercised	Template
Out-of-Box Installer Test	(no custom installers, used for baseline functionality)	see page 24
Custom Driver Installer Test	Custom driver installer	see page 25
Demonstration Application Executable Test	Custom driver installer and run-time components of the custom SDK installer	see page 26
Custom Application Compile and Run Test	Custom driver installer, run-time components, and compile-time components of the custom SDK installer	see page 28
Custom Application Compile-Only Test	Compile-time components of the custom SDK installer	see page 29
Cross-Compilation Test	Custom driver installer, run-time components, and compile-time components of the custom SDK installer independent of each other	see page 31

Unfortunately, the elements of an SDK's functionality may not always be evident or documented by the provider. Structuring the custom installers in a regular fashion can help to fill this gap as well as provide a universal and uniform method applicable to multiple biometric modalities. The tests will be described in more detail in Section VI with examples provided in the appendix.

Out-of-Box Installer Test

The purpose of the **Out-of-Box Installer Test** is to assess and document the baseline functionality of the provider's SDK within the target environment(s). If installation software is provided, it is usually in some form of a script, batch file, or "wizard." There are cases, however, where documentation is the only form of installation instruction. Regardless, this test generally consists of attempting SDK installation using the provided installer or information only (i.e., it does not use any custom installers).

To pass this test, the operating system must recognize the sensor, and the demonstration application (if provided) must perform its intended functions. A test result of "Pass" is an early indication that the sensor can be integrated into this environment and later into a larger custom software project in a similar environment. As such, this level of functionality becomes the baseline for what is achievable with this SDK in the target environment. The custom installers, described in the sections that follow, are expected to meet or exceed this baseline performance. Therefore, passing this test establishes that it is possible to satisfy all necessary SDK dependencies and produce full sensor operation.

If the out-of-box installer terminates installation, the test is determined to have failed. The installation software may provide a notification that the target environment is not supported by the provider, but in the authors' experience, this cannot be universally expected.

If evidence of a functioning sensor is not present or if the provided installer terminates prematurely, the test is determined to have failed. The installation software may provide a notification that the target environment is not supported by the provider.

If the SDK does not contain an out-of-box installer (or clear and successful manual installation instructions), this test is skipped. Not having an out-of-box installer or successful install instructions means

- the developer has no immediate evidence that the sensor will work under the target conditions
- the developer has no opportunity to observe the full set of provider prescribed sensor functionality (independent of whether the developer has implemented it fully and correctly).

Custom Driver Installer Test

The purpose of the **Custom Driver Installer Test** is to verify that the custom installer can successfully install device drivers (i.e., the fundamental run-time component). Having the operating system recognize the biometric sensor is the first and most basic step towards successful integration.

The criterion to pass this test is that the host operating system correctly recognizes the attached sensor. The sensor may be directly attached to the host, or it may be attached in some other way (e.g., via a network). A test result of "Pass" establishes that the run-time components are sufficient for the operating system to recognize the sensor and that the custom driver installer is performing correctly. Conversely, a "Failed" test highlights a specific type of error. If an out-of-box test has previously passed, the error(s) that surface here will be due to an ineffective driver/run-time installation.

If there is no out-of-box installer and, thus, no out-of-box installer test, the errors found do not have to be limited to run-time related errors. There could be another reason the sensor cannot work in this environment. Without a successful out-of-box installer test and, therefore, no associated baseline performance, no actual evidence exists to indicate that the sensor can achieve functionality under these conditions.

Demonstration Application Executable Test

The purpose of the **Demonstration Application Executable Test** is to execute the custom driver installer and SDK installer (on the same machine) and then measure whether they have performed correctly and completely. The demonstration application distributed in the SDK is used to perform this measurement. The executable could come from the provider as binary, as source code or both. If source code only is provided its resulting executable can be used for this test, provided it compiles successfully. The result of this test is a "Pass" if two things occur: (1) the host operating system must correctly recognize the attached sensor and (2) the demonstration application supplied by the sensor provider must be operational. "Pass" establishes that custom driver installer and SDK installer used in tandem provide the equivalent level of functionality to that provided by the out-of-box installer and that non-driver run-time elements have been installed correctly.

Run-time dependencies can include drivers and non-driver components. The previous custom driver installer test isolated and verified the correct installation of the drivers. The successful running of the demonstration application executable indicates that in addition to the drivers, the rest of the run-time elements (the non-driver run-time elements) are present and performing correctly. Errors that surface here could be related to a non-driver run-time dependency.

If there is no provider-supplied demonstration application, this test is skipped. If this test is not performed, the opportunity to isolate an error related to a non-driver run-time component is lost.

Custom Application Compile and Run Test

The purpose of the **Custom Application Compile and Run Test** is to measure whether using the custom installer(s), a sensor can be fully integrated with a custom application. During this test, both the custom driver installer and custom SDK installer are executed (on the same machine), similar to the previous test. This test further requires the incorporation of the SDK compile-time dependencies into the custom application source code to compile *and run* the custom application. If successful, the test then attempts to verify whether the sensor performs the functions prescribed via the custom application. This test is determined "Pass" if the application compiles and executes without errors and performs all its apparent functions.

A "Pass" result on this test indicates the sensor can be integrated with a custom application and that this integration can be done using the custom installers.

If a demonstration application was provided (and passed), then a "Fail" result here would be attributed to integrating compile-time components into the custom application source code. The demonstration application executable test established that the installers provided all the dependencies necessary for full operation of the sensor, leaving only the compile-time dependencies as potential sources for failure.

If a demonstration application was not included with the SDK, the opportunity to isolate errors related to the installers' effectiveness is lost. Without testing the demonstration application executable, custom installer errors would manifest here. Therefore, errors that surface here could be related to compile-time dependencies, failure of the custom installers, or both.

Custom Application Compile-Only Test

The purpose of the **Custom Application Compile-Only Test** is to demonstrate that the custom installers will allow a custom application to be compiled successfully. This begins with compile-time dependency installation. The test deliberately omits run-time dependency installation and execution (included in the

preceding Custom Application Compile and Run Test) to confirm whether the custom installer responsible for compile-time dependency installation is complete when used independent of custom runtime installers. The test then incorporates SDK dependencies into a custom application and attempts compilation.

This test has a result of "Pass" if the application compiles without fatal errors. However, even if the application compiles without fatal errors, the application still may not run correctly. A result of "Pass" on this test indicates only that a custom application and an SDK can compile successfully without drivers or run-time components installed. Confirming this capability supports the possibility of using a build server to manage this software. Because the success of this process and the product it yields is vital to the test described in the next section (and cross-compilation in general), this test provides the mechanism to identify and resolve compilation errors before the next test is performed.

Errors that surface here will likely indicate that the compile-time components installed are not sufficient without run-time resources. Integration issues should have been isolated and resolved during the previous test.

Cross-Compilation Test

The purpose of this test is to verify that using the custom installers, a custom application can compile on one machine and execute on another. The test has two segments:

1. In the first segment, a custom installer is used to install SDK compile-time dependencies on a machine (compilation host). No SDK run-time components are present. With the appropriate SDK resources integrated and with the use of a compilation tool, the custom application is compiled creating an executable file. The process carried out during this segment is the same as the Custom Application Compile-Only Test. It is a necessary component of this test as the executable file created is required to perform the next segment of this test.

2. The second segment is performed on a different machine (execution host) using the custom run-time installer and the custom application executable created in the first segment. The SDK compilation components are not installed for this segment. The custom application is opened, and all its intended functions are exercised. If the executable successfully runs on the execution host, this test is determined a "Pass," indicating that when the custom run-time installers are used, the custom application compiled on one system can successfully run on another system without the installation of compilation tools. "Pass" also confirms that a sensor server is an option using custom installers to manage SDK dependencies. Errors that surface here indicate that SDK run-time dependencies (in the absence of compilation components) are insufficient for cross-compilation.

Pretest Environment

Proper preparation of the pretest environment is necessary to draw accurate conclusions, track variations in results, and debug errors. Tests must be performed in a controlled environment. There should be no software or underlying files other than those specifically loaded by the test author. Test results heavily rely on the presence or absence of specific dependencies, so it is imperative that there are no remnants of software present unless left in place intentionally. To eliminate, or at least minimize variability, the system environment should be reset at the initialization of all tests.

Before the installers are executed, the environment must be well defined. Hardware and software used must be documented. For accurate analysis, test results must reflect solely the effects of the custom installers. Results must show whether installers work as intended and provide the dependencies needed.

The test environment should contain the least amount of software that would be present on a system used for developing custom applications that would incorporate the SDKs being installed. Typically, the test environment will consist solely of a "fresh" install of an operating system. If compilation tasks are performed in some of the tests, a compilation or development environment of choice must be included. These elements simulate an off-the-shelf computing environment that provides an equivalent starting point from which to measure results. This configuration should not have had any other operations performed (e.g., installs, uninstalls) before saving it for reuse, particularly since that can leave behind files that may be shared by other applications. Tests that perform compilation require a "development configuration" environment that includes the operating system, its updates, and a compilation tool. Tests that do not perform compilation require a "baseline configuration" environment consisting of only an operating system and its updates. Each environment can be saved as an image and reapplied, as needed, to a machine. Applying one of these images to a machine provides a consistent and known configuration needed to begin all tests in an equivalent controlled environment. A mechanism should be chosen to manage these images.

As each sensor integration test is performed, the result is measured by the capability added via an installer. Capabilities are added incrementally and in sequence to accomplish sensor installation and cross-compilation. For the outcomes to be attributed exclusively to the installers, all other factors must be equal. This preparation is intended to achieve the maximum credibility and accuracy possible for the testing process and its results.

Hardware should be the same for all tests in a series to eliminate variations in test outcomes. Hardware configurations should be recorded and stored for reference. Anomalies could require an investigation into possible issues related to hardware; repeating the same test on a different hardware configuration is a means to isolate and identify problems that arise from differences or incompatibilities among hardware.

For the work referenced in this document, the preparation of the pretest environment included virtual and physical machines, each with a unique identifier. Hardware configurations were the same among all the machines. Operating systems were run in 32-bit mode. For efficiency and repeatability physical machines were instrumented with an application that creates, stores, and applies computer configuration images or snapshots. Baseline and development configurations were prepared in advance and saved. Microsoft Visual Studio 2010 SP1 was selected as the compilation tool in the development configuration. The appropriate snapshot was applied before each test.

Every snapshot included uniform environment attributes necessary for testing, such as settings that make a machine part of an internal Active Directory domain to allow access to network file systems as well as devices connected via the network. Settings captured also included a uniform domain user account under which all testing was performed. The account was established with local administrator privileges to allow maximum access to resources and mitigate any unforeseeable SDK permission restrictions. This was done mostly for convenience; repeating tests with different sets of permissions would be a means to isolate and identify problems unique to security or permission settings.

A baseline configuration, which contains no compilation tool, was applied to set the environment for the Out-of-Box Installer Test, the Custom Driver Installer Test, the Demonstration Application Executable Test, and the execution segment of the Cross-Compilation Test. The development configuration, which contains a compilation tool, was applied for the Custom Application Compile and Run Test, the Custom Application Compile-Only Test, and the compilation segment of the Cross-Compilation Test.

6. Test Templates

Biometric sensors (1) come in many types; (2) can be categorized by many modalities, such as fingerprint, iris, face, palm, and hand scanners; (3) can be connected to a network or plugged directly into a host computer; and (4) can be embedded into computers, laptops, and various other digital devices. These variations present a challenge when composing test procedures that can be applied universally to biometric sensors.

This section provides templates that demonstrate how to write sensor integration tests. A template is a universal outline of the topics and considerations necessary to construct a test. Templates contain both text to be used verbatim and directions to a test author that explain how to formulate test content unique to the target SDK. Directions given are intended to render a test that is specialized in its goal and to the sensor used.

6.A Common to All Templates

In this section, we describe the aspects of the template sections that are common to all of the templates. Templates for all sensor integration tests are included in appendix A.

Each template is organized into five sections:

1. Purpose
2. Installer(s) Tested
3. Prerequisites
4. Pass/Fail Criterion
5. Instructions

Each section has a title (section header) and its associated text or directions (section content). If the section is essentially the same for all of the tests, it is shown in line for easy reference. As mentioned in the introduction, test authors are expected to tailor the tests for their own particular context

6.A.1 Purpose

The Purpose section provides a summary of the goals of the test. It is the only section that remains unchanged from the template to the text in a test.

6.A.2 Installer(s) Tested

The Installer(s) Tested section directs the test author to identify the installer(s) being tested and the way to properly invoke them. It typically takes the following form:

Installer(s) Tested

[The test author shall specify the path and filename of installer(s) along with relevant command line arguments and other relevant information, such as security requirements.]

When writing the test, the test author may use this direction for the test content or may decide to insert the specific name information in preparation for a specific test. Depending on the test performed, this section can refer to one or multiple installers.

Specific manufacturer, model, and version information can be used to help organize the logical location of the various installers within the file system. The filenames of the custom installers also reflected this same information for clear identification. Descriptive paths and filenames provide quick access to

installers, facilitate tracking, and help reduce the ambiguity that arises when managing collections of biometric SDKs.

6.A.3 Prerequisites

The Prerequisites section lists the requirements that must be met before the test can be meaningfully executed. They take the approximate form:

Prerequisites

The tester shall verify all of the following:

1. a predetermined configuration has been applied to the host
2. the biometric device *[is or is not]* connected to the host *[or its network]*
3. a *[prerequisite test]* has been successful.

The Prerequisites template content contains directions that describe the considerations for preparing the pretest environment.

The first of the considerations concerns the initial state of the host. Prior to every test, the host machine must be set to the appropriate known configuration. The appropriate configuration is determined by the needs of the test. If the test includes compilation, the host should be imaged with the development configuration. If no compilation is necessary, the baseline configuration image should be used. Using predetermined configurations ensures there are no unknown elements (such as files orphaned by uninstallers) that could positively or negatively influence test results. The evidence of applying the necessary state should be noted.

Second, there is a direction to the test author to consider whether the sensor involved has particular connection or configuration considerations that need to be addressed before the test begins. The test author must state what the connection status should be between the sensor and the operating system (or between the sensor and the physical host) for an embedded, networked, or USB sensor. Sensors can vary significantly in how and when they should be connected to a system. If the sensor has configuration steps to be executed before the test starts and they are not adhered to, the sensor will not be recognized by the system. The following examples illustrate some configuration steps.

- A networked device may need to be physically opened and assigned an IP address before it is connected to the network for the first time. This information should be included here.
- Some sensors must be connected in a specific order (such as after an installer is run) during the test. This should be noted in the instructions section. Still, other sensors may have the capability to be successfully connected at any point during the installation process, in which case it should be included in the section that corresponds to the time at which it is performed.
- If a dongle is provided with the biometric device by the manufacturer, it must be plugged into the system for the driver installer to execute. If the dongle is not present, the sensor will not be recognized. If an Out-of-Box Installer Test was not available, this would be the first test where that requirement is evident, and this knowledge is critical to the success of this test and subsequent tests.

Finally, prerequisite tests are listed. These are previous tests that, if failed, could render the current test meaningless.

6.A.4 Pass/Fail Criterion

The Pass/Fail Criterion section details what specific behavior must be observed for the test to pass. The criteria will vary widely depending on the SDK used in a test, the context of its use, and the ultimate level of functionality required.

The template instructs the tester to judge whether the test has passed or failed. In templates where criteria statements can be further tailored to the test or the sensor, the test author should include instructions that detail the criteria needed to determine a "Pass" or "Fail." If passing the test is defined by a requirement that the custom application compiles, evidence of this can be readily apparent. However, if passing a test is defined by the operating system recognizing the sensor, evidence of this recognition may not be displayed the same way for every sensor. Therefore, the test author should include what evidence must be observed to conclude a pass result.

When writing pass/fail criteria, test authors are encouraged to limit the criteria to the "go/no-go" decisions only, and leave *process* to the Instructions section. For example, one criterion for a fingerprint scanner might be that "the demonstration application successfully shows a fingerprint image." By limiting the scope of the criteria, section reuse is facilitated since *how* a tester performs the test may change independent of the criteria.

6.A.5 Instructions

The Instructions section details how the tester should execute the test. They take the approximate form:

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.]

[For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when it must be done.]

The Instructions section instructs the test author to provide specific, step-by-step instructions that allows the tester to evaluate results and compare them against the pass/fail criteria. These instructions will be specific to the sensor being used, and, therefore, the resulting test content must be crafted by the test author. The test content requires specific knowledge of SDK components and the way the sensor operates. The test content also requires that the test author understands the proper operations of the sensor and SDK being test.

In general, this section usually includes running installers, getting a sensor to work, and detailing any other steps necessary to accomplish the purpose of the test. The order of the steps may also be important. Some steps may seem obvious, but in the work that motivated this document, the authors were surprised on several occasions that tests would pass only when seemingly unrelated steps were executed in a particular order. For example, for some sensors, the point at which the device is connected is neither arbitrary nor flexible. In other cases, some software requirement(s) must be met immediately before or after connecting the sensor. Consequently, if the correct order is not followed, the device will not work.

6.B Template Detail

The templates in their entirety are included in the appendix at the end of the document. In this section, we detail those aspects of templates that are unique or significant related to composing a particular test and,

therefore, warrant more detailed discussion. This information is intended to be used primarily by test authors in addition to (a) the general contents of the previous section and (b) the templates themselves.

For each test, the section to note in a callout box is shown and additional information is provided below it.

Unattended installer developers, test authors, and testers should consider these special notes when developing installers, authoring tests, or executing them.

6.B.1 Out-of-Box Installer Test

This test is fully described by both the previous section and the template.

6.B.2 Custom Driver Installer Test

The following are special considerations for a Custom Driver Installer Test.

Purpose

To determine if the custom installer correctly installs (installers correctly install) the biometric sensor device driver *[or, if device drivers are built-in to the baseline configuration to determine if the baseline configuration can correctly recognize the biometric sensor]*

The Custom Driver Installer Test establishes the first and most basic step towards integrating the sensor into an environment that separates run-time and compile-time dependencies.

Prerequisites

The tester shall verify all of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. *[If applicable]* an Out-of-Box Installer Test has been successful.

An Out-of-Box Installer Test is helpful but not mandatory. This test provides value because, if passed, it proves that it is possible to get the sensor to its fully functional level within this system. Custom installers can be created (in the absence of an out-of-box installer) to perform an installation successfully. Therefore, if an Out-of-Box Installer Test was not performed because of the absence of an out-of-box installer, this test should still be performed because it can be successful.

6.B.3 Demonstration Application Executable Test

The following are special considerations for a Demonstration Application Executable Test.

Purpose

To determine, via a provided demonstration application, if the custom installer correctly installs (installers correctly install) both the biometric sensor device drivers *[if applicable]* and the SDK

Demonstration applications are not always available. If one exists, it is provided by the provider via the SDK in either source code or binary form. Ultimately, the decision to provide one is made solely by each sensor provider. Many SDKs come without one, and a demonstration application executable must be available to perform this test.

Prerequisites

The tester shall verify all of the following:

1. a demonstration application executable is available
2. a predetermined baseline configuration has been applied to the host
3. the biometric device is *[or is not]* connected to *[the host or its network]*
4. a Custom Driver Installer Test has been successful.

The Custom Driver Installer Test is a mandatory prerequisite to the Demonstration Application Executable Test. If that prerequisite test has not been passed, this test should not be performed. Performing this test without the success of the prerequisite test could expand the number of possible sources and increase the number of errors, thus potentially making the debugging process more time-consuming. If the Custom Driver Installer Test passes, then the success of the driver installer is a known and confirmed result. Therefore, this test isolates the unknown outcome that comes from using the SDK installer together with the driver installer. Errors that resulted from the driver installation alone in the present environment would have been resolved before this test was executed.

This section also specifies that a baseline configuration has been applied to set the host to a known configuration. A development tool should *not* be necessary to perform this test.

Pass/Fail Criterion

This test is a "Pass" if both of the following occur:

1. The host operating system correctly recognizes the sensor. *[The test author shall state specifically how to determine if the sensor is "correctly recognized."]*
2. The demonstration application executable operates correctly. *[The test author shall state specifically what evidence needs to be observed to deem the demonstration application executable operational.]*

[The "Passed" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration. Some of these steps may have been established in a previous test; they may be repeated here.]

Otherwise, the test is a "Fail."

The cumulative result of this test should be equivalent to that of the Out-of-Box Installer Test. The demonstration application executable is the tool used to exercise and evaluate that result, which includes the communication between the system and device as well as the operation of the device. These steps could differ depending on the sensor used. For example, this test would demonstrate the capabilities of an iris camera that often gives voice commands to move forward or backward to capture an image. This test will also show whether it is firing the correct commands to accomplish the capture. The test author has the opportunity to state which and how many of these capabilities should be observed to judge the sensor fully functional and determine the test as passed.

6.B.4 Custom Application Compile and Run Test

The following are special considerations for a Custom Application Compile and Run Test.

Purpose

To determine if the custom installers allow a custom application source code to be compiled and run successfully

Because biometric sensors are not intended to be used independently, a biometric sensor is almost always integrated with an application. This test provides evidence that a sensor can be fully and successfully integrated with a custom application using custom installers to perform the installation.

Prerequisites

The tester shall verify all of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. a Demonstration Application Executable Test has been successful, or if no demonstration application executable exists, a Custom Driver Installer Test has been successful.

The Demonstration Application Executable Test is not a mandatory prerequisite for performing this test, but having one available produces the best outcome. A successful demonstration test confirms that the two installers *used together* deliver all the functionality the provider intended, and the normal operation of the sensor can be observed.

If a Demonstration Application Executable Test is not available, as is the case with many providers, this test can still be performed if there has been a successful Custom Driver Installer Test. If there is no demonstration application executable, this knowledge is not available. Consequently, in *addition* to the uncertainty in correctly integrating the sensors in the custom application, there is uncertainty in the ability to accomplish all the capabilities of the provider's installer when the two installers are used together. Errors that arise during this test without the benefit of a prior successful Demonstration Application Executable Test will be harder to segregate.

When the host pretest environment for this test is set, the minimal software necessary includes a compilation tool. Therefore, the application of the predetermined development configuration should be noted.

Pass/Fail Criterion

This test is a "Pass" if the application compiles and is operational. *[The test author shall state specifically what evidence needs to be observed to deem the custom application operational.]*

[The "Pass" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration. Some of these steps may have been established in a previous test. They may be repeated here.]

Otherwise, the test is a "Fail."

The test author will need to list specific behavior within the custom application that should be observed before scoring this test a "Pass."

6.B.5 Custom Application Compile-Only Test

The following are special considerations for a Custom Application Compile-Only Test.

Purpose

To determine if the custom installer allows (installers allow) the custom application to compile without drivers or run-time dependencies present

This test uses the SDK installer to install compilation resources. It is intended to verify that the custom installer will execute and perform the installation of its resources correctly and that the components installed by this installer include all the resources required by the custom application to successfully compile. Passing this test does *not* mean that the application will execute correctly but rather that a complete set of resources are present and installed correctly that facilitate successful compilation *absent of run-time dependencies*.

Execution of the application is out of the scope of this test but specifically tested in the Cross-Compilation Test.

Prerequisites

The tester shall verify all of the following:

1. a predetermined development configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. a Custom Application Compile and Run Test has been successful.

[If the prerequisite Custom Application Compile and Run was unsuccessful, this test cannot be performed.]

A prerequisite to this test performs compilation and execution of the custom application. If it was determined a pass, then there is evidence that the integration is successful via the combination of resources installed by both installers used together. However, because both installers are used, there is no certainty that the SDK installer alone, absent of any run-time resources, supplies the correct selection of and complete set of components necessary for the application to compile without errors. Identifying the set of compile-time dependencies and achieving successful compilation (independent of run-time dependencies) is an integral step towards enabling cross-compilation of a custom application.

Because this test focuses on compilation, it is critical that its pretest environment include a compilation tool.

Pass/Fail Criterion

This test is a "Pass" if the application compiles without fatal errors.

Otherwise, the test is a "Fail."

Evidence of compilation without fatal errors should be unambiguous in a compilation tool. If the evidence is not clear, or if other non-fatal errors should trigger a failure, the test author should add those directions to the test that specify how to judge the pass criteria.

6.B.6 Cross-Compilation Test

The following are special considerations for a Cross Compilation Test.

Purpose

To determine if the custom installers enable the custom application to compile on one system (compilation host) and successfully run on a different system (execution host)

This test represents the ultimate goal of the custom installers and the accompanying tests. It is run using two separate machines: a compilation host and an execution host. The custom application is compiled, packaged, and moved from the compilation host to the execution host. When launched, if it opens and executes as intended, then cross-compilation is successful.

Prerequisites

Compilation Host

The tester shall verify both of the following:

1. a predetermined development configuration has been applied to the host
2. a Custom Application Compilation-Only Test has been successful.

Execution Host

The tester shall verify all of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. a Custom Application Compile and Run Test has been successful.

The Cross Compilation Test is a two-stage test. In the first stage, the custom application is compiled on the compilation host. In the second stage, the output of that compilation is run on the execution host. Therefore, this test's prerequisites correspond directly to the machines upon which the two stages of the test are performed.

The first segment of this test is equivalent to the execution of the Custom Application Compilation-Only Test. It is included for completeness, so that this test can stand alone. It follows that if that prerequisite is not met, the Cross-Compilation Test cannot be performed.

The execution host requires the Custom Application Compile and Run Test because it establishes the evidence that the custom application can successfully execute in this environment using the custom installers.

The successful outcome of the second segment of this test is evidence of the application's execution without a compilation tool present, completing the final stage of cross-compilation.

The compilation host requires that the development configuration be applied to set its environment. Because the goal of the test is to achieve execution without a development tool present, the execution host's pretest environment is the baseline configuration.

7. Conclusions

In this paper, we presented a framework that developers can use to improve the process by which a biometric SDK is integrated into a development or application environment. It represents the culmination of the experiences and lessons learned over many years of working with biometric SDKs. We hope that developers find this framework useful in their own work, and that SDK providers will use it to improve the integration experience of their products.

8. Acknowledgements

The authors thank the Federal Bureau of Investigation's Biometric Center of Excellence; the Department of Homeland Security's Science & Technology Directorate; the NIST Comprehensive National Cybersecurity Initiative project; Ellie Abrams, President of ESA Editorial and Training Services, INC. and Frederick Byers, Information Technology Specialist of NIST Information Technology Laboratory/Information Access Division for their support of this effort.

9. Glossary

SDK: A ***S**oftware **D**evelopment **K**it* is a set of development tools used to facilitate communication and functionality (integration) between a device or program and a custom application (e.g., software package, software framework, hardware platform, computer system, video game console, operating system, or similar platform).

Compile-time: The compile-time phase is the phase of computing during which source code (readable by a person) is converted into machine code (understandable by a machine). The resulting machine code (binary) is sometimes referred to as an executable. The operations included in this phase are syntax analysis, various kinds of semantic analysis, and code generation.

Run-time: Run-time is the phase of computing during which a program is executing. A program is "executing" when it starts processing and reacting to input (whether reading from a file or the console; examining environment variables, command-line arguments, or registry entries; or reacting to mouse or other controller events).

DLL: A ***D**ynamic **L**ink **L**ibrary* (or shared library) in Windows is an executable file containing compiled code that an application loads at execution time. These shared libraries enable developers to reuse and modularize code. The compiled code can be used by multiple programs to perform particular tasks. DLLs can also be used during compile-time. Microsoft Windows provides DLL files that contain functions and resources that allow Windows-based programs to operate in the Windows environment (Microsoft Support Article ID: 87934, Last Review: September 24, 2011, Revision: 2.0, "Definition and Explanation of a .DLL file").

.exe: An ***exe**cutable* file is a Microsoft Windows installer binary file that a computer can directly execute. Unlike source files, executable files cannot be read by humans. A source file has to pass through a compiler or assembler to be transformed to an executable file.

.msi: The ***M**icro**S**oft **I**nstaller* file type is primarily associated with "Windows Installer file" by Microsoft Corporation. This file type is an engine for the installation, maintenance, and removal of software on modern Microsoft Windows systems. The installation information, and often the files themselves, are packaged in installation packages, loosely relational databases structured as OLE Structured Storage Files and the default .msi file extension.

Dependency: A dependency is a relationship in which a shared software library (i.e., dll, ISScript files) is required by a principal software package to run successfully.

IDE: An **I***ntegrated **D**evelopment **E**nvironment* is a software application that provides comprehensive facilities to computer programmers for software development. An IDE at a minimum consists of a source code editor, build automation tools and a debugger.

Dongle: A small hardware device that plugs into the serial or USB port of a computer. Its purpose is to ensure that only authorized users can use certain software applications. It is sometimes referred to as a "security key."

10. Test Templates (Appendix)

Templates are included in this appendix. They provide the format and guidelines to create custom installer tests. The following documentation conventions are used in the templates to provide directions to formulate Sensor Integration Tests for various kinds of sensors.

Instructions to the test author that describe the data to be inserted into a test field are shown in *[bracketed text]*. In some cases, this information is a standard instruction that applies to any sensor. In other cases, where noted, the instruction states that the test author should supply test content specific to and guided by the sensor type.

Text from the template that should be replicated in the test exactly as stated is shown in standard non-italicized font. This content is specific to the test and should remain the same for any sensor.

Example Sensor Integration Workbook Cover Page

Sensor Integration Workbook

[The cover page for the Sensor Integration Workbook should summarize the relevant host configuration data that is shared across all of the attached tests. The test author should craft a form for the relevant configuration characteristics to be recorded by the tester.]

Host Machine

[The tester should identify the host machine by name and/or asset number. In addition, the tester should specify whether the host machine is physical or virtual.]

Operating System

[The tester should identify the host machine's operating system, along with relevant updates and patches, and architecture bit-depth (i.e., 32-bit or 64-bit). Depending on the desired sensitivity, this could be anything from a full enumeration of all of the patches on the machine, or simply the date that the machine was last checked for updates.]

SDK Version

[The tester should fully identify the SDK under test. This should include the full name of the SDK, the full version number, and perhaps where the SDK was obtained.]

Target Hardware

[The tester should provide information on the biometric sensor that was tested, for example, the model, serial number, firmware version, and other relevant characteristics such as connection type (e.g. USB or Firewire).

Template for Out-of-Box Installer Tests

Out-of-Box Installer Test

Purpose

To determine whether the installer(s), provided "as is" to the developer, is (are) sufficient for basic operation of a sensor.

[If there is no Out-of-Box Installer, this test is skipped.]

Installers Tested

[The test author shall specify the path and filename of installer(s), relevant command line arguments, and other relevant information, such as security requirements.]

Prerequisites

The tester shall verify both of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*.

Pass/Fail Criterion

This test is a "Pass" if the sensor is fully functional within the target environment.

[The test author shall state specifically how to determine if the sensor is "fully functional." The "Pass" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration.]

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.

In this test, these steps are typically explicit instructions on how to exercise the provided installer.

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]

Template for Custom Driver Installer Tests

Custom Driver Installer Test

Purpose

To determine whether the custom installer correctly installs (installers correctly install) the biometric sensor device driver *[or, if device drivers are built-in to the baseline configuration to determine whether the baseline configuration can correctly recognize the biometric sensor]*

Installers Tested

[The test author shall specify the path and filename of installer(s,) relevant command line arguments, and other relevant information, such as security requirements.

If device drivers are built in to the baseline configuration, then no custom installers will be explicitly tested.]

Prerequisites

The tester shall verify all of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. *[If applicable]* an Out-of-Box Installer Test has been successful.

Pass/Fail Criterion

This test is a "Pass" if the host operating system correctly recognizes the biometric sensor.

[The test author shall state specifically how to determine if the sensor is "correctly recognized." The "Pass" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration. Some of these steps may have been established in a previous test; they may be repeated here.]

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.

In this test, these steps are typically explicit instructions on how to execute the custom driver installer.

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]

Template for Demonstration Application Executable Tests

Demonstration Application Executable Test

Purpose

To determine, via a provided demonstration application executable, whether the custom installer correctly installs (installers correctly install) the biometric sensor device drivers *[if applicable]* and the SDK

Installers Tested

[The test author shall specify the path and filename of installer(s), relevant command line arguments, and other relevant information, such as security requirements.

Prerequisites

The tester shall verify all of the following:

1. a demonstration application executable is available
2. a predetermined baseline configuration has been applied to the host
3. the biometric device is *[or is not]* connected to *[the host or its network]*
4. *[If applicable]* a Custom Driver Installer Test has been successful.

[If the SDK does not include a demonstration application executable, then this test is skipped.]

Pass/Fail Criterion

This test is a "Pass" if both of the following occur:

1. The host operating system correctly recognizes the sensor. *[The test author shall state specifically how to determine if the sensor is "correctly recognized."]*
2. The demonstration application executable operates correctly. *[The test author shall state specifically what evidence needs to be observed to deem the demonstration application executable operational.]*

[The "Pass" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration. Some of these steps may have been established in a previous test; they may be repeated here.]

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.

In this test, these steps are typically explicit instructions on how to execute the demonstration application executable.

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]

Template for Custom Application Compile and Run Tests

Custom Application Compile and Run Test

Purpose

To determine whether the custom installers allow a custom application to be compiled and run successfully

Installers Tested

[The test author shall specify the path and filename of installer(s), relevant command line arguments, and other relevant information, such as security requirements or other installers that are triggered within the target development environment.]

Prerequisites

The tester shall verify all of the following:

1. a predetermined development configuration has been applied to the host
2. the biometric device is *[or is not]* is connected to *[the host or its network]*
3. a Demonstration Application Executable Test has been successful, or if no demonstration application executable exists, a Custom Driver Installer Test has been successful.

Pass/Fail Criterion

This test is a "Pass" if the custom application compiles and is operational. *[The test author shall state specifically what evidence needs to be observed to deem the custom application operational.]*

[The "Pass" criteria, to be determined according to the functionality and evidence desired, may be unique to the SDK under integration. Some of these steps may have been established in a previous test; they may be repeated here.]

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

At this point, "Pass" indicates that the sensor is fully capable of being integrated within a custom application.

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.

In this test, these steps are typically explicit instructions on how to compile and execute the custom application.

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]

Template for Custom Application Compile-Only Tests

Custom Application Compile-Only Test

Purpose

To determine whether the custom installer allows (installers allow) the custom application to compile without drivers or run-time dependencies present.

Installers Tested

[The test author shall specify the path and filename of installer(s), relevant command line arguments, and other relevant information, such as security requirements or other installers that are triggered within the target development environment.]

Prerequisites

The tester shall verify all of the following:

1. a predetermined development configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. a Custom Application Compile and Run Test has been successful.

[If the prerequisite Custom Application Compile and Run Test was unsuccessful, this test cannot be performed.]

Pass/Fail Criterion

This test is a "Pass" if the custom application compiles without fatal errors.

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.

In this test, these steps are typically the same steps used to accomplish compilation in the Custom Application Compile and Run Test.

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]

Example Cross-Compilation Test Cover Sheet

Cross-Compilation Test

[The cover page for the Cross-Compilation Test should summarize the relevant configuration data for a particular test (or a set of tests that share the same configuration information). The test author should craft a form for the relevant configuration characteristics to be recorded by the tester.]

Application Under Test

[The tester should provide information that uniquely identifies the name and particular version of the application under test. Ideally, this information would be comprehensive enough to trace the application under test to a specific build and source code snapshot.]

Host Machines

[For both the compilation host and the execution host, the tester should identify the host machine by name and/or asset number. In addition, the tester should specify whether the host machine is physical or virtual.]

Operating System

[For both the compilation host, and the execution host, the tester should identify the host machine's operating system, along with relevant updates and patches, and architecture bit-depth (i.e., 32-bit or 64-bit). Depending on the desired sensitivity, this could be anything from a full enumeration of all of the patches on the machine, or simply the date on which the machine was last checked for updates.]

SDK Version

[The tester should fully identify the SDK under test. This should include the full name of the SDK, the full version number, and perhaps where the SDK was obtained.]

Target Hardware

[The tester should provide information on the biometric sensor that was tested. For example, the model, serial number, firmware version, and other relevant characteristics such as connection type (e.g. USB or Firewire.).

Template for Cross-Compilation Tests

Cross-Compilation Test

Purpose

To determine whether the custom installers enable the custom application to compile on one system (compilation host) and successfully run on a different system (execution host)

Installers Tested

[The test author shall specify the path and filename of installer(s), relevant command line arguments, and other relevant information, such as security requirements.]

Prerequisites

Compilation Host

The tester shall verify both of the following:

1. a predetermined development configuration has been applied to the host
2. a Custom Application Compilation Test has been successful.

Execution Host

The tester shall verify all of the following:

1. a predetermined baseline configuration has been applied to the host
2. the biometric device is *[or is not]* connected to *[the host or its network]*
3. a Custom Application Compile and Run Test has been successful.

[If the applicable prerequisite tests were unsuccessful, these tests cannot be performed.]

Pass/Fail Criterion

This test is a "Pass" if the executable successfully runs on the execution host.

Otherwise, the test is a "Fail."

☐ Pass ☐ Fail

Instructions

[The test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with steps that exercise the "Pass" criteria of the test.]

Compilation Host: *[This segment typically has the same steps used to accomplish custom application compilation in the Custom Application Compile-Only Test.]*

Execution Host*: [In this segment, the test author shall specify step-by-step instructions that begin with the step after the prerequisites have been applied and end with the steps that execute the custom application. The custom application installation is performed using the build produced on the compilation host during segment one of this test.]*

For devices that must be manually connected, the point at which the connection happens can be significant. The test author shall include in these instructions whether that makes a difference and, if so, when the connection must be done.]