Interoperability Standards for ICCs and Personal Computer Systems

Part 7. Application Domain and Developer Design Considerations

Bull CP8, a Bull Company

Gemplus SA

Hewlett-Packard Company

IBM Corporation

Microsoft Corporation

Schlumberger SA

Siemens Nixdorf Informationssysteme AG

Sun Microsystems, Inc.

Toshiba Corporation

VeriFone,Inc.

**Revision 1.0-
December 1997**

.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.
NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.
BULL CP8, GEMPLUS, HEWLETT-PACKARD, IBM, MICROSOFT, SCHLUMBERGER, SIEMENS NIXDORF, SUN MICROSYSTEMS, TOSHIBA AND VERIFONE DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. BULL CP8, GEMPLUS, HEWLETT-PACKARD, MICROSOFT, SCHLUMBERGER, SIEMENS NIXDORF, SUN MICROSYSTEMS, TOSHIBA AND VERIFONE DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

Contents

.

1. Architecture, Scope

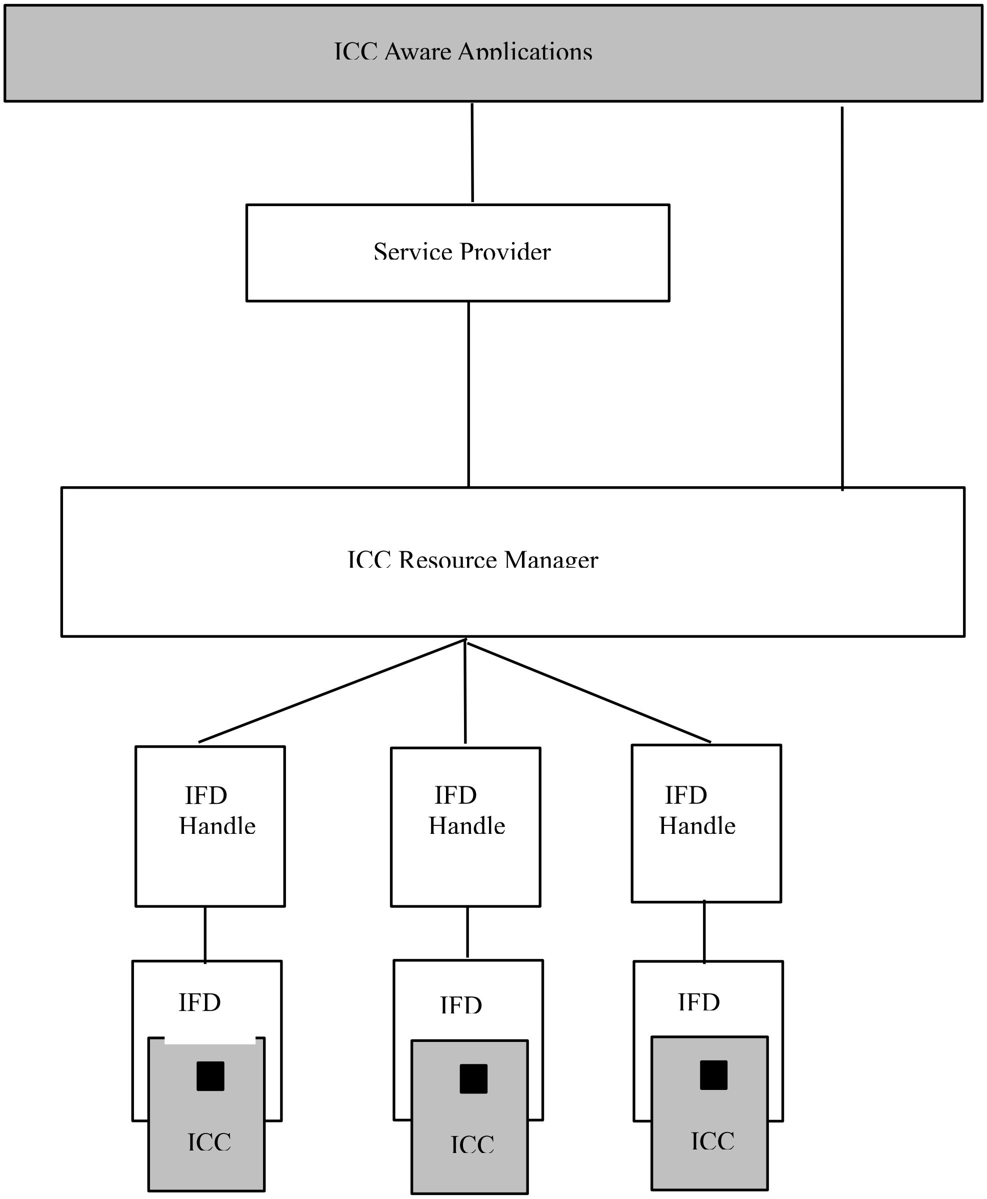

Figure 1

The software design considerations presented in this specification address the development of applications built on the architecture presented in Figure 1.

Part 7 describes the way ICC-aware applications can use the functionality provided by the ICC subsystem. By using the ICC Resource Manager and the ICC Service Provider layers, an application can use ICC functionality with some level of independence from a specific IFD, or to some extent, from a specific ICC.

Some general design considerations that the application developer should take into account are described in Section 2, “Implementation Considerations.” Section 3 describes the “Run-Time Considerations” that the application needs to take into account, as well as some example scenarios.

For a general overview of this architecture, please refer to Part 1, “Introduction and Architecture Overview” of the Interoperability Specifications. For a detailed description of the ICC Resource Manager, refer to Part 5 “ICC Resource Manager Definition.” For specific information about the ICC Service Provider and the functionality of your ICC, please refer to Part 6, “ICC Service Provider Interface Definition,” and to the documentation provided by the ICC vendor.

2. Implementation Considerations

2.1 Writing Applications Using Service Provider and Resource Manager Interfaces

In the described architecture, the Interface Device (IFD) subsystem (including IFD plus the handler) is provided by the IFD vendor, and the ICC subsystem (including the ICC and ICC Service Provider) is provided by the ICC vendor or ICC issuer. The application developer does not generally need to package any components related to ICC or IFD management.

An application developer who uses this architecture can access different services by using a functional API, or an object model interface such as C++, ActiveX, and Java.

Although this architecture is platform-independent, there are some general assumptions about the services provided by the operating systems for which an application is targeted:

- **Multithreading capability.** Most applications require two or more threads: one to monitor events through calls to **ScardTrack::GetStatusChanges()**, and one to perform the functional tasks of communicating with the ICC, handling UI messages, and so on.

- **Asynchronous event and message handling.** This is required for operations such as detecting card removal and insertion.

- **A shared library mechanism with dynamic linking to shared code.** This allows loading an ICC Service Provider, based on the type of ICC inserted into an IFD.

There are three ways to make use of this architecture:

1. **Use vendor ICCSPs.** When the supplied ICCSP interfaces correspond to an application's requirements, the application can (and should) use those ICCSPs to accomplish the requested actions on IFDs and ICCs. Interfaces for file access, authentication, and cryptographic services are defined in Part 6. ICCSP vendors may also provide proprietary interfaces for electronic purses, industry specific functions or special security functions.

2. **Develop a layered ICCSP.** When functionality not defined by vendor-supplied ICCSP interfaces is required, application developers can provide custom ICCSPs that meet their needs. They must, at minimum, expose the interfaces specified in Part 6. Domain-specific interfaces can be added to this ICCSP. This architecture allows multiple layers on a single system and also allows the implementation of ICCSPs to support multi-application ICCs.

3. **Access the Resource Manager directly.** This is useful when an application needs low-level control of the IFD or the ICC. Reserve direct Resource Manager access for very specific uses of IFDs or ICCs, such as personalization systems.

In these three cases, an application developer can access the IFD subsystem through the high-level interfaces implemented by the Resource Manager. No components related to IFD management need to be packaged with the application. In cases 1 and 2, there is no need to package any components related to ICC management, because this functionality is provided by the high-level interfaces of the ICC Service Provider.

When an application needs to manage a connection to an ICC, it uses the Resource Manager, defined in Part 5 of this specification, to locate the requested IFD or ICC. The application can then use the services provided by the ICCSPs.

The architecture provides the following methods for an application or ICCSP to connect to an ICC or IFD:

- The application can access whatever ICC is present in a specific reader. The Resource Manager provides an interface that allows you to connect to whatever ICC resides in a specified IFD. This is the simplest form of establishing communication with an ICC.

- The application can search for a specific ICC within a specified IFD group. When an application needs to interact with an ICC, it identifies the ICC by a friendly name, and specifies a group of IFDs in which the ICC may appear. The Resource Manager searches the group of IFDs for any ICC with an ATR string matching the named ICC, and returns status information to the application.

- The application can search within a specified IFD group for any ICC that provides a specified set of ICC interfaces. This is similar to the previous case, but any named ICC that supports all listed interfaces is considered a match.

When an application asks for an ICC, either by name or by interface, it supplies a list of IFD names in which it hopes to find the target ICC.

The application can use the context established by the preceding task to

- Use the ICCSP for high-level access to the ICC.

- Use the Resource Manager directly to have access to the low-level interface provided by the Resource Manager. This approach should be reserved for very specific applications, because it doesn't allow an application to operate independently of the IFDs or ICCs.

2.2 Advantages of using these interfaces

This specification is written using a system-independent formalism to promote and allow implementations that are independent of a specific programming language or computer architecture. This architecture is designed to allow maximum interoperability, upgradability, and flexibility. It allows you to create applications that do not need to target a specific ICC or IFD. End users can then run applications within this architecture by adding modules called "IFD handlers" and "ICC Service Providers," which are components available from the IFD and ICC vendors, ICC issuers or from third-party software providers. It allows different software vendors to provide ICCs, IFDs, and applications that can cooperatively use and share resources, thus enabling multiple ICC-aware applications to run simultaneously.

3. Run-Time Considerations

This section describes considerations related to run-time application behavior. It discusses the following tasks:

- Dynamic determination of available system resources
- Connecting to the desired ICC
- Device sharing
- Working with ICC security services
- Error recovery

This section provides information to aid application developers in understanding how to best meet their operational requirements. It discusses mechanisms for building several types of applications. As described in Section 2, there are three basic types of applications that may be developed to take advantage of ICC technology.

First, there will be applications designed to work with a specific IFD device. These may depend on a specific ICC type, or may work with several ICC types. This model is appropriate for applications such as ATMs, vending machines, public kiosks, and ICC personalization stations. Such applications would be designed to monitor a specific IFD and then respond after an ICC is inserted. This response could be triggered automatically, based on an ICC insertion event, or manually, based on user input.

A second kind of application is designed to work with a specific ICC, or a set of functionally equivalent ICCs. This approach is expected to be common, because applications tend to be designed to accomplish a specific objective. As such, these applications will require ICCs that contain appropriate application data or code. Examples could include a specific type of payment ICC, a group of similar payment ICCs, or a healthcare ICC.

Finally, it is possible to write applications that work with any ICC that supports a required interface. This approach is very reasonable for applications that use generic storage services or cryptographic services. In the future, it may also be possible to write applications that use standardized interfaces in markets such as payment, healthcare, and telecommunications.

3.1 Determining Resources

Applications will need to determine installed IFD characteristics and supported ICC types. This can be done either at installation time or run time, depending on the specific needs of the application. In addition, many applications will need to monitor ICC insertion and removal events at run time, and track the specific ICCs available. Using these basic services is described in the subsequent material. In addition, Section 2 discusses the use of these services to locate and connect to a desired ICC. The architecture allows the implementation of completely dynamic systems where support for specific ICC is loaded at run time through network services.

3.1.1 Determining Installed IFDs

To maximize flexibility, applications should be designed to work with the available IFDs, or an appropriate subset, on a system. While it is possible to create a static binding between an application and a specific IFD, this is not the recommended approach. Rather, applications should be designed to work with a specific IFD group or groups.

The notion of IFD groups was introduced in Part 5 of this specification. Groups provide a mechanism for specifying logical collections of IFDs. They will generally be created and managed by the administrator on a system (who may be the end user). The expectation is that groups will be fairly stable, with IFDs being added or removed over time. By using groups, applications can adapt at run time to changes in IFD resources.

Available IFD groups can be determined during the installation phase of an application or at run time. The list of defined IFD groups may be determined by:

1. 1. Instantiating an object of the **RESOURCEMANAGER** class by calling its constructor.

2. 2. Establishing a communication context to the ICC Resource Manager by calling **RESOURCEMANAGER::EstablishContext()**.

3. 3. Instantiating an object of the **RESOURCEQUERY** class by calling its constructor. The reference to the **RESOURCEMANAGER** object created at step 1 is supplied as a parameter.

4. 4. Determine the defined IFD groups by calling the **RESOURCEQUERY::ListReaderGroups()** method.

In general, the application should ask the user to specify the Group or Groups it is to work with. The Group names are arbitrary and may vary by system, so attempting to automatically configure the application based solely on Group names is not recommended. Note that if the application has a user interface, it should give the user the ability to select the default Group and reader. This setting will be saved in persistent storage, and can be modified at a later time. An application with no user interface can have a configuration program to accomplish this task.

When the application is invoked in the future, it can determine the specific readers within the Group(s) using the **RESOURCEQUERY::ListReaders()** method.

3.1.2 Determining Installed ICC Types

As described in Part 5, ICC Types are registered with the Resource Manager to make them available. For each ICC Type, the Resource Manager tracks the associated ICC ATR string and ATR mask, a friendly name for the ICC, a reference to the available Service Provider(s), and a list of supported interfaces.

To make use of a specific ICC Type, an application must know the appropriate Service Provider to use and its friendly name. An application can retrieve this information from the Resource Manager by the following process:

5. Instantiating an object of the **RESOURCEMANAGER** class by calling its constructor.

6. Establishing a communication context to the ICC Resource Manager by calling **RESOURCEMANAGER::EstablishContext()**.

7. Instantiating an object of the **RESOURCEQUERY** class by calling its constructor. The reference to the **RESOURCEMANAGER** object created at step 1 is supplied as a parameter.

8. Invoking the **RESOURCEQUERY::ListCardTypes()** method to enumerate the known ICC Types. The application developer can provide a list of known ATR strings associated with the ICCs of interest, a list of required standard interfaces, or no qualifications to this method. All ICC Types matching the specified qualifications are returned.

9. Retrieving the Service Provider reference for a specific ICC. To do this, use the **RESOURCEQUERY::GetProviderId()** method and the list of supported interfaces retrieved using the **RESOURCEQUERY::ListInterfaces()** method.

An application can determine this information at installation time if it is designed to operate only with a specific ICC. However, it is safer to retrieve this information at run time to insure that the application is working with information that reflects the current (true) state of the system.

3.1.3 Monitoring Card Events

Many applications need to monitor ICC insertion and removal events. As described later in this specification, monitoring card events may play an important part in locating the desired ICC. Monitoring is also important when allowing an application to take appropriate action in response to these events without requiring excessive user interaction. For example, an application could wait as a background task until an appropriate ICC is inserted, and then automatically connect to the ICC. Similarly, an application could automatically suspend itself, or terminate, if the user removes the ICC it requires.

To monitor insertion and removal events, use the following procedure:

1. Instantiate an object of the **RESOURCEMANAGER** class by calling its constructor.

2. Establish a communication context to the ICC Resource Manager by calling **RESOURCEMANAGER::EstablishContext()**.

3. Instantiate an object of the **SCARDTRACK** class by calling its constructor . A reference to the **RESOURCEMANAGER** object is provided as a parameter. Determine the IFDs of interest, as described in Section , and create an **SCARD_READERSTATE** structure for each IFD (see Part 5 for details).

```
typedef struct {
 STR Reader;         // IFD name
 VOID UserData;              // user defined data, may be NULL
 DWORD CurrentState;         // current state of IFD at time of
                             // call
 DWORD EventState;  // state of IFD after state change
} SCARD_READERSTATE;
```

4. Call the **SCARDTRACK::LocateCards()** method, supplying all ICC Types of interest as parameters to determine if a desired ICC is already inserted into an IFD. If the returned **SCARD_READERSTATE** structures indicate the presence of a desired ICC, skip to step 6.

5. To wait for a desired ICC type to be inserted, call the **SCARDTRACK::GetStatusChange()** method. Because this method will block until a change occurs in the state of one of the IFDs associated with the provided **SCARD_READERSTATE** structures, a separate thread should be used. Information on the nature of the change is returned in the **SCARD_READERSTATE::EventState** method.

6. If an insertion event has occurred for a specified IFD, the application can then determine which ICC Type was inserted by calling the **SCARDTRACK::LocateCards()** method. The type of ICC(s) of interest is provided as a parameter. If a desired ICC Type is not present, go back to the preceding step.

Currently, the final step (6), determining the ICC Type, cannot be guaranteed to uniquely identify an ICC Type. This is due to the lack of established standard methods for uniquely determining information identifying an ICC. The only information that can be retrieved with assurance from all ICCs is their ATR string. There are standards in ISO/IEC 7816-4 for encoding identifying information in the ATR history bytes. However, these are not used by a significant number of existing ICCs. Consequently, multiple ICCs can use identical ATR strings, but may be otherwise incompatible. However, it is highly recommended that ICC developers follow the ISO/IEC 7816 recommendations for ATR encoding. Only with an unique ATR the architecture can guarantee fully automated operation.

Application developers should be prepared to deal with the case in which multiple ICC Types map to the same ATR sequence. You can determine this by invoking the **RESOURCEQUERY::ListCardTypes()** method with the ATR for the desired ICC(s). In the event that multiple ICCs are indistinguishable using the ATR, applications should provide an appropriate user interface to allow the user to resolve such conflicts.

3.2 Connecting to the Desired ICC

A fundamental issue that each ICC-aware application must address is locating and connecting to the desired

ICC at run time. Application developers have a range of options in ways to perform this function. The architecture described in this specification allows the application developer to make the appropriate tradeoffs in terms of development complexity and flexibility. The subsequent material describes the various mechanisms available to application developers.

Connecting to Multi-application ICC

Multiapplication ICCs have an internal structure with an ICC carrying basic administration functions and multiple applications. A banking ICC may carry common applications like credit, debit, electronic purse, homebanking access and others. The architecture allows to build structured ICC service providers with application selection according to ISO 7816-5, EMV or other standards. However, in order to enable plug and play operation for ICCs and applications a common method for identifying the card operating system, the application selection method and the application is required. The PC/SC Workgroup will work with standards organizations and industry working groups to define a specification for management of multi-application ICCs.

3.2.1 Working with a Known IFD and ICC

The simplest approach to locating the desired ICC is to pre-configure the application (generally at installation time) to use a specific IFD and ICC Service Provider. You can determine the available IFDs, ICC Types, and associated Service Providers by using the **RESOURCEQUERY** object, as described in Sections and .

When the application is started, it simply retrieves the desired IFD name and a reference to the Service Provider from private storage as part of its initialization process. To connect to the desired ICC it:

10. Instantiates an object of class **SCARD** by calling the constructor exposed by the Service Provider.

11. Requests a connection to the ICC by invoking the **SCARD::AttachByIFD()** method.

Assuming this is successful, the application can then make use of the desired ICC services by calling the interfaces exposed by the Service Provider. When the application has finished using the ICC, it need only destroy the **SCARD** object to terminate its connection to the ICC and the Service Provider.

If the connection request returns an error, the application must request that the ICC be placed in the desired IFD, and then attempt to connect again. Because the application isn't monitoring insertion events, it must rely on the user to tell it when the correct ICC is inserted so it can attempt to connect.

The primary drawback to this approach is the lack of flexibility. Changes in the system configuration, such as installing a different IFD or renaming the IFD, will likely require that the application be reconfigured. Similarly, if the user receives an upgraded ICC, it is likely the application will need to be reconfigured to use a new Service Provider. In addition, this approach requires that the ICC be placed in a specific IFD. This may lead to some confusion and require extra ICC swapping on a system where the user has multiple IFDs and is using multiple ICCs.

This approach also limits the application developer in a couple of important ways. First, it forces the application to select a specific ICC Type to work with. This precludes the application from easily working with any functionally equivalent ICC that may be present (see Section 2). Second, it does not provide a means for the application to determine enhanced IFD features that a vendor may have provided, or to request use of those features. If this is important, the application must make use of the interfaces exposed by the Resource Manager, as described in the next section.

3.2.2 Working with a Known IFD

Working with a known IFD represents the next level of complexity in the use of preconfigured IFDs and Service Providers. In this instance, it is reasonable to pre-configure the application to work with a specific IFD, but the binding to a specific Service Provider (or ICC Type) is made at run-time.

This approach is reasonable if the typical user is assumed to have only a single IFD, or if a specific IFD will always be used for ICCs of a particular type. For example, a user may have a single IFD with secure PIN entry support required for the use of specific financial services. In this case, it is reasonable for an application supporting these financial services to require a specific IFD. The drawback, of course, is that changes in the available IFDs will likely necessitate reconfiguration of the application.

To connect to an ICC inserted in the designated IFD, the application:

12. Instantiates an object of the **RESOURCEMANAGER** class by calling the constructor.
13. Establish a communication context with the ICC Resource Manager by calling **RESOURCEMANAGER::EstablishContext()**.
14. Wait for ICC insertion for the designated IFD, as described in the section 3.1.3.

After determining that an ICC has been inserted, the application should determine if the ICC Type is one it is capable of working with. (This is most easily done by calling **SCARDTRACK::LocateCards()** with the *Cards* parameter sequenced through the ICC Types of interest). If it is an unrecognized ICC Type, the application should continue monitoring for the next insertion event. If it wants to work with the ICC, then the following actions are taken:

15. Instantiate an object of the **SCARDCOMM** class.
16. Open a connection to the ICC by calling **SCARDCOMM::Connect()**.

If this process is successful, the application is now connected to the ICC. It may now interact directly with the ICC by using the **SCARDCOMM::Read()** and **SCARDCOMM::Write()** methods. However, this requires detailed knowledge of the ICC implementation, and the application will most likely pass control to the appropriate Service Provider. This is done by:

17. Instantiating an object of the **SCARD** class by calling the constructor exposed by the appropriate Service Provider.
18. Passing control of the existing connection to the Service Provider by calling **SCARD::AttachByHandle(SCARDCOMM::hCard)**. Note that the **SCARDCOMM** object instance should be deleted only after the **SCARD** object to avoid prematurely destroying the communications context.

The application can then access the ICC services using the high-level interfaces exposed by the Service Provider. As an alternative, the application could simply call the Service Provider method **SCARD::AttachByIFD()** after the IFD and ICC Type are known. This is the preferred approach if all interaction with the ICC will be done using the Service Provider’s exposed interfaces.

It is generally not recommended for applications to mix calls to the Resource Manager **SCARDCOMM** object and Service Provider interfaces. If an application does this, it should make appropriate use of transaction support, and should pay careful attention to the ICC interface requirements to avoid unexpected results.

One reason an application may need to directly interact through an **SCARDCOMM** object is to control specific IFD features. An application can retrieve information about the IFD by calling the **SCARDCOMM::GetReaderCapabilities()** method. This method retrieves information for specific properties defined by the tag values given in Part 3 of this specification. The IFD vendor can define additional tags for proprietary features, such as a secure display or a PIN pad. The vendor must then specify specific control codes (see Part 3, Section 3.2.4) to access these features through the **SCARDCOMM::Control()** method. Also, using the **SCARDCOMM::SetReaderCapabilities()** method, an application developer can set values for the writable subset of the IFD properties.

3.2.3 Working with Known ICCs and IFD Groups

This represents a more general approach than those based on a statically determined IFD, though it is more complex to implement. In this case, it is assumed that the application knows both the ICC(s) and IFD

Group(s) it can work with. The major advantage of working with known ICCs and IFDs is that the application can connect to an appropriate ICC placed in any IFD within the designated Group(s). Also, altering the assignments of IFDs to Groups can be done without impacting operation of the application.

To determine the availability of a desired ICC Type, an application should proceed in the following way:

19. 1. Determine the IFDs within the Groups of interest, using the process described in Section .

20. 2. If the friendly names for the ICC Types of interest aren't known, then determine the ICC Types of interest using the process described n Section . The desired ICC Types can be determined by supplying the ATR string(s) associated with ICCs or searching for known friendly names.

21. 3. Wait for the insertion of one of the desired ICC Types using the process described in Section .

22. The application should, in general, provide user-interface support to aid the user in inserting the ICC into an appropriate IFD. If the application determines that no desired ICC is in an IFD, it is recommended that the application prompt the user to place the desired ICC into an appropriate IFD. The appropriate IFD(s) depend on whether there are empty IFDs, or the current state of IFDs that have ICCs inserted in them.

23. Using the methods of the **SCARDTRACK** object, the application can determine the state of all IFDs of interest. Hence, the application will know if any IFDs are empty. If there are empty IFDs, the user should be prompted to use one of them. If there are no empty IFDs, the application can indicate which IFDs contain ICCs that are not in use. The application can recommend that one of these IFDs be used for the new ICC. If all IFDs contain ICCs in use, then the application can do little beyond asking the user to swap out one of the ICCs currently in use.

After a desired ICC has been located in an IFD, the application connects to the ICC using the process described in the preceding section. In summary, the application has two options: it can connect directly through the appropriate Service Provider using the **SCARD::AttachByIFD()** method, or it may connect through the Resource Manager using the **SCARDCOMM::Connect()** method. The former is simpler, but allows less control to the application.

3.2.4 Working with a Known ICC Interface

This approach varies slightly from the preceding case. It is more general in that the application can work with any ICC that has a Service Provider supporting the required interfaces. This potentially allows the application to take advantage of new or upgraded ICC Types as they are introduced to the system, without any reconfiguration.

Use of this approach is only slightly more complex than the preceding case, the principle difference being that the desired ICC Types are determined by invoking the **RESOURCEQUERY::ListCardTypes()** method with a list of required Interfaces (see Section 2). From that point, establishing a connection to a desired ICC proceeds as described in Section 3.2.

3.3 Device Sharing and Control

Current (1997) ICCs are fairly simple devices with limited ability to manage multiple transaction streams. In this context, a transaction stream includes a sequence of commands to the ICC that may not be interrupted without destroying critical intermediate state data. ISO/IEC 7816 defines a “channel” mechanism for handling up to four logical streams between external applications and the ICC; however, few current ICCs support this. Most devices are designed to manage only a single stream.

When employed as part of a PC system supporting multiple concurrent applications, the limited ability to support multiple concurrent transaction streams imposes some significant limitations. It forces the applications or Service Provider developer to explicitly control device sharing to:

- Insure requests for ICC services are completed without error
- Allow other applications timely access to the ICC resource

To meet these objectives, it is recommended that applications use the following guidelines:

- Limit the use of “exclusive” access to an ICC. There are few cases where this is necessary. If you do require exclusive access, close the connection to the ICC as soon as the required ICC operations are complete.
- Use the Resource Manager transaction support for all operations requiring more than a single command-response operation.
- Limit transactions to the minimum number of commands necessary due to intermediate state requirements. That is, don’t pack multiple transactions within a single Begin-End Transaction request for convenience. This will likely block access by other applications for excessive time periods.
- Don’t “reset” an ICC unless absolutely necessary. In general, this should be done only if required to clear the ICC security state or to attempt recovery from a fatal error that has caused the ICC to become unresponsive.
- Close connections to ICCs before program termination.

3.4 ICC Security Services

One of the prime motivations for using ICC technology is the inherent security it provides for storing data that is private to either the user or application. Access to, and the ability to manipulate such information, is generally protected by one or more security mechanisms. Hence, the ability to interact with the ICC’s security interfaces can be of critical importance. The subsequent material discusses application development considerations in using these services.

3.4.1 Authentication Services

ICCs typically support user or application authentication services. User authentication, often referred to as CHV (CardHolder Verification), is typically done by having the user supply a secret code (or PIN) to the ICC. As described in Part 6 of this specification, Service Providers should support CHV requirements and take care of prompting the user at appropriate times. However, application developers may explicitly request that CHV be performed through a **CHVERIFICATION** object.

Many ICCs will additionally support the ability for applications to authenticate to the ICC using cryptographic methods. This encompasses the Internal Auth and External Auth mechanisms defined in ISO/IEC 7186-4, but may also include proprietary methods. If supported, a Service Provider will expose an interface to this service through a **CARDAUTH** object. The cryptographic keys and algorithms used to support these functionalities will be ICC-specific, and it is the responsibility of the application developer to obtain the required information from the appropriate vendor or ICC issuer.

3.4.2 Security State Management

ICCs maintain an internal security state that reflects current access rights. This security state is generally affected by the current file selection and prior authentication actions. Applications that access the ICC solely through a Service Provider can generally expect the Service Provider to handle required security state management. To determine the requirements for security state management in an application, the application develpoper should check the specifications for any domain-specific interfaces the Service Provider may implement. In addition, if the application works directly with the ICC through the Resource Manager to access functionality not exposed through a Service Provider interface, then the application must handle security state management.

Management of security state information is generally ICC-specific and application-specific. If an application needs to manage this state itself, obtain the necessary information from the ICC vendor or issuer. In general, achieving a desired set of access permissions will require stepping through a sequence of file selection and authentication actions in a prescribed order. This sequence should be done within a single transaction sequence to avoid the possibility of another application corrupting the desired state.

Because each security state implies specific access rights, it is also important for an application to clear the security state when a transaction is complete to prevent another application from taking advantage of this state. Clearing this security state can vary by ICC. In ICCs that follow the security state management guidelines in ISO/IEC 7186-4, the current security state may usually be cleared by selecting the MF of the ICC. In the worst case, clearing the security state may require resetting the ICC. If necessary, you can do this by setting the **SCARD_RESET_CARD** flag before calling the **SCARDCOMM::Disconnect()** or **SCARDCOMM::EndTransaction()** method.

3.4.3 Secure Messaging

Secure Messaging (SM), as defined in ISO/IEC 7816-4, encompasses the application of cryptography to solving the problem of data integrity, authentication, and confidentiality. Specific algorithms and cryptographic keys are ICC-specific or domain-specific, and to use SM features, you will need to obtain specific implementation information from the ICC vendor or issuer.

The services that the Service Provider supports determine the SM requirements in an application. For example, if the Service Provider internally implements required SM, the application simply invokes the methods exposed by the Service Provider, and SM will be applied as necessary. This puts the security burden on the Service Provider developers. They must insure that required cryptographic keys are distributed in accordance with domain security policy. In addition, they may need to validate the calling application to prohibit hostile processes from attempting to access or modify (critical) data on the ICC.

Although this approach raises some serious security issues, it is reasonable in many cases. As an example, we might assume that SM is used to protect critical ICC data from modification (such as an electronic cash value) unless accessed by authorized payment server applications software. In this instance, one option would be for the Service Provider developer to write a client/server Service Provider in which the portion of the Service Provider that implements SM resides only in the “trusted” payment server environment.

The other option is for the Service Provider to not support SM. In this case, the application software must contain the SM knowledge. The application developer must build the ICC commands and handle the ICC responses using the low-level read/write methods exposed by the Resource Manager **SCARDCOMM** class.

3.5 Recovering from Errors

Applications should be designed to provide robust handling of all possible error conditions. These can arise from a variety of causes, some that can be recovered from, and some that can’t. In the case of unrecoverable errors, the application should gracefully close its connection to the ICC and alert the user. The user then has the option of physically removing and reinserting the ICC to force a cold reset. Unless there is a hardware failure in the system, this should clear the error condition and allow operation to proceed. In the case of recoverable errors, the application should attempt to silently recover and continue operation. For example, removal of a card by the user during a WRITE operation. .

Possible errors and the recommended action are indicated in the following table.

Error/Warning Description	Error Code	Recommended Action
ICC reset by another application	SCARD_W_RESET_CARD	Reconnect to clear the warning and continue operation.
ICC removed by user	SCARD_W_REMOVED_CARD or SCARD_E_NO_SMARTCARD	Disconnect from the ICC. If the ICC is still required, ask the user to re-insert the ICC and reestablish connection when the ICC is detected.
ICC unpowered	SCARD_W_UNPOWERED_ CARD	Reconnect to clear the warning and repower the ICC.
ICC response not received	SCARD_E_TIMEOUT	Re-attempt last command twice. If the ICC remains unresponsive, attempt to reset the ICC.
ICC does not respond to reset	SCARD_W_UNRESPONSIVE_ CARD	Attempt to reconnect to the ICC to clear the warning. If this is unsuccessful, an application should attempt no more than a single additional reset attempt before assuming the error is unrecoverable.
IFD device has failed	SCARD_F_COMM_ERROR	Disconnect and alert the user. The User must correct the problem with IFD device.

4. Installation and Configuration

Applications shouldn’t rely on the names of IFDs, IFD groups, or ICCs presented during installation, because the names can be changed and new IFDs or IFD groups can be added during the lifetime of the application.

Information about the currently installed components can be obtained by database query calls to the ICC Resource Manager as described earlier. Applications can interact with the user to get the components they should work with, but this may not be comfortable for the user. A better solution would be a configuration dialog exposed within an options menu.

4.1 Installation & Software configuration

Configurable information for ICC-aware applications may include the following:

- The name of a specific IFD.
- The names of specific IFD groups.
- The name of a specific ICC.
- The name of the default IFD or IFD group.
- The name of a default ICC.
- Time-out values for actions like ICC insertion.