**JAVA Card OS**

A typical Java Card device has an 8- or 16-bit CPU running at 3.7MHz, with 1K of RAM and more than 16K of non-volatile memory (EEPROM or flash). High-performance smart cards come with a separate processor and cryptographic chip and memory for encryption, and some come with a 32-bit CPU.

The Java Card technology specification, currently in version 2.2, consists of three parts:

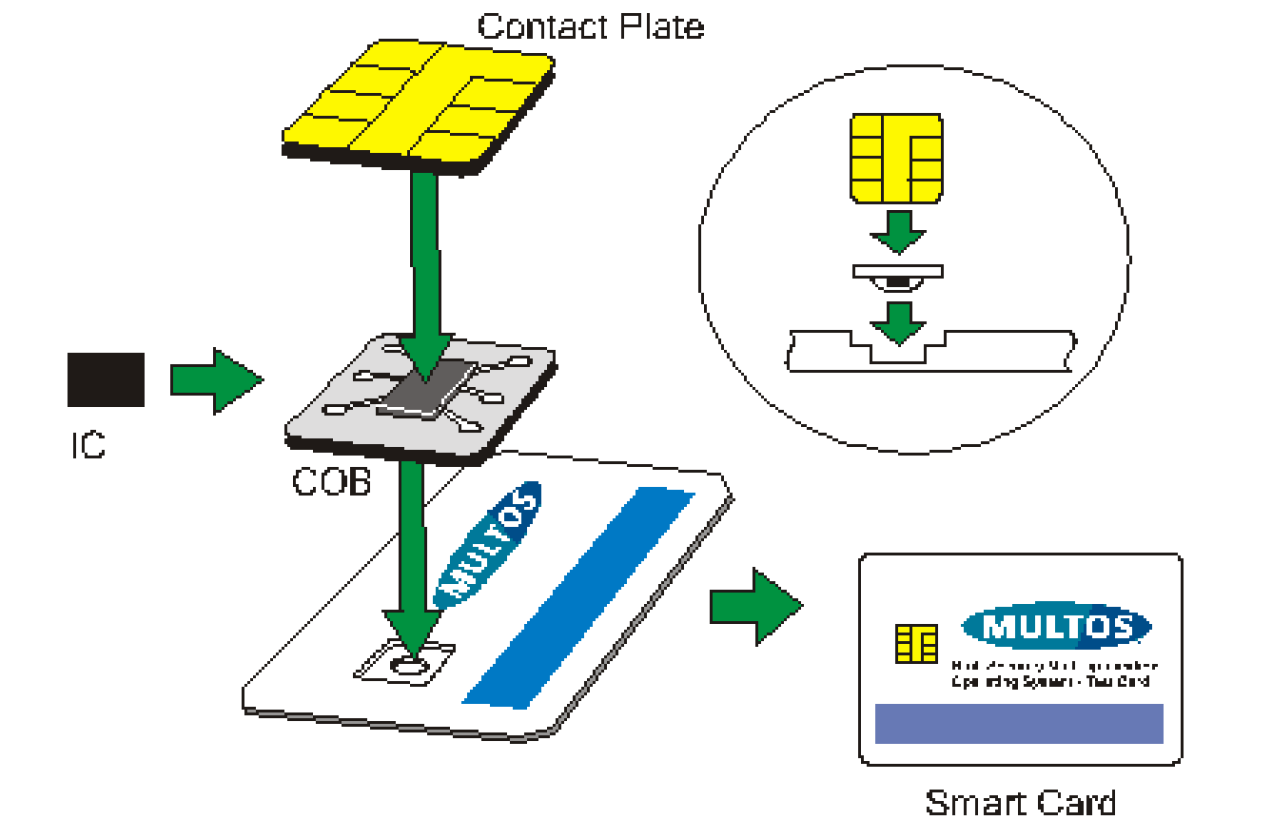
* The Java Card Virtual Machine specification, which defines a subset of the Java programming language and a VM for smart cards
* The Java Card Runtime Environment specification, which further defines the runtime behavior for Java-based smart cards
* The Java Card API specification, which defines the core framework and extension Java packages and classes for smart-card applications

Sun also provides the Java Card Development Kit (JCDK), which includes a reference implementation of the Java Card RE and Java Card VM, and other tools to help you develop Java Card applets.

A complete Java Card application consists of a back-end application and systems, a host (off-card) application, an interface device (card reader), and the on-card applet, user credentials, and supporting software. All these elements together compose a secure end-to-end application:

**MULTOS**

**1. Anatomy of a Smart Card and Smart Card Manufacture**

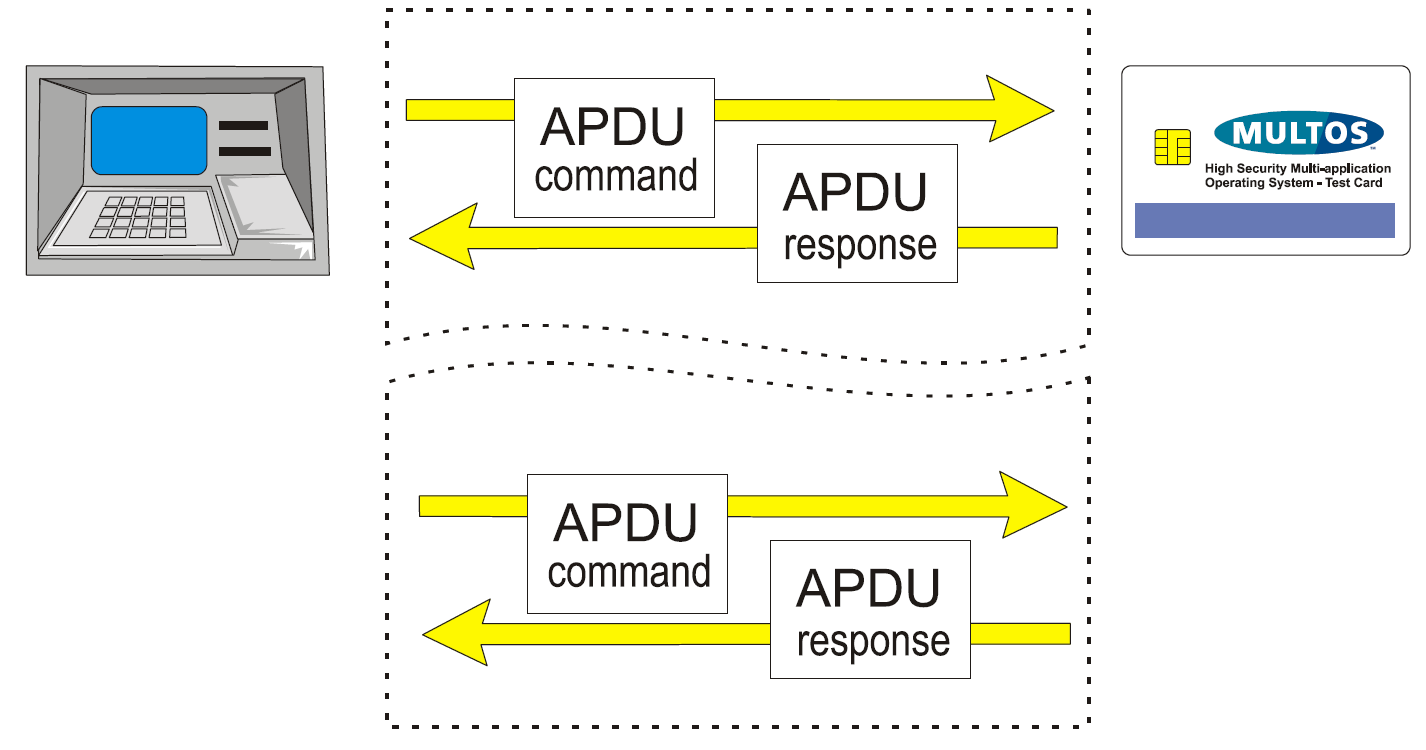
****

* **Single Application Smart Cards:** designed to carry only a single application and these cards display various levels of sophistication. The simplest can be considered memory cards. Single application cards successfully fulfil many smart card business cases. There are still some drawbacks. Once a card is manufactured it is difficult, if not impossible, to change the application on the chip. If a card issuer wished to offer several different applications to their cardholders, then a cardholder could have to be issued with two or more cards. The resulting cost to issuers may prove to be too high.
* **Multi-Application Smart Cards:** are programmable chip cards that allow multiple applications to be loaded onto the chip. Each application can run independently on the chip. So, a single chip card can be used to perform multiple and very different functions. It would be possible, for example, to have a single smart card that could serve as ID, a payment card and a holder of health records. Each of the application will have access to the necessary tools to cater for its own data security. Applications can be loaded and deleted from multi-application cards. This greater freedom allows cardholders and issuers to change the application mix on the chip during the normal lifetime of the card. It can also extend the useful life of a card. Finally, these cards have an operating system. An OS helps to create a known environment within which applications operate. It also facilitates loading and deleting as well as other common operations. Multi-application cards are not without their weak points. The cost of multi-application chip cards is higher than other types. More flexibility brings complexity, which means that there is an initial learning phase that may be longer than anticipated.

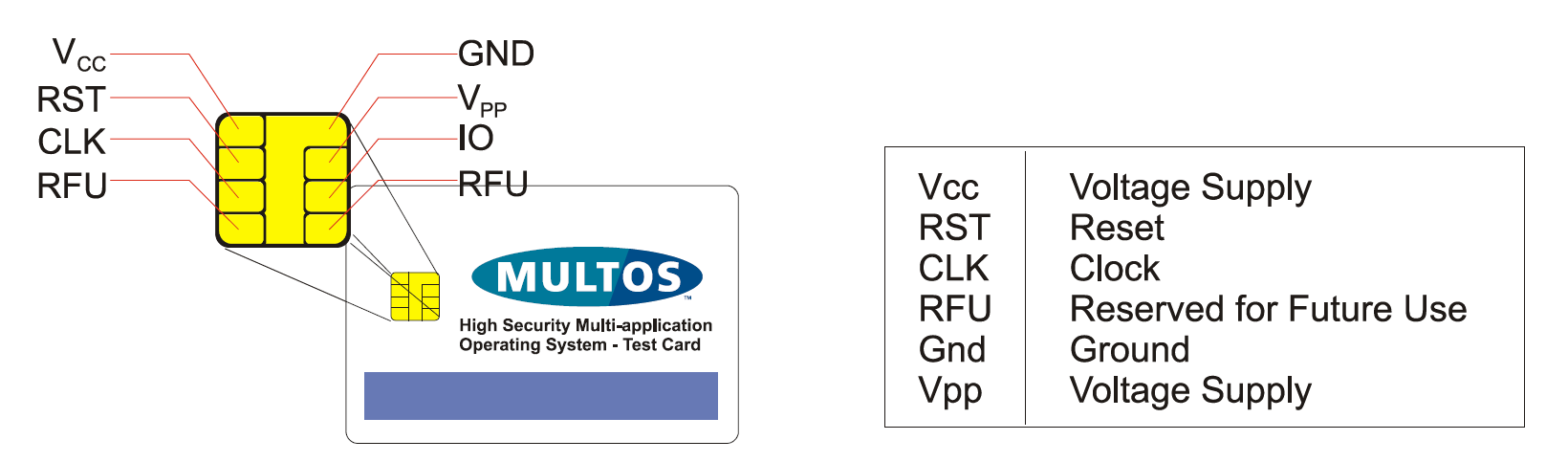
**2. General Operating Environment**

Smart cards are not stand-alone modules. A card on its own is simply a piece of plastic with a silicon chip embedded in it. In order for the chip to function it needs a power source. Furthermore, an application on a chip does not run on its own. It waits for data to be sent to it so that it can then carry out its function.

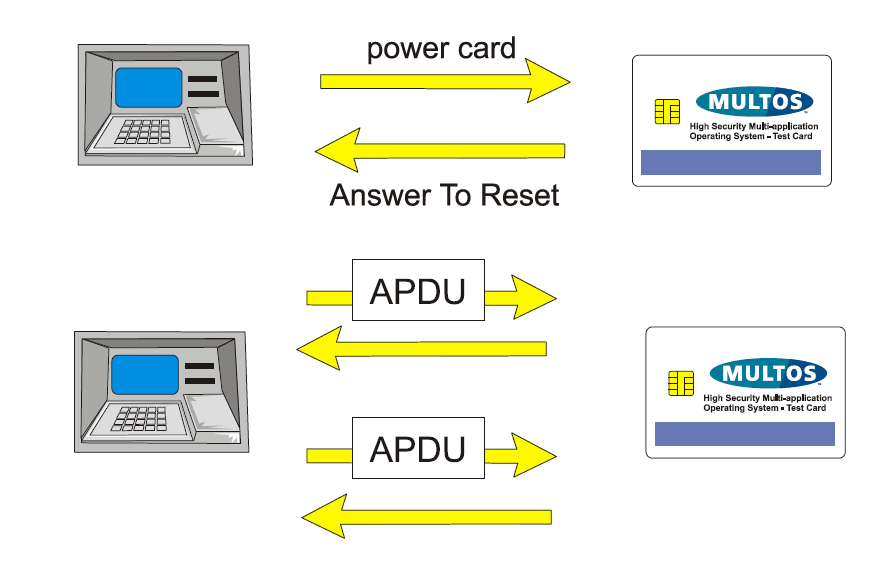
* **Card Applications:** is one that is able to receive and process commands. The command processing can be simple or complex. The application should also be able to return data in response to a command.
* **Interface Devices (IFD):** In order to use a chip card some sort of reader is required. Readers can come in many shapes and sizes from hand held devices to stand alone kiosks. An IFD is more than a chip card reader. It must also be able to supply power the chip, transmit commands and handle the chip application’s responses. It is the IFD initiates and directs the terminal –application interaction. This is referred to as the command – response dialogue, the basis of application development.

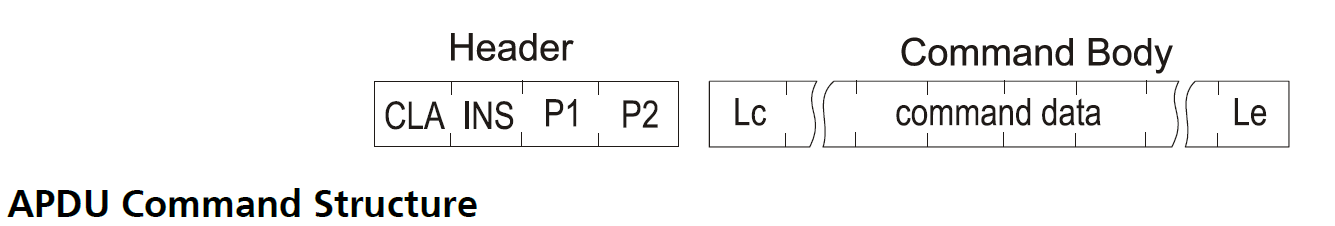
****

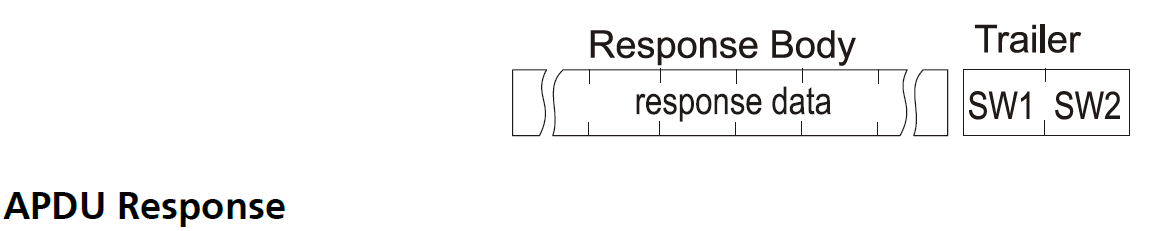
**3. Smart Card Communications**



* The clock is used to regulate the speed of operation. The reset contact is present to allow an IFD to start or reinitiate communications.
* When a smart card is powered on, it replies to the IFD by sending a string of bytes referred to as the Answer-To-Reset or ATR. The ATR is used by the smart card to inform the IFD of its electrical and communication capabilities and to help the IFD to establish a commonly understood protocol for further communications.
* Once communication has been established based on the information contained in the ATR it is then possible to send commands to the card. The command structure is defined in [7816-5] and is called an Application Protocol Data Unit. They are more commonly referred to as APDU.

****

****

****

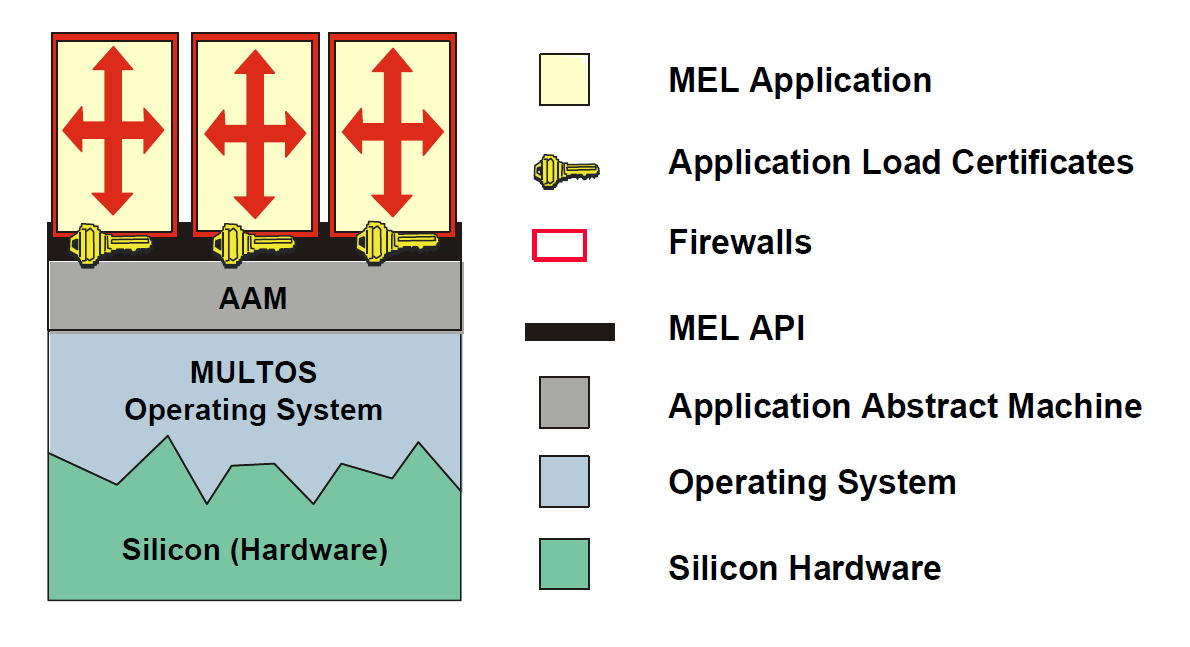
**4. MULTOS Applications as Dedicated Files**

* ISO File Structure shows that a smart card can be considered to consist of a series of files. Now, the definition of an Elementary File (EF) clearly states that it is meant to hold data. Dedicated Files (DF), on the other hand, can hold other DF and / or EF.
* MULTOS File Structure the operating system considers top level dedicated files as applications. What this means is that MULTOS treats the loading of an application as the creation of a new dedicated file. So, an application can be selected as described in the section MULTOS and Application Selection and application specific commands can then be sent to it.

**5. Application Selection and Application ID**

* A multi-application card will have more than one application per card. So, a method was required to indicate to which application a command is sent. The method is application selection, which is done by transmitting a SELECT FILE command as defined in [7816-4].
* The key bit of information sent as command data in the SELECT FILE command is the Application ID, also known as the AID. This ID with a length between 1 and 16 bytes inclusive is unique on the card and may be proprietary or ISO registered as per [7816-5]. Please note that both parameter bytes are used to indicate if FCI (File Control Information) should be returned.

**....MULTOS chip Architecture**



**The components are:**

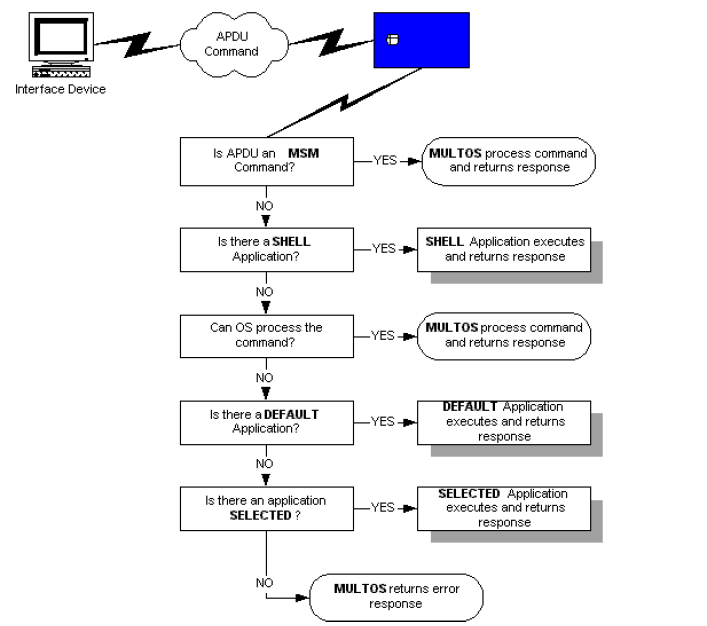
* Silicon Hardware: The underlying hardware is the physical platform that supports the OS functions. Those functions are written in native code, but are accessed via a fully specified virtual machine, which is the same no matter the hardware used.
* Operating System: The operating system provides the underlying communications, memory management and virtual machine. It also handles the loading and deleting of Applications, application selection and the handling of APDU commands and responses.
* Application Abstract Machine: The AAM provides a standard API consisting of a set of instructions and in-built functions, called primitives.
* MEL API: this provides support for MULTOS Assembly Language code. There will be some examples of this in the document. It is also possible to use C or Java to write MULTOS applications.
* Application Load Certificates: this has been included to indicate that a digitally signed certificate is required to load an application on to a MULTOS card. The certificate once used is discarded. For more information on loading and deleting information see [GLDA].
* Applications and Firewalls: see Figure 17: Application Space in MULTOS for a description.

**Modes of Operation: Standard, Shell, Default and Proprietary**

Applications on MULTOS cards need to be able to work in a variety of environments. There are different operational modes intended to facilitate this.

* **The standard mode:** corresponds to the general concept of multi-application smart cards. There are applications on the card and in order to use one it must be selected. Once that application is no longer required, another application can be selected and commands sent to it.
* The assumption made when using standard mode is that the existing external infrastructure is able to work with multi-application smart cards. In some cases, this assumption does not hold. In order to permit MULTOS applications to work in this type of environment it is possible to use **shell mode**. The shell application, then, serves as the sole interface between any IFD and any other applications on the card. This means that there is no need to select the shell application for it to be ready to process commands and also that it is not possible to directly select any other application. So, the shell application must be able not only to process commands destined for it, but also must be able to route commands to the relevant applications that are also on the chip.
* **Default mode:** is similar to shell mode in that the default application is immediately available to receive commands. However, it is still possible to use SELECT FILE as in standard mode. For example, a SIM phone application could be on chip along with, say, a payment application. In this case it would make sense for the phone application to be a default application because in that way it would be able to process incoming phone calls immediately. If, however, the payment application was required, a SELECT FILE command could be used and MULTOS, not the phone application, would handle it normally.
* **Proprietary mode**: (an optional feature from MULTOS 4.5.1) allows implementers to create other types of applications for specific purposes on a particular MULTOS product range.

**Command Routing:**



* Only the OS can process an MSM (MULTOS Security Manager) command. These commands include all of the commands used for loading an application, deleting an application or enabling a card.
* The third step “Can OS process the command?” is best illustrated by the SELECT FILE command. So, if a select command is sent the OS will check to see if there is an existing application with the ID given and, if so, will select the file indicated.
* The implication of the previous point is that a shell application will have to handle all command routing because it receives all incoming commands.