
**Exam Preparation Notes
Covering
Java Specification Requests
For
Sun Certified Mobile Application Developer
Sun Microsystems, Inc**

01 September, 2004

**Ahmed S. Habib Khan
(SCJP, SCWCD, OCP, MCP)
Ahmadkhan18@hotmail.com**

Following JSRs are covered:

[JSR-185: Java Technology for Wireless Industry v 1.0 \(JTWI 1.0\)](#)

[JSR-030: Connected Limited Device Configuration v 1.0 \(CLDC 1.0\)](#)

[JSR-139: Connected Limited Device Configuration v 1.1 \(CLDC 1.1 Only Changes since 1.0 \)](#)

[JSR-118: Mobile Information Device Profile v 2.0 \(MIDP 2.0\) –Not Completed](#)

[JSR-135: Multimedia Application Programming Interface \(MMAPI 1.1\)](#)

[JSR-120: Wireless Messaging API \(WMA 1.1\)](#)

Definitions

This document uses definitions based upon those specified in RFC 2119 (See <http://www.ietf.org>).

Term	Definition
MUST	The associated definition is an absolute requirement of this specification.
MUST NOT	The definition is an absolute prohibition of this specification.
SHOULD	Indicates a recommended practice. There may exist valid reasons in particular circumstances to ignore this recommendation, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	Indicates a non-recommended practice. There may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	Indicates that an item is truly optional.

JSR-185: Java Technology For Wireless Industry v 1.0 (JTWI 1.0)

Recommended Resources (In addition to the minimum requirements of MIDP 2.0):

- Screen size: 125x125 pixels.
 - Color depth: 4096 (12 bit).
 - Pixel aspect ratio: 1:1
 - Volatile memory (e.g. Java Heap): 256kb
 - Standard Application size (Jar file): 64k
 - Application Descriptor (JAD file): 5kb
 - RMS Data Size: 30kb (This should be the minimum amount of RMS that application can work with).
 - Minimum Application Thread Count: 10
 - Minimum clock resolution: no more than 40ms and
 - Record Store Minimum: at least 5 independent record stores.
 - Minimum numbers of timers: 5 simultaneously running timers. This requirement is independent of the *Minimum thread*.
- An JTWI compliant implementation must pass the TCKs for all the implemented JCP specifications.
 - The amount of memory specified for RMS-Data-Size is permanently reserved, so a lower value should be specified as nothing stops the MIDlet from allocating more memory later on.
 - A MIDlet suite may try to store more data than RMS-Data-Size; in this case the MIDlet suite is responsible for correct operation even when no space is available.
 - CLDC 1.0 is the minimum required J2ME configuration. JTWI must be implemented on top of CLDC 1.0 and pass the corresponding TCK.
 - A configuration of J2ME specifies a subset of java programming, JVM as well as the core libraries.
 - A JTWI compliant device the method `java.lang.System.currentTimeMillis()` must record an elapsed time in increment of no more than 40 milliseconds. At least 80% of the test attempts meet the elapsed time requirement to achieve the acceptable performance. A compliant device may support a smaller increment.
 - A compliant must allow use of custom time zones. No day light saving transition can be specified with custom time zones.
 - Only the time zone ID "GMT" is mandatory in CLDC 1.0/1.1.
 - Implementations must support the preferred MIME name as defined by IANA.

-
- Compliant implementations must support for character properties and case conversions for characters in “Basic Latin” and “Latin-1 Supplement” blocks of Unicode 3.0. Other blocks may be supported.
 - Compliant device must support Unicode characters 3.0, however since the full character tables can be excessively large, by default character properties and case conversions assume the presence of ISO Latin-1 range of characters only.
 - MIDP 2.0 or a subsequent compatible version must be implemented and the corresponding TCK must be passed. An implementation of MIDP 2.0 may implement any optional features of MIDP 2.0, but some optional features of MIDP 2.0 are mandated by JTWI.
 - A compliant device must provide support for HTTP 1.1 for all supported media types. HTTP 1.1 conformances must match the MIDP 2.0 specification.
 - A JTWI compliant device must support JPEG image format. The mandate is voided in the event that JPEG image format becomes encumbered with licensing requirements.
 - The only mandatory image format by MIDP 2.0 is PNG.
 - A compliant must allow an application to specify values for *firstTime*, *delay*, *period* parameters of `java.util.Timer.schedule` method with a distinguishable resolution of no more than 40 milliseconds.
 - A compliant device must support loading of PNG images with pixel color depth of 1,2,4,8,16 and 32 bits per pixel. For each of these color depth as well as for JFIF image format a compliant device must support images up to 32768 pixels.
 - Although devices are not mandated to have colored displays they must be able to process colored images.
 - A compliant device must provide a mechanism to select a phone number from the device phone book when the user is editing `TextField` or `TextBox` and the constraint type is `TextField.PHONENUMBER`. This requirement is void if phone book is not accessible
 - A compliant device must implement alarm-based push registry entries. If no other security mechanism is available The `PushRegistry` alarm must not be allowed without explicit user permission.
 - The system property “`microedition.jtwi.version`” must return 1.0 which is retrieved by `java.lang.System.getProperty()` method.

-
- WMA 1.1 or a subsequent compatible version must be implemented and a compliant device must pass the corresponding TCK. A device may optionally implement some of the optional features of WMA, but some of the optional features of WMA are mandated by JTWI.
 - GSM/WCDMA(UMTS) must support the GSM SMS service.
 - GSM/WCDMA(UTMS) phones must support MIDP 2.0 Push handling for the SMS protocol. If no other security mechanism is available, it must not be allowed without explicit user permission.
 - MMAPI may be implemented by a JTWI compliant device, if implemented version 1.1 is the minimum version.
 - A compliant device must implement MMAPI support for all media services that are exposed through java API. Implementation should expose support for these media services via these java APIs. Some of MMAPI features are made mandatory by JTWI specification, the remaining optional features remain optional.
 - HTTP 1.1 must be supported for media file downloads for all supported media formats. MMAPI does not mandate any protocols. It states that protocols must be handled in profiles. It leaves the supported protocols and media formats up to the implementation. It also does not make any requirements on which content types should work over which protocol.
 - Compliant device must implement MIDI feature set specified in MMAPI specification (JSR-135).
 - Compliant device must support MIDI file playback.
 - Support for "VolumeControl" must be implemented.
 - "MIDIControl" is not mandatory.
 - A compliant device that supports the video feature and video image capture must support JPEG encoding in Video snapshots.
 - Tone sequence file must be supported.
 - **The base MIDlet suite security framework is defined in MIDP 2.0, JTWI extends the base security framework in following areas:**
 - The required security model for GSM/UMTS compliant devices
 - Capabilities of MIDlets based on permission defined by MIDP 2.0 and other JCP specification.
 - The definition of user permission interaction mode
 - Guidelines on user prompts and notifications

-
- The JTWI specification defines the requirements for *Untrusted domain*. MIDlet suites that are not trusted belong to Untrusted domain. The integrity and origin of an untrusted MIDlet suite can not be reliably trusted.
 - A Protection domain is a way to differentiate MIDlets suites based on the potentially present digital signature in the MIDlet suite.
 - A MIDlet suite cannot belong to more than one security domain
 - MIDlets signed using the MIDP 2.0 signing mechanism must not be installed as untrusted.
 - A user must be informed whenever a new MIDlet suite is installed in the untrusted domain. The notification must indicate that the application does not come from a trusted source. The user must make an informed decision
 - **Function Groups:**
 - **Network/ Cost related groups:**
 - Net Access (e.g. GSM, GPRS)
 - Messaging (e.g. SMS, MMS)
 - Application Auto Invocation (e.g. push and time MIDlets)
 - Local connectivity (e.g. local comm. Ports, irDa and bluetooth).
 - **User privacy related groups**
 - Multimedia Recording (e.g. capture still images, record audio/video).
 - The function group not the individual permission should be presented to the user.
 - Each permission must occur only in one function group

Table 5

Group Name	Default value	Allowed setting
Net Access	Session	Session, Oneshot, No
Messaging	Oneshot	No
Application auto invocation	Session	Session, Oneshot, No
Local connectivity	Session	Blanket, Session, Oneshot, No
Multimedia recording	Oneshot	Session, Oneshot, No

- When a user grants permission to a function group, this then effectively grants access to all individual permissions user this function group.

-
- Implementation must guarantee that a `SecurityException` is thrown if the caller does not have the security permission.
 - A device cannot access the network with out user notification
 - Untrusted MIDlet suites may request permissions using the attributes defined in the JAD file or JAR manifest file
 - A device must have a set of function group for each untrusted MIDlet suite. The device should not present a way to set function group setting for several MIDlet suites at once.
 - Compliant implementation must be able to present the MIDlet suite name, version number and function group settings to the user for each installed MIDlet suite. An implementation may present additional security related information.
 - A MIDlet must obtain user approval to connect to the network in accordance with user permission settings of the policy.
 - A MIDlet should not be able to simulate security warnings to mislead the user.
 - The Implementation of “`HttpConnection`” and “`HttpsConnection`” must include a separate User-agent header with Product-Token “UNTRUSTED/1.0”
 - User-agent headers supplied by the applications must not be deleted.
 - The implementation of “`SocketConnection`” using TCP must throw a “`java.lang.SecurityException`” when an untrusted MIDlet suite attempts to connect to port 80 and 8080 (http) and 443(https).
 - The implementation of `SecureConnection` must throw a `SecurityException` if an untrusted MIDlet suite attempts to connect to port 443(https).
 - An Implementation of “`DatagramConnection`” must throw a `SecurityException` if an untrusted MIDlet suite attempts to send datagrams to any of port 9200-9203(WAP gateway).
 - The above requirement should be applied regardless of the API used to access the network.

JSR-030: Connected Limited Device Configuration v 1.0

- CLDC is intended to run on a variety of devices, which differ considerably, that is why CLDC Specification does not impose any specific hardware requirements other than memory requirements.
- **CLDC specification assumes that the JVM, libraries, profile libraries and the application to all fit within a total memory budget of 160-512 KB with following assumptions:**
 - 128kb of non-volatile memory for the Java virtual Machine and CLDC libraries
 - 32kb of volatile memory for java runtime and object memory
- **The host operating system must provide schedulable entity to run the Java virtual Machine.**
- **CLDC Scope:**
 - Java language and virtual machine features
 - Core java libraries (java.lang. *, java.util. *)
 - Input/ Output
 - Networking
 - Security
 - Internationalization
- **Out of CLDC Scope (CLDC Should not address these features):**
 - Application life cycle (Application launching, installation, deletion)
 - User interface functionality
 - Even handling
 - High-level application model (The interaction between the user and the application).
These features can be address by the profiles implemented on top of the CLDC
- **CLDC assumes that device has the following capabilities for managing java applications:**
 - Inspect existing java application stored on the device
 - Select and launch java application
 - Delete existing java applications (If applicable)
- **Concerning security CLDC focuses on two areas:**
 - **Low-level virtual machine security**
 - Low-level virtual machine security means that the java application executed by the virtual machine must not be able harm the device in which it is running. This is guaranteed by the Java classfile verifier. A JVM supporting CLDC must reject invalid classfiles

-
- **Application level security**
 - There are still areas, which will go unnoticed by the verifier, for instance access to external resources such as the file system, printers, and infrared devices. The J2SE security model is too huge to fit in a CLDC device.
 - **So CLDC provides a Sandbox Model:**
 - A JVM supporting CLDC provides a simple “Sandbox” security model. Sandbox means that a java application must run in a closed environment in which the application can only access those APIs that have been defined by the configuration, profiles and other licensee open classes supported by the device. More specifically the Sandbox model means:
 - Java class files have been properly verified and are guaranteed to be valid Java application (Via classfile verification).
 - Only limited, predefined set of java APIs is available to the application developer, as defined by CLDC, profiles and licensee open classes.
 - The downloading and management of Java application on the device takes place at the native code level inside the JVM, and no user defined class loaders are provided, in order to prevent the programmer from overriding standard class loading mechanism of the virtual machine
 - The set of native functions accessible to the application is closed, meaning that the application programmer cannot download any new libraries containing native functionality, or access any native functions that are not a part of the java libraries provided by the CLDC, profiles and licensee open classes.
 - A CLDC implementation must not allow the user to override the classes in protected packages (e.g. java. *, java.microedition. *) or other profiles or system specific classes.
 - Application programmer must not be allowed to change the class look up order in anyway
 - A CLDC implementation can run a single java application or multiple java applications
 - All end-to-end security solutions are assumed to be implementation-dependent.
 - **Main language difference between JLS and CLDC specification is:**
 - No floating point support
 - No finalization (Object. finalize () is not included)
 - **Error handling Limitations**
 - Limited set of Exception classes
 - Mandating the presence of all error classes would impose a significant overhead.
 - An implementation shall support a limited set or error classes and handle the error that is appropriate way to the device.

-
- No Reflection API (Class.forName () is available)
 - No Java Native Interface (JNI)
 - No Thread Groups and daemon threads
 - No Weak references
- Classfile verification: Like a standard J2SE virtual machine, a JVM supporting CLDC must be able to reject invalid classfiles.
 - The KVM's verifier performs only a linear scan of the byte code, without the need of a costly iterative algorithm.
 - **The new KVM verifier performs in two steps:**
 - Pre verification (Off the device)
 - In device verification
 - The new verifier requires the java class files to contain a new attribute. The new verifier has a pre-verification tool, which inserts this attribute to the class files. A transformed class file is still a valid J2SE class file
 - A missing corrupted or invalid attribute causes the class file to be rejected by the runtime verifier.
 - A JVM supporting CLDC must support Java archive files (JAR).
 - Whenever a Java application intended for CLDC is represented publicly, the compressed jar file representation format must be used.
 - Classfile lookup order is implementation dependent but the implementation must make sure that the system class cannot be overridden and the class lookup order must not be altered in any way.
 - The rule for J2Me configuration mandates that each class that has the same class name and same package as J2SE class must be identical or a subset of the corresponding J2SE class.

-
- **The following classes are included from J2SE into J2ME:**
 - **System Classes**
 - java.lang.Object
 - java.lang.System
 - java.lang.Class
 - java.lang.Runtime
 - java.lang.String
 - java.lang.StringBuffer
 - java.lang.Thread
 - java.lang.Runnable (Interface)
 - java.lang.Throwable
 - **Data type classes**
 - java.lang.Long
 - java.lang.Byte
 - java.lang.Short
 - java.lang.Integer
 - java.lang.Boolean
 - java.lang.Character
 - **Collection classes**
 - java.util.Stack
 - java.util.Vector
 - java.util.Hashtable
 - java.util.Enumeration (Interface)
 - **Input/ Output classes**
 - java.io.InputStream
 - java.io.OutputStream
 - java.io.ByteArrayInputStream
 - java.io.ByteArrayOutputStream
 - java.io.DataInputStream
 - java.io.DataOutputStream
 - java.io.InputStreamReader (Also used for internationalization)
 - java.io.OutputStreamWriter (Also used for internationalization)
 - java.io.DataInput
 - java.io.DataOutput
 - java.io.Reader
 - java.io.Writer
 - java.io.PrintStream

-
- **Calendar And Time Classes**
 - By default only one time zone is supported, additional time zones maybe provided by the actual implementation.
 - java.util.Calendar
 - java.util.TimeZone
 - java.util.Date
 - **Additional Utility Classes**
 - java.util.Random
 - java.lang.Math
 - **Property support**
 - CLDC does not implement java.util.Properties, instead system properties are access via System.getProperties (String key).
 - Property “microedition.encoding” describes the default-encoding name.

Key	Explanation	Default Value
microedition.platform	Name of the host platform	null
microedition.profiles	Name of the supported profiles	null
microedition.encoding	Default character encoding	“ISO8859_1”
microedition.configuration	Name and version of supported configuration	“CLDC-1.0”

- **Generic Connect Framework**
All connections are created using a single static method in a system class called Connector.
If successful, this method will return an object that implement one of the generic connection interfaces. This method takes a single string in general form:
Connector.open (“<protocol>:<address>;<parameters>”)
- **CLDC it self does not include any protocol implementation**

HTTP

Connector.open("http://www.foo.com")

Sockets

Connector.open("socket://192.2.1.2:9000")

Communication Port

Connector.open("comm:0;baudrate=9600")

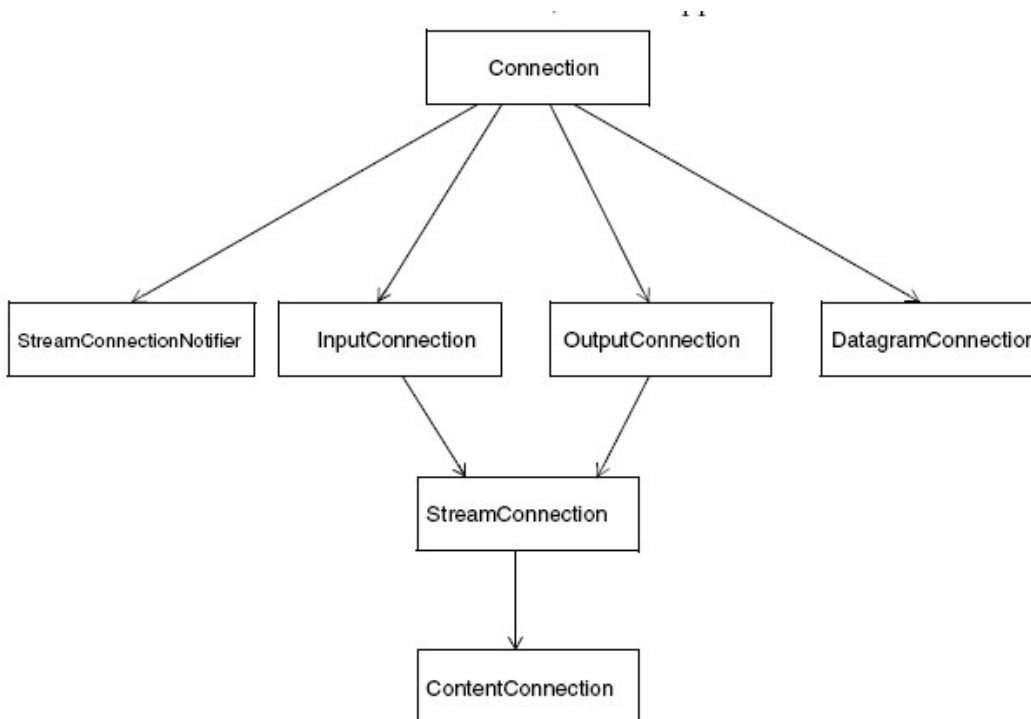
Datagrams

Connector.open("datagram://192.144.111.222")

Files

Connector.open("file:/foo.dat")

- **Connection interface hierarchy**



JSR-139: Connected Limited Device Configuration v 1.1 (CLDC 1.1) (Only Summary of Changes since 1.0)

- The list below summarizes the main differences between CLDC specification version 1.0 (JSR 030) and version 1.1 (JSR –139):
 - Floating point support has been added
 - Float and Double classes are added
 - Weak reference support (small subset of J2SE weak reference classes) has been added.
 - Classes Date, Calendar and TimeZone have been modified to be more J2SE compliant.
 - Error handling requirements have been clarified and one new error class NoClassDefFoundError has been added.
 - In CLDC 1.1 Thread objects have name like in J2SE. The method Thread.getName() has been added plus the Thread class has few more constructors inherited from J2SE.
 - Various minor library changes like the following has been added:
 - Boolean.TRUE and Boolean.FALSE
 - Date.toString()
 - Random.nextInt(int n)
 - String.intern()
 - String.equalsIgnoreCase()
 - Thread.interrupt()
- Minimum memory budget has been raised from 160(128kb non-volatile/32kb volatile) to 192(160kb non-volatile/32volatile) kilobytes, mainly because of the added floating-point functionality.

JSR-118: Mobile Information Device Profile v 2.0 (MIDP 2.0)

Not Complete

- MIDP 2.0 is fully backward compatible with MIDP 1.0
- MIDP specifies no additional low level security API other than those provided by CLDC
- MIDP focus on enabling application programming rather than system programming
- **MIDP Minimum required hardware characteristics:**
 - Display
 - 96x54 pixel
 - Display depth: 1-bit
 - Pixel Shape(aspect ration): 1:1
 - Input: One or more of the following user-input mechanisms: one-handed keyboard, two-handed keyboard, or touch screen.
 - Memory
 - 256kb of non-volatile memory for MIDP implementation beyond what is required by CLDC.
 - 128kb of volatile memory for java runtime
 - 8 kb of non-volatile storage for application created persistent data
 - Networking: Two-way wireless possibly intermittent, with limited bandwidth
 - Sound: The ability to play tones, either via dedicated hardware or software algorithm
- **Minimum software requirements:**
 - A minimal kernel to manage the underlying hardware. This kernel must provide at least one schedulable entity to run the virtual machine
 - A mechanism to read/write to non-volatile memory to support RMS (Record Management System) for persistent storage.
 - Read and write access to device wireless network to support the Networking APIs.
 - A mechanism to provide time base for use in time-stamping the records written to persistent storage and to provide the basis for timer APIs.
 - A minimal capability to write to a bit-mapped graphic display.
 - A mechanism to capture the user input from one or more of the three input mechanisms.

-
- A mechanism for managing the applications life cycle of the device.
 - **Specification requirements:**
 - Requirements specified here take precedence and replace the conflicting requirements listed elsewhere.
 - **Compliant MIDP 2.0 implementations:**
 - MUST support MIDP 1.0 and MIDP 2.0 MIDlets and MIDlet suites.
 - MUST include all packages, classes and interfaces described in this specification.
 - MUST implement the OTA user initiated provisioning specification
 - MAY incorporate zero or more supported protocols for push.

 - MUST give the user a visual indication of network usage generated when using mechanisms indicated in this specification.

 - MAY provide support for accessing any available serial ports on their device through the CommConnection interface.

 - Must provide support for HTTP 1.1 servers and services either directly or by using gateway such as provided by WAP or i-mode.

 - Must provide support for secure HTTP connections either directly or via a gateway.

 - Should provide support for datagrams

 - Should provide support of server socket stream connection.

 - Should provide support for socket stream connection.

 - Should provide support for Secure Socket stream connection.

 - Must support PNG image transparency.

 - May include support for additional image formats.

 - Must support Tone generation in media package.

-
- Must support 8 bit, 8 KHz, mono liner PCM wave format IF any sampled sound support is provided.
 - May include support for additional sampled sound format.
 - Must support scalable polyphony MIDI(SP-MIDI) SP-MIDI device 5-24 Note Profile IF any synthetic sound support is provided.
 - May include support for additional MIDI formats.
 - Must implement the mechanism needed to support “Untrusted MIDlets Suites”.
 - Must implement “Trusted MIDlet Suite Security” unless the device security policy does not permit or support trusted applications
 - Must implement “Trusted MIDlet Suites X.509 PKI” to recognize signed MIDlet suites as trusted unless PKI is not used by the device for signing applications.
 - Must implement “MIDP x.509 Certificate Profile” for certificate handling of HTTPS and SecureConnection.
 - Must enforce the same security requirements for I/O access from media API as from Generic Connection Framework.
 - Must implement at least the UTF-8 character encoding for APIs that allow the application to define character encoding.
 - May support other character encodings.
 - Should not allow copies to be made of any MIDlet suite unless the device implements a copy protection mechanism.
- **Over The Air Initiated Provisioning Specification:**
 - **Functional Requirements**
 - Browsing, locating MIDlet suites over the network.
 - Transferring a MIDlet suite and its associated Application Descriptor to the device from a server using HTTP 1.1 or a session protocol that implements HTTP 1.1 functionality.

-
- Responding to 401(*Unauthorized*) Or 407 (*Proxy Authentication Required*) response to an HTTP request by asking the user for username and password and resending the HTTP request with the credentials supplied.
 - At least the Basic Authentication Scheme.
 - Installing MIDlet suites on the device
 - Invoking MIDlets
 - Allowing users to delete MIDlet suites. Individual MIDlets cannot be deleted.
 -

JSR-135: Multimedia Application Programming Interface (MMAPI 1.1)

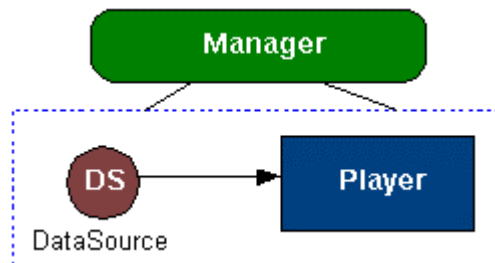
- **Basic concepts: Protocol and content Handling**

Basically Multimedia processing can be broken into two parts:

- **Handling the data delivery protocol**
 - Protocol handling refers to reading data from a source such as file, capture device into a media processing system. The DataSource Object handles this.
- **Handling the data content**
 - Content handling usually requires processing the media data and rendering the media to output device such as and audio or video display. This is handled by Player Object.



- DataSource encapsulates protocol handling. It hides the details of how the data is read from its source, whether the data is coming from a file, streaming server. DataSource provides a set of methods to allow a Player to read data from it for processing.
- A factory mechanism the Manager, creates Players from DataSources. For convenience, Manager also provides methods to create Players from locators and InputStreams.



- **The createPlayer method is the top-level entry point to the API:**
 Player Manager.createPlayer(String url),
 The url fully specifies the protocol and the content of the data:
 <protocol>:<content location>
- The Manager parses the URL and creates a DataSource to handle the specified data delivery protocol. The DataSource derives the content type from the data. The Manager then takes this content type and creates a Player to handle the presentation of the data. The resulting Player is returned for use by the application.
- **MMAPI has some properties that can be queried by System.getProperty(String key):**

Key	Description
supports.mixing	Query for whether audio mixing is supported. The String returned is either "true" or "false". The following conditions are true: <ul style="list-style-type: none"> ▪ At least two tones can be played with Manager.playTone simultaneously. ▪ Manager.playTone can be used at the same time when at least one Player is playing back audio. ▪ At least two players can be used to playback audio simultaneously.
supports.audio.capture	Query for whether audio capture is supported. The String returned is either true of false.
supports.video.capture	Query for whether video capture is supported. The string returned is either true of false.
supports.recording	Query for whether recording is supported. String returned is either true of false
audio.encodings	The string returned specifies the supported capture audio format. Each format is at least separated by one space.
vedio.encodings	The string returned specifies the supported capture video format. The formats are at least separated by at least

	one space.
video.snapshot.encodings	Supported video snapshot format for the “getSnapshot()” method in VideoControl

- Manager provides getSupportedContentTypes() and getSupportedProtocols() to query for supported content types and protocols.
- MMAPI does not mandate any protocol or media type. That is left to the profiles, but the implementation must guarantee to support at least one media type and protocol.
- MMAPI optional features are organized as Controls. A Player can be queried for supported controls with getControls(String controlName), or getControl().
- MMAPI mandates support for some features for some media types.

Feature Set

Feature Set	Implementation Requirements
Sample Audio	<ul style="list-style-type: none"> • Should implement AudioControl and StopTimeControl
MIDI	<ul style="list-style-type: none"> • Should implement AudiControl, MIDIControl, TempoControl, PitchControl, StopTimeControl
Tone Sequence(Player for TONE_DEVICE_LOCATOR)	<ul style="list-style-type: none"> • Must implement ToneControl • Should implement VolumeControl, StopTimeControl
Interactive MIDI(Player for MIDI_DEVICE_LOCATOR)	<ul style="list-style-type: none"> • Must implement MIDIControl
Vedio	<ul style="list-style-type: none"> • Must implement VideoControl • Should implement FramePositioningControl, StopTimeControl, VolumeControl (If audio is also available).

- The following controls does not belong to any above feature set and implementations may implement them when applicable: GUIControl, MetaDataControl, RateControl and RecordControl.
- **A player has five states:**
 - UNREALIZED
 - REALIZED
 - PREFETCHED
 - STARTED

-
- CLOSED

 - **The implementation must guarantee that these methods succeed under normal runtime conditions.**
 - **MMAPI does not define any security mechanism, an implementation must guarantee:**
 - An exception is thrown when the caller does not have appropriate security permissions.
 - The method can be used when appropriate security permissions are granted.

 - **The MIDP 2.0 subset differs from general multimedia API in following ways:**
 - Its audio only. It excludes all controls specific to video or graphics
 - It does not support synchronizing simultaneous playback of multiple Players using a common time base.
 - It does not provide custom protocols at the application level. The javax.microedition.media.protocol package (DataSource) is excluded
 - A simplified version of Manager is used
 - The MIDP 2.0 subset is fully upward compatible with MMAPI

JSR-120: Wireless Messaging API (WMA 1.1)

- The interfaces for the messaging API are defined in the `javax.wireless.messaging` package.
- The base interface implemented by all messages is “Message”, it provides methods for address and timestamps.
- `TextMessage` and `BinaryMessage` are sub interfaces of `Message` and handle text and binary payload respectively.
- If an application specifies a full destination address that defines a recipient to the Connector, it gets a `MessageConnection` that works in “Client” mode. This kind of connection can only be used to send messages.
- If an application specifies a connection string that only includes an identifier that specifies the messages intended to be received by the application (e.g. port number on local host), It will get a `MessageConnection` that works in “Server” mode and can both send and receive messages.
- The format of the URL string that identifies the address is specific to the messaging protocol used.
- For sending message the `MessageConnection` object provides factory methods for creating `Message` objects.
- In addition to a synchronized blocking `receive()` method, the `MessageConnection` also supports an event listener-based receive mechanism
- The methods for sending and receiving messages can throw `SecurityException` if the application does not have the appropriate permissions.

-
- The `MessageConnection` does not support Stream operations. A call to either `Connector.openInputStream()` or `Connector.openOutputStream()` will throw an `IllegalArgumentExpection`
 - To send and receive messages using this API the application must be granted permissions, for which the mechanism is implementation dependent.
 - An application must not assume that successfully sending one message implies that they have the permissions to send all kind of messages to all kind of addresses.
 - The `setAddress` method of `Message` interface does not validate or parse the address.
 - An `IOException` is thrown if an application tries to call any method of `MessageConnection` except `close()`.
 - If the requested end point is reserved while trying to open a server mode `MessageConnection` an `IOException` will be throw.
 - An application can have several `MessageConnections` open at the same time. They can be both in client and server mode.
 - The values for `MessageConnection.TEXT_MESSAGE` and `MessageConnection.BINARY_MESSAGE` are “text” and “binary” respectively.
 - The `MessageConnection.numOfSegment(Message msg)` does not send the message. It will only calculate the number of protocol segments needed for sending this message.
 - The `MessageConnection.receive()` method is blocking. This method will block until either a message for this connection is received or the connection is closed.
 - The `MessageConnection.send(Message msg)` will throw an `IllegalArgumentExpection` if the message exceeds the maximum length for given messaging protocol.
 - There can be at most one register `MessageListener` with a `MessageConnection` at a given point in time. Setting a new listener will de-register any previously set listener. Passing null to `MessageConnection.setMessageListener()` will de-register any current listener.
 - If multiple messages arrive very closely together in time, the implementation has the option of calling the `MessageListener` from multiple threads in parallel

-
- The implementation of `MessageListener.notifyIncomingMessage()` must not perform extensive operations and must return quickly. The application should not receive and handle the message during this method call
 - The implementation message must support the concatenation of SMS protocol messages and pass the fully reassembled payload to the application via this API.
 - Implementation if this API must support at least 3 SMS protocol messages to be received and concatenated together. Similarly for sending messages which can be sent up to 3 SMS messages must be supported.
 - When a port is present in address, the TP-User-Data of the SMS MUST contain a User-Data-Header with the application port addressing scheme information element.
 - When the recipient address does not contain a port, the TP-User-Data must not contain the application port-addressing header. Java applications cannot receive this kind of messages.
 - When a message identifying a port number is sent from a server type `MessageConnection`, the originating port number in the message is set to the port number of `MessageConnection`. This allows the recipient to send a response to the message that will be received by the `MessageConnection`. However when a client type `MessageConnection` is used for sending a message with a port number, the originating port number is set to a implementation-specific value and possibly messages received to this port number are not delivered to the `MessageConnection`.
 - The implementation must guarantee that the messages are delivered to the application in the same order as they were received by the implementation of the recipient terminal.
 - The SMSC address must be made available using `System.getProperty` with the property name `"wireless.messaging.sms.smsc"`.
 - When an application uses `BinaryMessage`, the TP-Data-Coding-Scheme in the SMS must indicate 8 bit data
 - When an application uses `TextMessage`, the TP-Data-Coding-Scheme the GSM default 7-bit alphabet or UCS-2.
 - 7 bit alphabet must be used for encoding if the string that is given by the application contains characters that are present in GSM 7-bit alphabet

- If at least one character in the supplied string by the application is not available in GSM 7-bit alphabet, then UCS-2 must be used.

- **For security reasons Java applications are not allow to send sms to following ports. Implementation must throw a SecurityException if an application try to send message to any of the following ports:**

Table A-4: Port Numbers Restricted to SMS Messages

Port number	Description
2805	WAP WTA secure connection-less session service
2923	WAP WTA secure session service
2948	WAP Push connectionless session service (client side)
2949	WAP Push secure connectionless session service (client side)
5502	Service Card reader
5503	Internet access configuration reader
5508	Dynamic Menu Control Protocol
5511	Message Access Protocol
5512	Simple Email Notification
9200	WAP connectionless session service
9201	WAP session service
9202	WAP secure connectionless session service
9203	WAP secure session service
9207	WAP vCal Secure
49996	SyncML OTA configuration
49999	WAP OTA configuration

- For CBS messages, the Message.getTimeStamp() method must always return null .
- For a CBS message calling the MessageConnection.send method must throw IOException, because CBS message can only be received.
- **The following permissions are required for opening message connection:**

Permission	Protocol
javax.microedition.io.Connector. sms	sms
javax.microedition.io.Connector. cbs	cbs

-
- Permissions for Send and Receive operations:

Permission	Protocol
javax.wireless.messaging.sms.send	sms
javax.wireless.messaging.sms. receive	sms
javax.wireless.messaging.cbs. receive	cbs