Agency Connection and State Management Guidance

Version X.X
MM/DD/YYYY

Contents

1. Background

1.1. Document Purpose

The purpose of this document is to provide guidance and information regarding connection type, card state management, and resource management that developers can use when writing libraries and applications that use a smart card. This would include libraries such as a BSI, a PKCS#11 module, and a CSP. It is also applicable to any application that connects to a Smart Card through one of the aforementioned libraries as well as any application that uses a PC/SC implementation to connect to the card directly. This guidance will help ensure more consistent behavior among middleware and applications to ensure that multiple applications will be able to run on the same desktop as well as to ensure that certain minimal security precautions are taken into account during software design.

More consistent behavior is advantageous to both middleware vendors and application designers, and ultimately to the forces who will use the Smart Card. Application designers will be able to create software that is not dependent on one implementation of the middleware. They will also be able to have some reassurances that other Smart Card-enabled applications on the same desktop will not cause unwanted side effects with their application. Middleware vendors will have a better understanding of their customer's needs and will be able to develop to those needs. This type of infrastructure in critical to a true multi-application environment.

1.2. Audience

1.2.1. Application Developers

This document provides guidance and lessons learned for application developers who will be connecting to both a Smart Card through PC/SC and those using a higher level interface such as the BSI, a CSP, or a PKCS#11 module. Some sections on connection modes may not apply directly to those application developers who are using middleware to connect to the Smart Card, but the information is still useful to help application developers understand the different modes that middleware may use and to take the possible side effects into account when designing applications.

1.2.2. Middleware Developers

This document may be most useful to middleware developers since most application developers will use some sort of middleware to connect to the Smart Card and rely on the middleware developers to make the proper decisions regarding card connection and state management and handle issues like resource contention and security.

1.3. Document Scope

This document focuses primarily on software that runs on the Microsoft Windows operating system and describes connection modes that are specific to the Windows Smart Card Subsystem, though the general principles apply to other operating systems as well. In addition, we are focusing exclusively on Smart Card related applications and issues rather than general smart card issues.

1.4. Document Objectives

The objectives are as follows:

- To inform developers about preferred connection methods
- To ensure more consistent behavior from middleware
- To ensure that multiple Smart Card-enabled applications work together on the desktop

1.5. Assumptions and Constraints

Since this document is intended primarily for software developers who create software for the Smart Card, it is assumed that the audience is already familiar with the basics of a Smart Card such as how to send commands to the Smart Card, which applets are on a Smart Card, their purposes and their command sets.

2. References

2.1. Agency

2.1.1. Middleware Specifications

2.2. Federal

2.2.1. GSC 1x thru 2x

2.3. Industry

2.3.1. PC/SC

PC/SC is not technically a Microsoft specification. Microsoft spearheaded the effort, but the PC/SC specification is maintained by the PC/SC Workgroup (http://www.pcscworkgroup.com/). Microsoft's smart card subsystem is simply based on this PC/SC specification.

Smart Card Connection and Card State

2.4. Connection Mode

The connection mode is managed by the Windows Smart Card Sub-System which is based on the PC/SC specification and determines several variables about how a logical connection to the Smart Card is made through the various software layers that an application uses. The most important of these variables is whether the logical connection is exclusive or shared.

Shared mode allows multiple processes to create logical connections to the Smart Card simultaneously. This means that any process with a logical connection can send commands to the Smart Card. Having at least one shared mode connection also prevents any other process from establishing a connection in exclusive mode.

Exclusive mode means that no other process can access the Smart Card while the connection is active. This means that any other process that attempts to create any type of logical connection to the Smart Card will receive an error message indicating that the resource is not available.

2.5. Card State

The card state is a concept composed of several variables relating to the Smart Card's chip. Card State is important to take into account since the output of the card is dependent upon both the current state of the card as well as the next command sent to the card.

The variables that determine the card state are too numerous to list here, but they consist of information such as: is the card powered up or reset or powered off, has user authentication been performed or not, which applet is currently selected, etc. Applications and/or middleware must keep track of the state that the card is in so that they will know what command to send to the card next, in order to receive the desired output. This information must be tracked during the use of the card since there is no easy way to query every variable that constitutes the current state of the card.

2.6. Why Card State and Connection Management are Important to Application Developers

2.6.1. Resource Contention

It is important to remember that the Smart Card itself can only be used by a single process at any given time. Typically, middleware gives the impression that a Smart Card can be used by multiple processes simultaneously, but in reality, the Smart Card's operating system and applets are single threaded. Only a single applet can be selected at any given time and only one APDU at a time can be received by the Smart Card. This is not a problem that is unique to the Smart Card. This situation is analogous with many hardware devices, some examples are printers and disk drives.

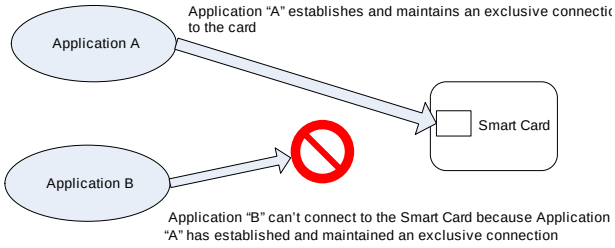

Figure 1 - Application Lock Out

Any software which uses the Smart Card must take this fact into account. Accordingly, the design of the software should be "friendly" to other processes which may want to use the Smart Card by not retaining exclusive use of the card for an extended period. Care must also be taken to avoid a deadlock situation. (For example, Process A is using the Smart Card and waiting on an output from Process B before it releases the Smart Card, but Process B cannot provide the required output until it can acquire exclusive access to the Smart Card.)

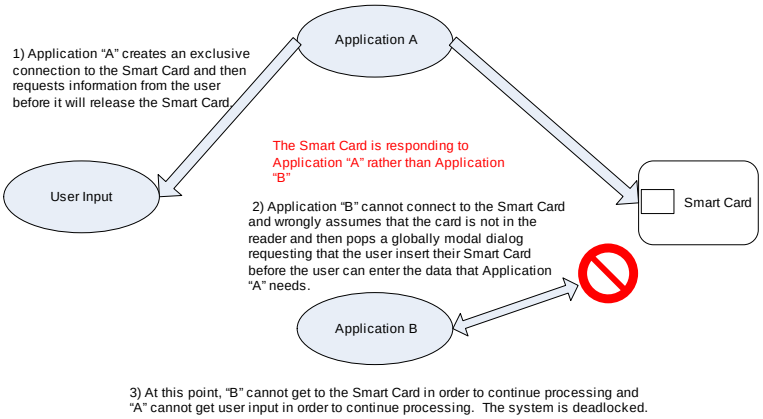

Figure 2 - Application Deadlock

2.6.2. Unknown Card States

There are several ways that a card can be inadvertently put into an unknown state which may have a variety of impacts from simply requiring the application to have to restart the process to corrupting data on the card or rendering it unusable. One example is that if Applications A and B both make simultaneous logical shared connections to the Smart Card, the state of the Smart Card can become unknowable to both applications and under certain circumstances, the Smart Card data could potentially become corrupted.

Consider the following example involving Applications A and B from above. A's job is to modify a certain data element in the Personnel Container and B's job is to simply read some information from a different Container Applet. First, A and B both create a logical shared connection to the Smart Card. This means that either process can send commands to the card at any time. The commands are received and processed sequentially by the Smart Card. Then, A first selects the Personnel Container Applet on the Smart Card, reads the T-buffer, finds where the data is in the V-buffer, reads out the data that it intends to modify and then performs some time intensive processing. Meanwhile, B selects a different container and reads the data that it is interested in. By this time, A has finished its processing and is now ready to write the data back. A simply sends the Update Binary command to the Smart Card, assuming that the Personnel Container applet is still selected since it does not know that B has selected a different container applet. If the correct security conditions were satisfied by B, then A has now corrupted the data in the container that B had selected.

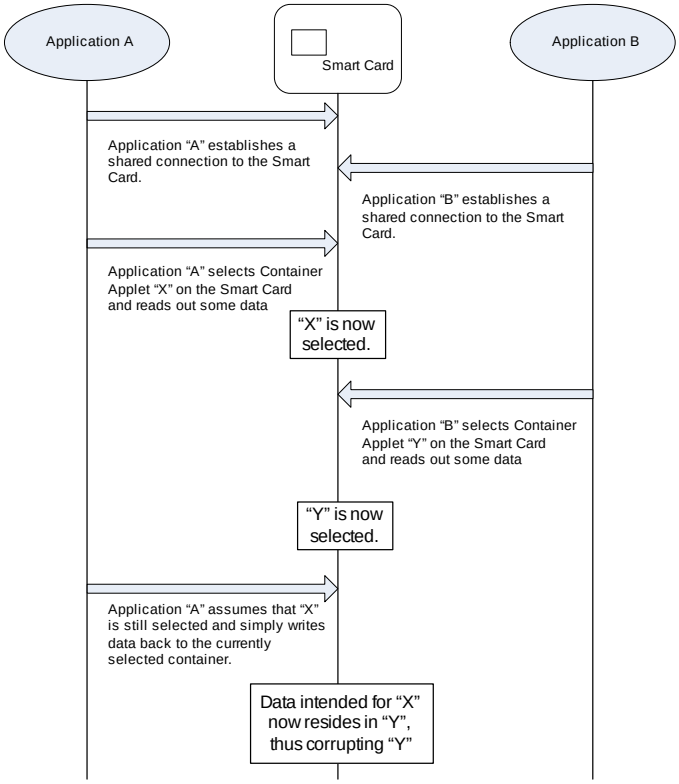

Figure 3 - Data Corruption Example

2.6.3. Security

This example also allows us to draw some conclusions regarding security and using shared mode to connect to a card. In the example above, it may be the case that A did not have permission to write to the container that B had selected, but since the connection was not exclusive, B essentially unlocked the applet allowing any other application to modify data that should have required an authentication.

A similar security problem can exist even when an application uses an exclusive connection to a Smart Card. Lets say that B connects to the card in an exclusive mode and satisfies the necessary security conditions for the container that it is using, completes its Smart Card related processing, and releases the connection to the Smart Card without powering down or resetting the Smart Card. This leaves the Smart Card powered up and in an authenticated state which means that the next application that connects to the Smart Card will find it in an authenticated state and will thus not have to satisfy security conditions in order to access what would otherwise be protected functionality on the Smart Card.

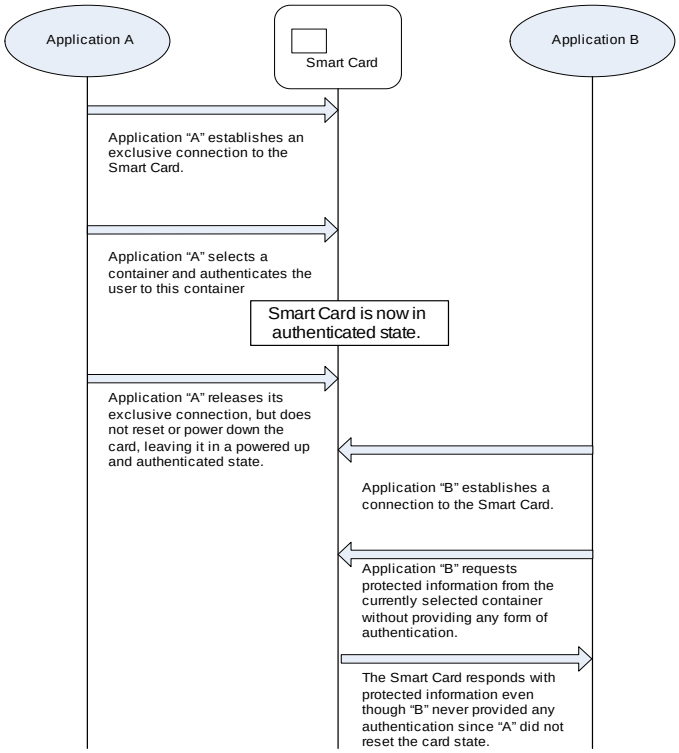

Figure 4 - Security Problem

2.7. Application Implications

The examples above refer to applications which form their own connections to the Smart Card through PC/SC. In these cases, some basic rules can be derived:

1. When an application will be modifying data on a Smart Card or accessing functionality or data which requires a security condition to be met, then the application should form an exclusive connection to the Smart Card in order to adequately secure the card and prevent data corruption.

2. When an application is only reading unprotected data from a Smart Card, it is advisable to use an exclusive connection since other applications could interfere with the proper Smart Card command sequence and confuse the application, even though the Smart Card itself is still secured and has all of its data intact.

3. When an application which was using an exclusive connection, releases the Smart Card, it should power down or reset the Smart Card to ensure that the card is no longer in an authenticated state, thus allowing unrestricted access by other applications.

The exclusive connections referred to in the above cases may be made either by connecting to the card in *exclusive mode* or by connecting to the card in *shared mode* and bracketing the critical section of code with the SCardBeginTransaction() and SCardEndTransaction() functions.

For applications which connect to the Smart Card through middleware libraries, developers should be aware of how the middleware works with regard to connection mode so that applications do not unknowingly corrupt data or create security holes.

Care must also be taken to ensure that an application does not fail or cause unwanted side effects (such as a deadlock condition or popping a globally modal window which prevents the user from accessing any application on the desktop) if a Smart Card is not immediately available when requested. Thus, applications should provide some sort of basic allowance for resource contention with other applications as well as ensuring that the Smart Card as a resource is not held onto for too long so that other applications which may be waiting will be able to access the resource.

2.8. Middleware Implications

There are also several implications for middleware. Application developers often do not have access to the source code or specifics regarding the internal workings of middleware and thus depend on middleware developers to make appropriate choices regarding the management of logical connections to the Smart Card.

2.8.1. BSI

GSC-IS v2.1 now specifies begin and end transaction function calls as a standard part of a BSI. In order for these functions to truly lock access to a card from all other applications (not just the applications using the BSI), the BSI must create a logical exclusive connection to the Smart Card. This may be done either by connecting to the card in *exclusive mode* or by connecting to the card in *shared mode* and bracketing the critical section of code with the SCardBeginTransaction() and SCardEndTransaction() functions. Otherwise, each function call within the BSI may be treated as a separate transaction and there is no need for a continuous connection to the Smart Card. This ensures that access to the Smart Card will not be withheld unnecessarily.

2.8.2. P11 & CSP

P11 libraries and CSPs, like the BSI, must be able to use the Smart Card and address the needs of multiple applications without also maintaining an extended logical exclusive connection to the Smart Card. Since these libraries focus on the use of the cryptographic services of the Smart Card, it is especially important to ensure that the card does not remain in an authenticated state for longer than necessary.

2.9. Impact of the BSI 2.1 Transaction Locking Functions

2.9.1. Definition

The GSC-IS 2.1 specification includes two new function calls in the BSI regarding transaction locking. One call begins an exclusive transaction with the Smart Card and the other call ends the exclusive transaction with the Smart Card. The purpose of these calls it to allow for an application which is using the BSI to connect to the Smart Card to gain exclusive access to the Smart Card, thus blocking access by not only all other applications attempting to gain access directly through PC/SC directly, but also to all other applications which might be using the BSI or any other middleware library concurrently.

2.9.2. Limitations

These function calls do allow application developers who use the BSI to connect to the Smart Card with some level of control over the connection mode. The obvious limitation of this mechanism is that it provides connection control only for those applications that use the BSI. There may be cases where use of the BSI is unavailable or inappropriate.

The other limitation is that it is difficult for a developer to tell which connection modes are used at what time when using middleware and so application developers must be careful not to assume that a connection is either exclusive or shared.

3. Smart Card Connection Modes

3.1. PC/SC connection modes

The first and most important decision to be made when establishing a connection with a card is the choice of which Share Mode to use. Share Mode determines how the PC/SC resource manager performs resource control for a particular card. There are three possible values for Share Mode.

"Shared" mode is denoted by SCARD_SHARE_SHARED and means that the connection to the card is non-exclusive and that other processes can create a simultaneous valid connection to the same card and send commands to the card at the same time. This is generally not good practice for several reasons. Shared mode can pose a security problem since this mode of operation can allow one application to perform cardholder authentication while another application could access protected resources on the card without submitting authentication. It should also be noted that creating a shared connection precludes any other application from forming an exclusive connection; therefore not even shared mode should be used for any longer than absolutely necessary.

Note: While using SCARD_SHARE_SHARED, the functions SCardBeginTransaction() and SCardEndTransaction() can be used to effectively create a logical exclusive connection even though the connection mode is shared. When SCardBeginTransaction() is called, the connection state effectively becomes just like an exclusive connection.

"Exclusive" mode, denoted by SCARD_SHARE_EXCLUSIVE, means that the PC/SC resource manager will not allow any other process to create a simultaneous connection to the same card. This prevents the two problems that can occur with shared mode, but introduces a new problem. When a process is using exclusive mode, the card resource is unavailable until the process disconnects from the card. This means that care must be taken to prevent excessively long wait times and the possibility of deadlocks.

There is one other rarely used method for connection to a card. This is the "Direct" connection, denoted by SCARD_SHARE_DIRECT. This method, also provides an exclusive connection, but it requires that the application provide the protocol negotiation. This is not necessary for any currently available Smart Card and is recommended for use only by advanced smart card application developers in special cases.

There is one other parameter in the SCardConnect function that is sometimes confusing to programmers who are not familiar with smart cards. This is the Preferred Protocol parameter. There are two basic types of communication protocols called T=0 and T=1. These refer to a value returned by the card's ATR. The details of each protocol will not be described in this document because they are already specified by ISO-7816 and the details are mostly irrelevant to application development for the Smart Card.

The best practice is to allow both protocols to be used because most modern readers will support both protocols. This can be done by performing a bitwise or operation on the two possible values: SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1. There is one other possible value for the protocol parameter: SCARD_PROTOCOL_UNDEFINED. This is only used when performing a direct connection to the card with the SCARD_SHARE_DIRECT share method. As with the direct sharing method, this protocol directive typically doesn't need to be used for applications which are written for the Smart Card.

3.2. The importance of choosing the right mode

Generally, the best practice for connecting to a smart card, is to use exclusive mode to prevent security problems and to ensure that the card remains in a known state while making the duration of each connection as short as possible.

Using shared mode is generally not good practice for several reasons. Shared mode can pose a security problem since this mode of operation can allow one application to perform cardholder authentication while another application could access protected resources on the card without submitting authentication.

Another potential problem with shared mode is that if two processes try to send two different sets of commands to the same card at the same time, the commands can be received by the card out of sequence, and possibly put the card into an unknown state.

4. Smart Card State Management

4.1. Maintaining the Smart Card in a known state

As mentioned earlier in this document, connection management and state management are intimately intertwined and must both be considered to ensure that Smart Card related software does not corrupt data or create any security holes.

The best way to ensure that an application or middleware can maintain the Smart Cardin a known state is to ensure that only a single process is sending commands to the Smart Card at any given time. This requires that an exclusive connection be used.

If an application is using middleware instead of connecting to the Smart Card directly, then it is much harder to ensure that an exclusive connection is being made. There are some safeguards that applications can perform to minimize (but not eliminate) the possibility of the Smart Card getting into an unknown state.

In the example in section 3.2.2, application A assumes that it is the only application using the card and that between the reading of the container and the updating of the container data that the card is in the same state (same applet selected and same authentication state). In the example, Application B has actually selected a different applet in between these two operations by A.

One way that A can minimize the likelihood of this sort of condition is by using the Get Properties function of the currently selected applet. This would tell A which applet is currently selected and if it is not the applet that A had originally selected, then it can be assumed that some other process is sending commands to the card at the same time and take appropriate steps to attempt to get the card back into the state that A requires to update the data.

Note: Since the BSI may be called simultaneously from multiple applications, the DLL should be threadsafe.

4.2. Resource Contention Solutions

There are several methods that can be used to manage the Smart Card as a resource via middleware, the details of which are outside the scope of this document. Though, the simplest way is for middleware to simply report to applications that a card is in use and that the requested resource is unavailable if an application requests it. This pushes the burden of waiting for a card to become available off to the application level and it also removes the complexity of the middleware having to maintain context information about the Smart Card for each application.

5. Compliance Testing

5.1. TBD….