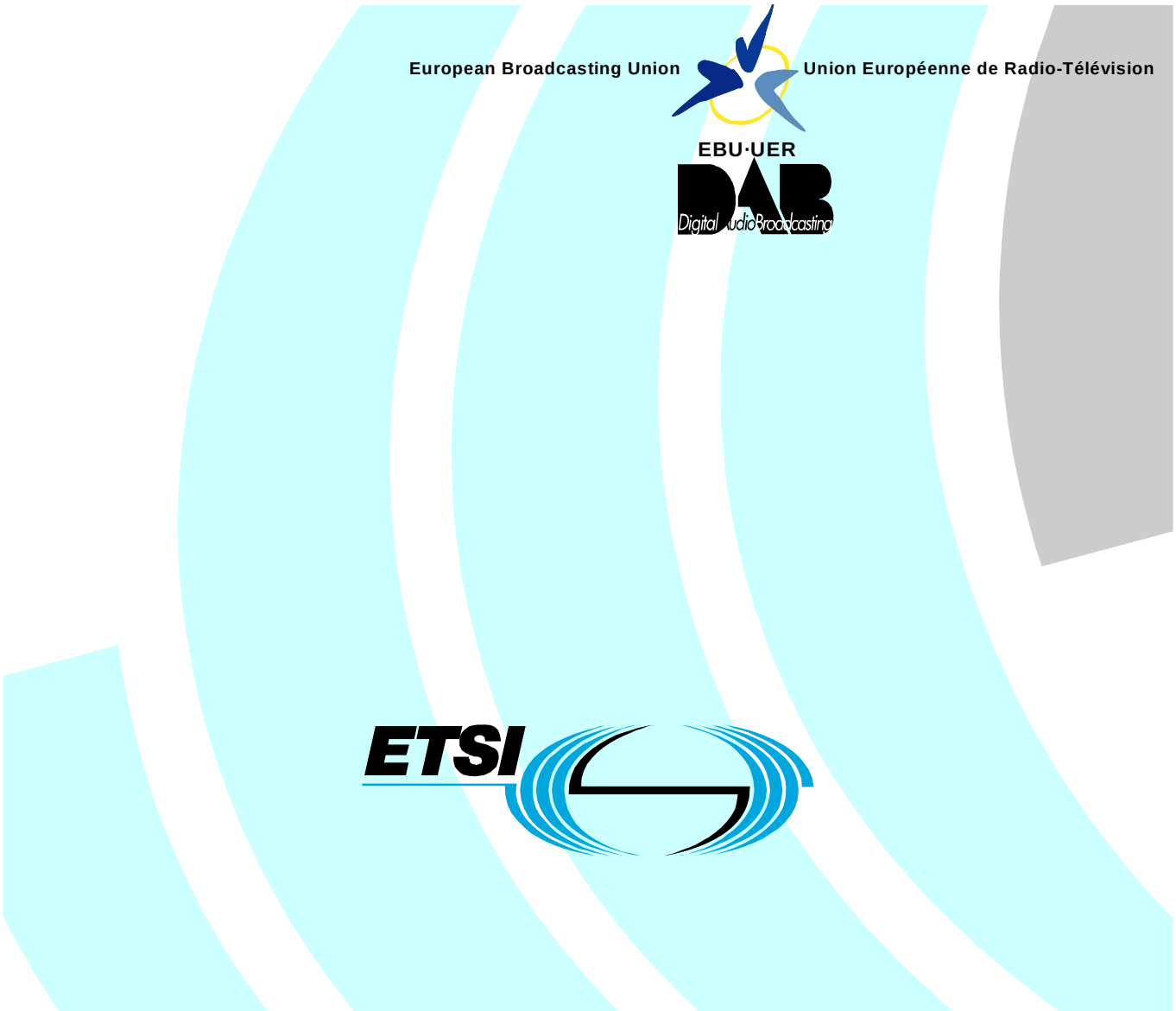
ETSI TS 102 635-1 V1.1.1 (2009-08)

Technical Specification

Digital Audio Broadcasting (DAB); Middleware; Part 1: System aspects



Reference
DTS/JTC-DAB-54-1

Keywords
Broadcasting, DAB, digital

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE 1: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

The Eureka Project 147 was established in 1987, with funding from the European Commission, to develop a system for the broadcasting of audio and data to fixed, portable or mobile receivers. Their work resulted in the publication of European Standard, EN 300 401 [23], for DAB (see note 2) which now has worldwide acceptance. The members of the Eureka Project 147 are drawn from broadcasting organizations and telecommunication providers together with companies from the professional and consumer electronics industry.

NOTE 2: DAB is a registered trademark owned by one of the Eureka Project 147 partners.

1 Scope

The present document establishes a standard for a platform-independent environment, where executable applications can be signalled and transferred to a receiver via a broadcasting network and executed on the receiver. It does not suppose the exclusive use of a specific broadcast network but defines the commonly-required specifications among diverse broadcast networks. It includes the definitions of basic data formats, protocols to deliver data, to signal downloadable applications and to download them, ways to denote resources on broadcast networks, and detailed interfaces among receiver platform, broadcast and communication networks, and the applications.

In order to apply the present document to a target broadcast network, it is required to map abstract interfaces to concrete entities of the network and to add additional definitions specific to the network.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause-were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] JSR 118: "Mobile Information Device Profile 2.0".

NOTE: See at (http://www.jcp.org/en/jsr/detail?id=118).

[2] JSR 217: "Personal Basis Profile 1.1".

NOTE: See at (http://www.jcp.org/en/jsr/detail?id=217).

[3] ISO/IEC 10918-1: 1994 "Information Technology –Digital compression and coding of continuous-tone still images –Requirements and Guidelines".

NOTE: See at (http://www.w3.org/Graphics/JPEG/itu-t81.pdf).

[4] JPEG File Interchange Format, Eric Hamilton, C-Cube Microsystems.

NOTE: See at (http://www.w3.org/Graphics/JPEG/jfif3.pdf).

[5] PNG (Portable Network Graphics) Specification, Version 1.0. W3C Recommendation, October 1, 1996.

NOTE: See at (http://www.w3.org/TR/REC-png.html).

[6] IETF RFC 793 (TCP): "Transmission Control Protocol", J. Postel.

[7] IETF RFC 768 (UDP): "User Datagram Protocol", J. Postel.

[8] IETF RFC 2616: "IETF Hypertext Transfer Protocol -- HTTP/1.1".

[9] IETF RFC 1034: "Domain Names - Concepts and facilities".

[10] IETF RFC 1035: "Domain Names - Implementation and specification".

[11] IETF RFC 1982: "Serial Number Arithmetic".

[12] IETF RFC 2181: "Clarifications to the DNS Specification".

[13] IETF RFC 3280: "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[14] IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".

[15] IETF RFC 3066: "Tags for the Identification of Languages".

[16] JSR 135: "Mobile Media API 1.1".

NOTE: See at (http://www.jcp.org/en/jsr/detail?id=135).

[17] JSR 75: "PDA Optional Packages for the Java ME Platform".

NOTE: See at (http://www.jcp.org/en/jsr/detail?id=75).

[18] ITU-T Recommendation X.509: "Information technology - Open Systems Interconnection - The Directory: Authentication framework".

[19] T. Porter and T. Duff, "Compositing Digital Images", SIGGRAPH 84, 253-259.

[20] Info-ZIP Application Note 970311.

NOTE: See at (ftp://ftp.uu.net/pub/archiving/zip/doc/appnote-970311-iz.zip).

[21] ISO 10646-1: "Information technology - Universal multiple-octet coded character set (UCS), part 1: Architecture and Basic Multilingual Plane".

[22] JSR 172: "Location API for J2ME".

NOTE: See at (http://www.jcp.org/en/jsr/detail?id=172).

[23] ETSI EN 300 401: "Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers".

[24] ISO/IEC 13818-1: "Information technology -- Generic coding of moving pictures and associated audio information: Systems."

[25] ISO 4217: "currency names and code elements".

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Not applicable.

3 Definitions, abbreviations and conventions

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

device driver: system software responsible for basic operation of hardware units

event: group of one or more media with specified start and end times

EXAMPLE: An event can be a soccer half time, a news flash, and so on.

platform standard: standard that is defined by the present document, and designates media-specifics, where media means either terrestrial or satellite DMB

program: group of one or more events being transmitted under a single broadcaster's control

EXAMPLE: A program can be news or entertainment.

service: series of programs being transmitted under a single broadcaster's control

service binding: binding of applications with services

NOTE: An application bound to a service is executed automatically upon user's selection of the service. If the user stops the service, the application is also destroyed.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
BNF	Backus-Naur Form
CA	Certificate Authority or Conditional Access
CAS	Conditional Access System
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
DMB	Digital Multimedia Broadcasting
DNS	Domaine Name Service
DSMCC	Digital Storage Media Command and Control
EPG	Electronic Programme Guide
HTTP	Hyper Text Transfer Protocol
JAR	Java ARchive
JVM	Java Virtual Machine
MATE	Multimedia Application Terminal Environment
MOT	Multimedia Object Transfer
PIN	Personal Identification Number
SI	Service Information
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

3.3 Conventions

3.3.1 Syntax of binary messages

The symbols, the abbreviations, and the methods for the description of syntaxes of binary messages in the present document shall follow those defined in sections 5.2 and 5.3 of ISO/IEC 13818-1 [24].

3.3.2 BNF

Unless otherwise specified, the BNF notation in the present document shall follow the definitions of section 1.3 of RFC 2616 [8].

4 Introduction

MATE provides a definition of a platform-independent environment, where executable applications can be signalled, transferred to receivers, and executed. The present document specifies abstract models for external entities which need to be further defined by the MATE external environment specifications.

5 System architecture

5.1 Introduction

The present document defines standard interfaces between external environment shown in figure 1 and the receiver implementing the MATE specification. There may be additional external entities other than those specified in figure 1, but within the scope of the present document, they are irrelevant and never mentioned.

In defining such standard interfaces, the present document takes an approach of specifying abstract models for external entities such as broadcast network to broaden the range of external environments to which the present document is applicable. Thus, MATE is not a valid standard for any implementation by itself and the abstract models need to be concreted according to the external environments.

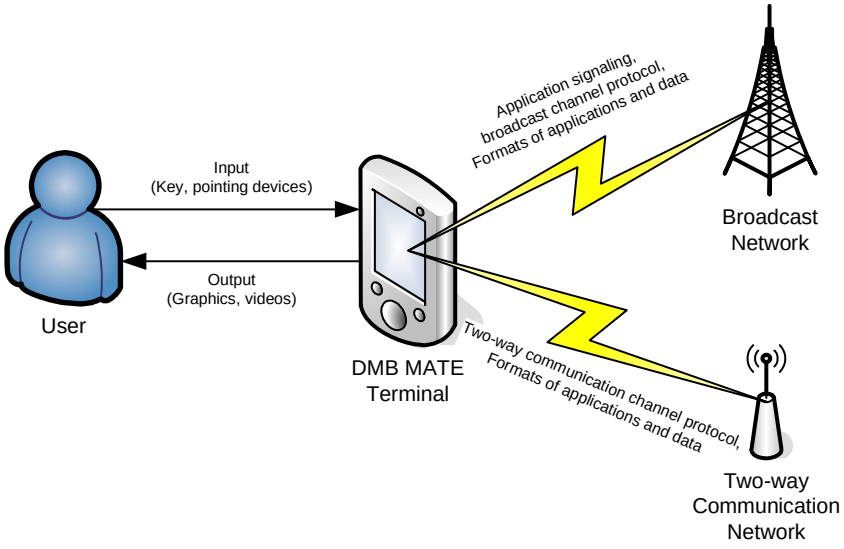


Figure 1: Environment external to MATE terminal

5.2 Receiver model

The basic purpose of MATE is to provide an environment for control of receivers and execution of applications transmitted to the receivers via broadcasting and communication networks. MATE receivers can be described as 3 layers as shown in figure 2. Since the basic model of a receiver presented here is an abstract model for reference in the present document, an actual receiver does not have to follow the structure of the model.

For instance, the present document can be implemented as a separate software module distinct from hardware and other software modules such as device drivers and operating systems with a porting layer to facilitate porting of the implementation across different receivers. On the other hand, some implementation may be interconnected with such software modules and hardware devices in an inseparable way.

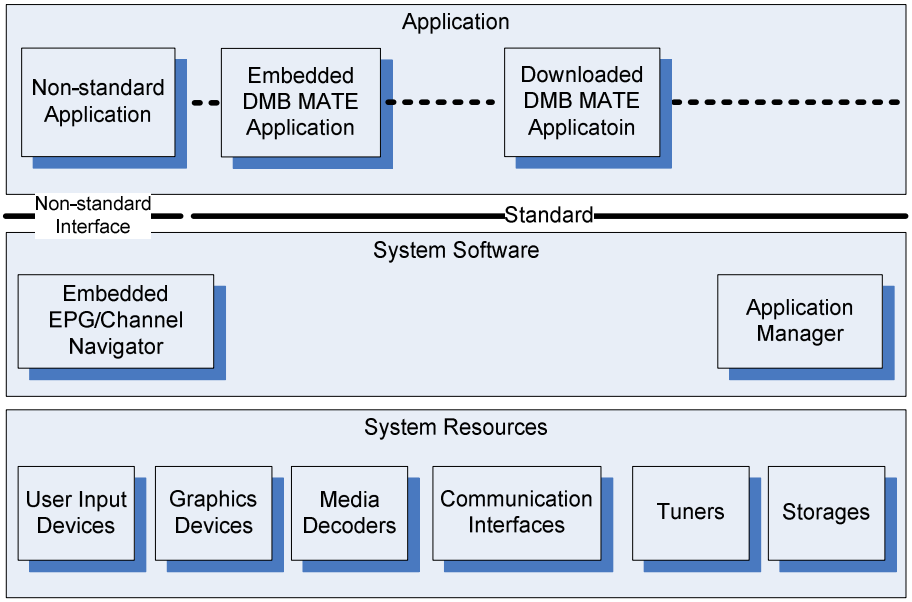


Figure 2: Basic receiver model

5.2.1 System resources

System resources refer to every single hardware and software resources composing a receiver. The same system resource can be implemented in hardware, software or hybrid manner depending on the receiver implementation. For instance, a media decoder may be implemented as either software or hardware device.

Not regarding concrete implementation form, the present document defines system operations and APIs based on the abstract model of resources shown in figure 2. Other than already mentioned resources, different kind of resources may be supported by actual receivers. The present document, however, does not define anything about such resources.

5.2.2 System software

System software makes applications executable without any modification in different receiver implementation by taking charge of management and control of system resources. It also provides applications high level functionalities using low level system resources.

Besides the components depicted in figure 2, there are more system software constituents although the present document does not deal with them.

Application manager: application manager receives, executes and manages applications. When there coexist types of embedded or downloaded applications that the present document does not specify, receiver implementation should be careful not to violate the present document.

Embedded EPG/Channel Navigator: EPG and channel navigator are the system software for basic TV watching. In addition to appropriate user interfaces, they provide a means to change the current channel, control media and so on depending on receivers. It can be implemented using APIs defined by the present document or in completely different ways. Even if they are not implemented with the standard APIs defined in the present document, the receiver implementation must be careful not to violate the present document.

5.3 Application

Application refers to software which operates using the APIs in the execution environment designated in the present document. The present document defines only how applications are downloaded via broadcasting and/or communication networks.

An application can be preinstalled in the receiver, or manually via interfaces (such as USB), other than broadcasting and communication networks, by users. The present document does not limit other possibilities, as long as the application conforms to it. However, such possibilities are not designated in details in the present document.

In addition, types of applications not designated by the present document can coexist, but interfaces between system software and those applications, and methods of embedding and downloading them are not standardized in the present document. But there must not be any violations to the present document caused by them.

6 Basic data formats

This clause-defines data formats that receivers complying with the present document should support. Some data formats designated in this clause-are mandatory meaning they must be supported in all the receivers in compliance with the present document. On the other hand, some formats are optional, but when they are supported in an implementation, the rules set in this clause-must be followed. Each platform standard may define additional data formats.

6.1 Image file formats

The following image formats should be supported by receivers complying with the present document.

6.1.1 JPEG

JPEG files must conform to ISO/IEC 10918-1 [3] and the file exchange format defined in JFIF [4], and the following decoding methods must be supported:

- Sequential DCT-based mode.
- Progressive DCT-based mode.

6.1.2 PNG

PNG files must conform to PNG 1.0 [5], and the additional requirements designated in the documentation of javax.microedition.lcdui.Image API in MIDP 2.0 [1].

6.2 Font file formats

The present document defines a method to download fonts but no format for font files is defined. In addition, font download is optional.

6.3 Video file formats

The present document does not designate a specific video file format. Instead, each platform standard is supposed to define some.

6.4 Audio file formats

The present document does not designate a specific audio file format. Instead, each platform standard is supposed to define some.

7 Transport protocol

This clause-defines broadcast and communication channel protocols which receivers conforming to the present document should support. The present document does not designate any specific broadcast channel protocol. Instead, only the properties that such protocols should have are described. In a platform standard based on the present document, concrete broadcast channel protocols should be defined according to the relevant network specification. In doing so, more than one protocol may be designated for each protocol model defined in this clause.

7.1 Broadcast channel protocol

For broadcast channel protocols, file transport protocol, packet streaming protocol, and trigger protocol for synchronization of timed media such as AV must be supported.

7.1.1 File transport protocol

File transport protocol is for conveying a set of files via one-way broadcast network. A platform standard based on the present document must support at least one file transport protocol. The properties of a file transport protocol conforming to the present document are as follows:

- Reliable transport of a set of files via repetitive transmissions of the same files.
- Each file in a set of files must be distinguishable via its name represented as a string. This does not necessarily mean that each file is distinguished by a unique string. For instance, since integers can be represented as a string, it is legal to distinguish each file by an integer.
- For a file designated by a name, it should be possible to detect version changes.

Besides these mandatory properties, the following properties may be supported. The present document is designed to accommodate the properties.

- A directory that can list all the files within it can be supported. Such directory structure may be either single depth (that is, only top level directory exists), or multi-depth forming a tree-like structure.
- For each file, various metadata can be associated such as file type (e.g. MIME type).

MOT defined in DAB, object and data carousels defined in DSMCC are examples of such file transport protocols.

7.1.2 Packet streaming protocol

Packet streaming protocol is for continuous streaming of variable or fixed length packets via broadcast channel. A platform standard based on the present document must support at least one packet streaming protocol. The properties of a packet streaming protocol conforming to the present document are as follows:

- The integrity of each packet must be verifiable via a means such as CRC check. That is, each packet is either correctly received or not received at all.

Besides the standard properties stated above, the following additional properties may be supported by each platform standard. The present document is designed to accommodate such properties:

- Some packets may be repeatedly transmitted to a restricted degree to enhance packet reception.
- Additional information such as error correction codes may be attached to packets to enhance packet reception.
- Packets may have addresses for recipients to designate that they are delivered only to the designated recipients.

Packet mode TDC in DAB and sections in DSMCC, and IP packet transport protocol via IP tunneling are examples of such protocols.

7.1.3 Trigger protocol

Trigger protocol is for sending information required to trigger receivers to do designated operations at specific times based on a certain time base. A platform standard may decide not to support any trigger protocol, but if supported, the following properties must be satisfied:

- The time at which a trigger should be activated must be specifiable. But in the present document, no requirement is defined for time precision.
- Arbitrary data must be able to be conveyed together with a trigger.

- For every action that a trigger designates, a unique ID must be assigned to distinguish between the actions. A trigger with the same ID may be transmitted more than once. By doing so, it is possible to enhance reception of triggers, and changes in the trigger time and associated data can be communicated to the receiver.

7.2 Communication channel protocol

Communication channel protocol in this clause-shall be supported only if communication interfaces exist. The present document does not define network dependent protocols in figure 3. Only UDP, TCP and HTTP which are above the IP layer and have direct effects on applications are defined and required to be implemented by the present document.

However, all the defined protocols do not need to be implemented solely within a receiver. For instance, there may be a server-sided gateway and a specialized protocol may be used between a receiver and the gateway while TCP and UDP are supported through the gateway. Service specific protocols are not defined in the present document.

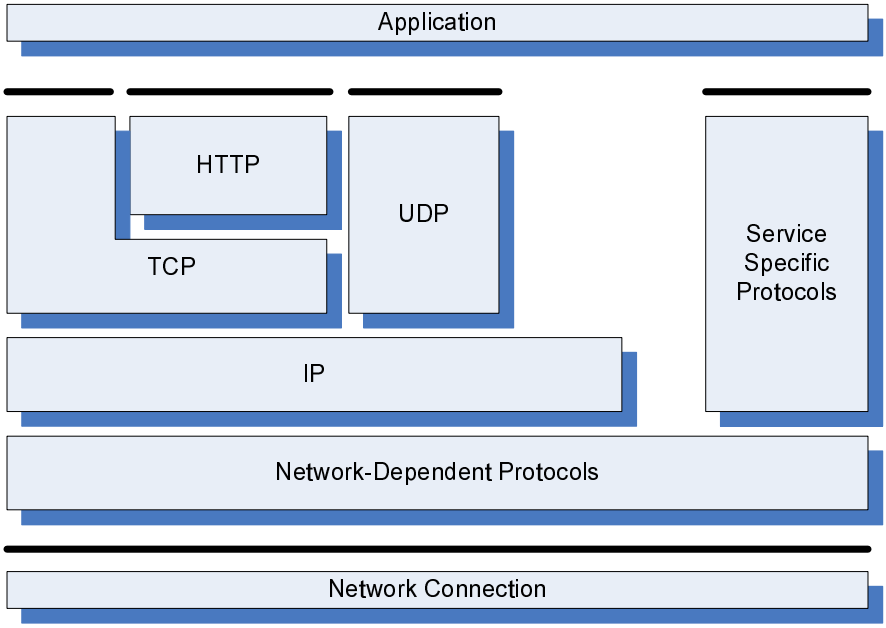


Figure 3: Communication channel protocol stack

7.2.1 Transmission Control Protocol (TCP)

Should support TCP protocol defined in RFC 793 [6].

7.2.2 User Datagram Protocol (UDP)

Should support UDP protocol defined in RFC 768 [7].

7.2.3 Hyper Text Transfer Protocol (HTTP)

Should support HTTP 1.1 protocol defined in RFC 2616 [8] with additional requirements set in the documentation on javax.microedition.io package in MIDP 2.0 [1]. That is, HEAD, GET, and POST methods should be supported, and absolute URIs should be supported as well as relative ones.

7.2.4 Domain Name Service (DNS)

May implement DNS that is defined in RFC 1034 [9] and RFC 1035 [10], and clarified in RFC 1982 [11] and RFC 2181 [12]. But implementation of DNS is optional.

8 Locator model

8.1 Introduction

Locator is a means to indicate an object on a broadcast network in MATE. A locator, for instance, can be used to designate a broadcasting channel, or a file, a packet stream, a video stream, an audio stream, and so on within a broadcast channel. A locator is represented as a string, and Java APIs that take locators as parameters actually accept java.lang.String objects.

Most applications with a few exceptions like EPG can identify objects required during their execution when they are written. For example, list of channels to tune to, files to read, and so on can be identified in advance. On the other hand, specific locations of objects may change depending on the network on which the application is transmitted and the configuration of the transmission scheme even in the same network. Thus, for most applications, locators should be regarded as opaque pointers.

Although the present document does not preclude the possibility that an application recognizes the exact format of locator strings and directly manipulates them accordingly, it is designed with general cases in mind, where locators are simple opaque pointers. Therefore, for applications complying with the present document, knowledge on specific format of locator strings is not required. This means that applications based on the present document can be executed independent of concrete broadcast network specifications.

8.2 Format

The present document does not define a specific format of locator strings. Since the format of locator strings, in general, depends on the logical structure of the underlying broadcast network complying with the present document, it is the responsibility of each platform standard to define such a format.

8.3 Use in APIs

When a locator is specified in the context of an API, which object the locator designates should be designated. For instance, locators designating a multiplex to be tuned to should be specified for tuning APIs, while locators locating files be given to the APIs for reading files. If the actual locators given to APIs are not compliant with the requirement set by the APIs, it is regarded as an error, and java.lang.IllegalArgumentException shall be thrown from the method that took the wrong locator.

9 Security model

9.1 Purpose

The security model defined by the present document has a purpose of providing the following features.

9.1.1 Guarantee of application integrity

Since application modules composing an application can be installed on a receiver through various routes other than broadcast networks, it is possible for application data and codes to be modified for a malicious purpose. Thereby, there should be a way to guarantee that applications installed on receivers are not modified in a way unintended by the original author.

9.1.2 Verification of application provider

In the following cases, identification of application provider is needed:

- When the set of permissions given to an application is decided based on the identity of its provider.

- When it is required to attribute some operations performed by an application to its provider (e.g. when a user is charged for use of a communication line by an application without one's confirmation).
- When an application provided by a provider needs to give other applications from other providers authorities to access services or resources provided by it.

Identity of an application provider must not be forgeable and there must not be any chance the provider can deny its responsibility for applications it provides.

9.1.3 Control of application permissions

Some operations should not be allowed to some applications. For instance, use of a charged communication line generally requires user's prior confirmation. And depending on business models, moving to other channel programmatically by an application should not be allowed if an advertisement is showing. The present document designates a mechanism for granting permissions to an application based on the identity of its provider.

9.1.4 Trace of tasks performed by applications

In some cases, rather than restricting permissions granted to applications to prevent them from performing some operations, the applications need to be allowed to do the operations, and when some problems happen, or there are some reasons to do so, the records of the performed operations may be examined later to put it on a charge for the causes. For example, an application that has passed the authentication process may take out private information that is stored within the receiver. Later when such an activity is discovered, it is possible to blame the application for the illegal activity. For this to be possible, application providers must not be able to deny the responsibility on their applications.

9.1.5 Authority delegation among applications

There may be occasions where applications may need to grant permissions to use resources or services provided by them to other applications they designate to do so. The applications granting the permissions can specify exactly what permissions other applications are granted based on the identities of the application providers.

9.2 Application authentication

9.2.1 Application signing

In the present document, digital signing is used to guarantee application data integrity and verify application provider as described in clause-6.1. However, application signature may not be supported in platforms based on the present document.

To digitally sign an application, the corresponding application definition containing module information and each application module composing the application are signed. Signing an application definition, the following restrictions are imposed on the application modules:

- Application modules containing executable codes must be signed. In other words, receivers must not read codes from unsigned application modules.
- Application modules without executable codes may or may not be signed, but it is recommended to sign them if applications referring to them may behave differently from their intended behaviours in case such modules are forged.

Since an application provider takes charge of operations performed by its application, the provider should decide carefully which modules are to be signed or which are not to prevent its applications from behaving unexpectedly.

9.2.2 Application authentication procedure

Applications can optionally be signed. For a signed application, the corresponding application definition (refer to clause-12.5.5) contains a signature part. The application authentication procedure is as follow:

1) A receiver verifies digital signatures contained in application_definition(). The certificate chains to verify can be located within the certificate message (refer to clause-9.4.1) by keyIdentifiers in signatures.

2) If the verification fails in step 1, the designated application is treated as if it did not exist.

3) If "!" is prefixed to a module ID, the application module corresponding to the ID is verified when it is completely downloaded. Here, again keyIdentifiers are used to locate certificate chains to verify within the certificate message (refer to clause-9.4.1).

4) If the verification in step 3) fails, the designated module is marked as failed to pass the verification.

5) If any module constituting an application has failed to pass the verification in step 4, then the application itself is regarded as failed to pass the authentication, and the application is considered non-existent.

9.2.3 X.509 profile

This clause-defines a subset of X.509 Internet Profile defined in RFC 3280 [13]. The subset is defined to satisfy the minimum requirement set by the present document, and receivers complying with the present document only need to support the subset. This clause-only describes parts different from RFC 3280 [13] or coming with additional restrictions. The following is the X.509 certificate syntax in ASN.1. Refer to RFC 3280 [13] for more details on ASN.1.

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING  }
```

9.2.3.1 signatureAlgorithm

Only "SHA-1 with RSA" shall be supported.

9.2.3.2 tbsCertificate

Restrictions on the fields constituting TBSCertifcate type are as follows:

version: shall always be version 3.

signature: only "SHA-1 with RSA" shall be supported.

issuer

- CN (common name) and C (country) must exist in DN.
- When RDN is defined as a DirectoryString, utf8String shall be used.
- String comparison must be case-sensitive.

validity: shall always be GeneralizedTime.

subject

- If the subject is a CA (Certificate Authority), CN (common name) and C (country) must exist in its DN, and if it is an end entry, it must be encoded as an empty sequence.
- If RDN is defined as a DirectoryString, utf8String shall be used.
- String comparison must be case-sensitive.

subjectPublicKeyInfo: Only RSA is allowed.

issuerUniqueID and subjectUniqueID: Not used.

extensions: Refer to table 1.

Table 1: Profile for standard certificate extensions

Extension	CA	End Entity	Semantic
Basic constraints	C M	NONE	Must exist in a CA certificate, where cA must be set to TRUE. In the case of an end entity this extension shall not exist.
Authority key identifier	NC M (not self-signed), O (self-signed)	NC M	Only keyIdentifier is used.
Subject key identifier	NC M	NC M	Only uses values generated by the method (2) in clause-5.1.2, RFC 3280 [13].
Key usage	C M	C M	For CA certificates, only keyCertSign (5) and cRLSign (6) shall be set. And for end entities, only digitalSignature (0) shall be set.
Subject alternative name	NONE	C M	Shall be set only for end entities. Domain name of the application provider shall be capitalized and set in the form of dNSName. The domain name must be identical to the domain name that is a part of the corresponding application ID.
Extended key usage	NONE	C O	Shall be set only for end entities, and must always be id-kp-codeSigning.
CRL distribution point	NC O	NC O	Shall exist only when CRL is in use. Only a URI on the Web shall be set to distributionPoint. The reasons and the cRLIssuer fields shall not be used.

Table 2: Meanings of acronyms used in CA and End Entity columns in table 1

Acronym	Descriptions
C	Critical. Refer to clause-5 in RFC 3280 [13].
NC	Non Critical. Refer to clause-5 in RFC 3280 [13].
M	Mandatory. Mandated in the context of the profile defined in the present document.
O	Optional. Optional in the context of the profile defined in the present document.
NONE	The corresponding extension shall not exist.

9.2.4 Root certificates and CRL management

The present document does not designate a mechanism for updating root certificates and CRLs. The rationales behind the decision are as follows:

- In general, a root cerficate is valid for 10 years. More than one root certicates with overlapping valid durations may be installed in the receiver. At least one root certificate would be valid during the whole lifespan of a receiver.
- CAs (Certificate Authorities) who issue root certificates are likely to issue certificates to broadcasters and application developers rather than to individuals. Therefore it is very unlikely that CRLs are published.
- If renewal of root certificates or CRLs is required, receivers may decide to do so in a way specific to their implementation. For examples, the renewal can be done via a two-way communication channel if such a channel is available in the receiver. Otherwise, it can be performed when a receiver is connected to a PC via USB interface.

9.3 Application authorization

9.3.1 Introduction

The following factors determine whether applications are authorized to do something or not:

- *Application Requested Permissions:* they are permissions specified in the application definition, and required for the execution of the corresponding application.
- *Receiver Security Policy:* every receiver has a policy for determining which permissions are granted to an application depending on the properties of the application such as the identity of the application provider.

The actual permissions granted to an application are the intersection of the permissions determined by the receiver security policy and those requested by the application.

9.3.2 Notation for permissions

Permission is represented as a string, and such a representation is used to specify permissions requested by an application in the corresponding application definition (refer to clause-9.3.3), and those granted to other applications in credentials (refer to clause-9.3.5).

The detailed format for strings representing permissions may differ from an application type to another. In the case of MIDlet, a format defined in MIDP 2.0 [1] is used with additional wildcard characters defined in the present document. The permissions required to use APIs referenced or defined in the present document are specified in the documentation for each API.

When specifying permissions, it is sometimes convenient to specify a pattern on more than one strings rather than listing all of them. For this, "*" and "?" may be used to designate such a pattern. If it is required to treat the characters as normal ones, "\" (backslash) may be used. That is, to represent "*", "?", and "\" as normal characters, "*", "\?" and "\\" should be specified instead, respectively:

- *: matches a sequence of zero or more characters other than "/"s (slashes).
- **: matches a sequence of zero or more characters possibly including "/"s (slashes).
- ?: matches a character other than "/" (slash).

For example, "dmb.service.ServiceManager.select.default.*" represents permission to select arbitrary services. Here "/" is treated special because of its use in URIs and file paths as the path separator.

9.3.3 Permission request

Applications should specify permissions required for their operation within their definitions. The reasons for verifying permissions requested by an application prior to its execution are as follows:

- To execute applications, only when all the permissions mandated by them are granted.
- To prevent applications from performing unintended actions because of errors or codes inserted with a malicious intent.

9.3.4 Receiver security policy

Security policy of a receiver is a rule for determining permissions granted to an application based on the properties of the application. The present document does not designate how the security policy of receivers is implemented. Each implementation can choose any approach. For example, requested permissions might be granted to all signed applications, and in other cases, for a set of permissions, users might be asked whether to grant them.

9.3.5 Authority delegation

Authority delegation means that an application allows other applications to access services and/or resources provided by it. An application can delegate to other applications only the permissions it owns. To delegate permissions to other applications, an application must create a credential and put it into the definitions of those applications. A credential contains the following, and must be signed by the provider of the application delegating the permissions:

- List of permissions to delegate.
- Valid duration of the credential.

9.4 Formats of the relevant messages

9.4.1 Format of certificate message

A certificate message contains several certificate chains. Its format is described in table 3. Each chain includes all certificates from the end entity up to the certificate right under the root certificate.

Table 3: Format of certificate message

Syntax	No. of Bits	Mnemonic
certificate_message(){		
certificate_chain_count	16	uimsbf
for(i=0;i<certificate_chain_count;i++) {		
certificate_count	16	uimsbf
for(j=0;j<certificate_count;j++) {		
certificate_length	24	uimsbf
certificate()		
}		
}		
}		

certificate_chain_count: number of certificate chains in this message.

certificate_count: number of certificates within a chain (excluding the root certificate).

certificate_length: length of a certificate in bytes.

certificate(): Certificate data structure defined in ITU-T Recommendation X.509 [18].

9.4.2 Signature format

The signature structure referred to in clause-12.5 represents a digital signature, and follows the ASN.1 DER structure shown below:

```
Signature ::= SEQUENCE {
    certificateIdentifier AuthorityKeyIdentifier,
    hashSignatureAlgorithm OBJECT IDENTIFIER,
    signatureValue BIT STRING }
```

certificateIdentifier: used to identify the certificate containing the public key for verifying the signature. Its structure is as follows. Within the scope of the present document, it shall be enough for a receiver to use keyIdentifier when identifying the key used for signature verification.

```
AuthorityKeyIdentifier ::= SEQUENCE {
    keyIdentifier [0] KeyIdentifier OPTIONAL,
    authorityCertIssuer [1] GeneralNames OPTIONAL,
    authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL }
```

hashSignatureAlgorithm: represents the hash algorithm used for the signature. Within the scope of the present document, only SHA-1 be allowed. Note that RSA, which is the encryption algorithm used for signature generation, is specified in SubjectKeyInfo within the certificate. That is why only hash algorithm is specified here.

9.4.3 Credential format

The credential structure referred to in clause-12.5.3.5 represents a credential, and its structure is shown in table 4.

Table 4: Format of credential

Syntax	No. of Bits	Mnemonic
credential {		
grantor_id		utf8
expiration_date	64	uimsbf
permission_count	16	uimsbf
for(i=0;i<permission_count;i++) {		
permission		utf8
}		
signature		signature
}		

grantor_id: ID of an application granting permissions. The domain name part of the ID must coincide with the capitalized domain name in dNSName field of "subject alternative name" of the relevant certificate.

expiration_date: the time when this credential expires. It is the milliseconds passed since January 1st, 1970 UTC.

permission_count: number of strings representing the permissions to grant (refer to clause-9.3.2).

permission: a string representing a permission to grant (refer to clause-9.3.2).

signature: signature obtained by signing all the values specified above in this structure (refer to clause-9.4.2).

10 Graphic system model

10.1 Introduction

MATE supports a means to control the size and the location of videos presented on the screen, and also provides a basic mechanism for presenting graphics to and getting inputs from users. In the receiver implementations conforming to the present document, display devices capable of presenting only graphics and those presenting both graphics and videos at the same time can coexist.

Figure 4 depicts a display device capable of presenting videos. Both video and graphics planes are identical in their sizes, and occupy the whole screen. On the other hand, in a display device that cannot present a video, it contains only a graphics plane.

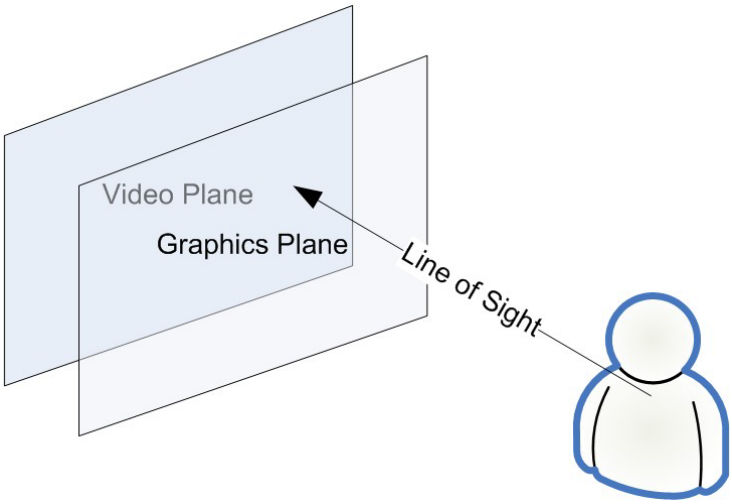


Figure 4: Display model

10.2 Video plane

On a video plane, one or more videos may be presented, and there is a Z-order associated with each of them. Since each video is fully opaque, one closer to the eyes will obscure another farther from the eyes when they overlap.

The present document is deliberately silent on the area on a video plane that lies outside video presentations when videos are not covering the whole screen area. Therefore, to get a predictable result on the area outside those covered by video presentations, the uncovered area in the graphics plane should be painted opaque with a colour of application's choice.

10.3 Graphics plane

Graphics are always presented above video presentations, and a graphics plane is fully transparent. But if a display device does not support video presentation, then the graphics plane in that device shall be opaque. The aspect ratio of a pixel in a graphics plane is always assumed to be 1:1 meaning it is a square, even if it is physically not.

In the case of display devices capable of presenting videos, the levels of transparencies that can be designated within a graphics plane against the corresponding video plane may differ depending on devices. All the receivers, conforming to the present document, shall distinguish at least between fully-transparent and fully-opaque pixels.

More than one application can present graphics on a graphics plane at the same time, but it is not mandated. But, any platform standard based on the present document may mandate the capability to present more than one application simultaneously. When more than one application is presented on a graphics plane simultaneously, and there are overlaps among graphics presented by those applications, the end result must be identical to the case where graphics from the application that lies farther from the eyes are drawn first, and the rest in the same way to the top most application, and at least, fully transparent pixels are treated as such (i.e. any non-transparent approximations are not allowed) in composing graphics from multiple applications. But it is also permitted to draw graphics from an application onto a separate buffer, and then move the final result at once to the graphics plane.

10.4 Composing a screen

A graphics plane is overlaid atop its corresponding video plane, where the source over rule defined in Porter-Duff [19] shall be used. Depending on the levels of transparencies supported by the graphics plane, the end result of the plane composition may differ.

11 Application model

11.1 Introduction

An application can be executed only after all the constituting modules are stored. Once executed, it follows the application lifecycle defined in this clause.

11.2 Application storage and removal

11.2.1 Storage

The present document does not designate properties of application storages in detail. For instance, it is even allowed to store applications in a volatile memory. And the minimum amount of time for which a stored application should be retained is also not specified. Therefore, it is even legal to remove an application as soon as it is fully stored. But when an application is removed, receivers shall perform any cleanups designated in the present document such as reclaiming resources used by the application.

11.2.2 Receiver policy

In the present document, no specifics on application storage, removal, and state changes in the lifecycle are designated except for those defined in the application signalling part of the present document. So receivers may change state of applications according to their own policies.

For example, receivers may define a rule to remove applications upon memory shortage, and can provide users with appropriate interfaces for downloading, removing, launching, pausing, starting, and stopping applications.

As a part of the application signalling, the present document designates control information for application storage, removal, and lifecycle management, and it is recommended that receivers follow the control information. However, it is also permitted for receivers to deal with it differently from what is designated in the present document depending on their policy.

11.3 Application storage, update, and removal

11.3.1 Application download

Download and storage of an application may be initiated via application signalling or by other policy specific to each receiver (refer to clause-11.2.2). An application may be launched only when all the modules composing the application are fully downloaded.

11.3.2 Application update

When a version change in an application is detected via application signalling, the application may be updated anytime. Here if invalidate_previous_version field defined in clause-12.5.5 is 1, the current version of the application should be terminated as soon as a new version of the application is detected, and should not be launched until the new version is downloaded. Conversely, if invalidate_previous_version is 0, the current version can be run and launched.

In general, current versions of applications may not be used (that is, its invalidate_previous_version is set to 1) because of changes in the application codes.

11.3.3 Application removal

An application may be removed at any time except when it is running, according to the receiver policy (refer to clause-11.2.2). Besides when a receiver detects that application_definition() of an application disappeared and it contained signal bound descriptor, then the corresponding application may be removed.

11.4.4 Lifecycle

Upon completion of downloading modules constituting an application, the application may be launched in the following cases:

- When a service is selected and associated with the application via a service binding message defined in clause-12.3.1.3.
- When a service is selected, associated with the application via a service binding message defined in clause-12.3.1.3 in a way that the association is specific to a certain period of time, and the execution of the application is signalled via the application control message defined in clause-12.3.1.4.
- Designated to launch by a receiver-specific policy (refer to clause-11.2.2).

A running application goes through the following lifecycle depicted in figure 5.

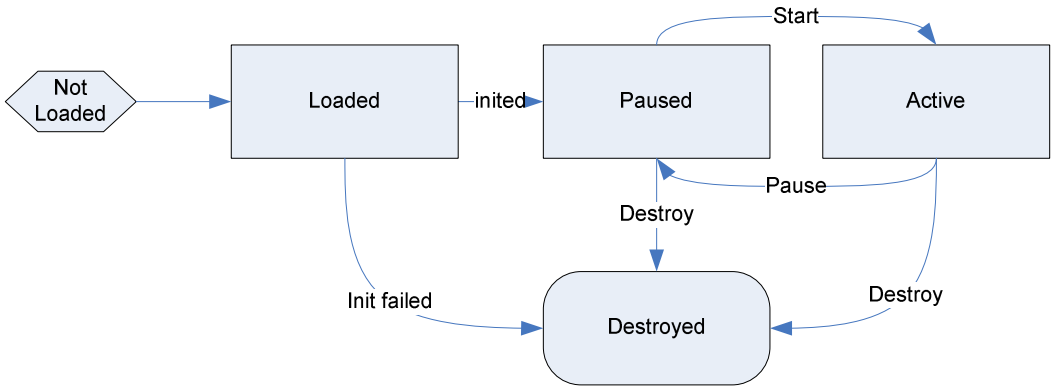


Figure 5: Application lifecycle

11.4.4.1 Loaded state

The loaded state is a state in which codes and data required for execution of an application are loaded into memory after launching of the application is requested. In this state, initialization of the application immediately begins. If the initialization is successful, the application moves into the paused state. Otherwise, it moves directly to the destroyed state.

11.4.4.2 Paused state

Applications that completed its initialization move to the paused state. Besides those that were already activated may move to the paused state in the following cases:

- When applications are voluntarily moving to the paused state.
- When designated by receiver policy (refer to clause-11.2.2).

In this state, applications regardless of their types should follow the following basic rules:

- Graphics drawn to the screen, sounds, and so on should be minimized not to interfere with other applications.
- Should release any shared resources such as video decoders as much as possible.
- Should reduce resource consumption to its minimum.
- The application should be able to return to the active state promptly upon requests.

11.4.4.3 Active state

The active state represents a state in which an application is operating actively.

11.4.4.4 Destroyed state

The destroyed state is a state in which an application is destroyed. Applications in this state no longer own the resources they owned while running, and are removed from the memory. Applications are destroyed in the following cases:

- Upon a change of services where an application is associated with the previous service via application signalling, but not with the next service.
- When termination of an application is requested via application signalling.
- When an application voluntarily requests self-termination.
- When other application requests termination of an application via an appropriate means (such as API calls).
- When designated to do so by receiver policy. (refer to clause-11.2.2).

11.5 MIDlet model

MIDlet is as defined in MIDP 2.0 [1]. This model is compatible with the model defined in figure 5. The original MIDlet model has 3 states, seemingly different from the 4-state model specified in the present document. But within the scope of the present document they are considered compatible for the following reason.

The loaded state corresponds to the state in MIDlet where bytecodes constituting a MIDlet are loaded into memory right before the constructor of the MIDlet is invoked. Initialization occurs when the Java constructor is invoked, and additionally, if an exception is thrown in the constructor, it is regarded as if the MIDlet immediately moved into the destroyed state.

12 Application signalling and transport

12.1 Application module

12.1.1 Definition and purpose of application module

Application module is the basic unit for composing an application, and also the unit in the transport of an application. An application consists of one or more application modules, and all codes and data constituting an application are transmitted, received, and stored in the unit of an application module.

Application modules constituting applications are identified by their application module ID. Therefore, each application module must be assigned a unique ID. Different from URLs, application module IDs are independent of the locations of the corresponding modules on broadcast or communication networks. So the same application module may be received via broadcast or communication network, and alternatively, installed manually by a user. As an effect, when more than one application modules containing program codes share the same ID, and their implementations are different from one another, they can be used interchangeably in an application as long as their interfaces are identical.

The concept of application module has the following benefits:

- Bandwidth and memory savings: codes and data shared among applications can be transferred only once, and shared within the receiver.
- Easy updates of applications: an application may be decomposed into multiple modules depending on their update characteristics, and some of the modules may be updated separately. As such, application management becomes easy.
- Flexibility in application distribution: application modules constituting applications may be obtained via various routes including broadcast and/or communication channels.

12.1.2 Structure of application module

Each application module consists of metadata containing additional information on the module and a body. The body may be in either ZIP format or an application-defined format. With the evolution of the present document, additional formats for the body may be added.

12.1.2.1 ZIP format

The body in ZIP format follows the structure of the standard ZIP file format [20], and may contain multiple files including Java class files and image files. In the present document, an API for reading files within application modules in ZIP format is provided.

And if such an application module is used to constitute a Java application, it is added to the classpath of the application. In that case, class files in the application module may be loaded from it, and contents of files in the application module may be retrieved with getResourceAsStream(String) of java.lang.Class.

Note that Java defines JAR [21] format based on ZIP file format. JAR format is extended from ZIP format by adding metadata in the META-INF directory created within ZIP files. For the present document, JAR format is not used in favour of the efficiency in transporting and handling application modules. Instead, additional information and digital signature related data are stored in the metadata portion of application modules mentioned before. Therefore, it is not possible to digitally sign only a subset of files within a ZIP format application module, different from the standard JAR file. If there are files not requiring digital signing such as large image files, they need to be separated into another application module.

12.1.2.2 Application-defined format

This is a format defined by each application, and applications may be free to decide the format of their application modules. The present document defines only APIs for retrieving binary data constituting the body of application modules. If an application module in an application-defined format is used as part of a MIDlet, then JVM does nothing for it.

12.1.3 Application module ID and version

Each application module is assigned an ID and a version.

Application modules are identified by IDs of the following format.

```
[!]<domain_name>/<module_specific_name>
```

!: If exists, it means that the corresponding application module is digitally signed.

domain_name: An internet domain name owned by the organization providing the application module. Due to this scheme, unique IDs may be assigned to application modules without a separate registry for IDs.

module_specific_name: A name for identifying the application module within the organization creating it.

Since the IDs in this format may be used as a part of URLs, the IDs shall not contain characters violating RFC 2396 [14]. The follows are examples of application module IDs. The former is an ID for an unsigned application module, and the latter is for a signed application module.

```
xyz.com/images
!foo.com/library/ui/textentry
```

The version of an application module is a 32-bit unsigned integer. When two application modules have an identical ID, the one with newer version must be backward compatible with the other with older version. When a new version of a software product is not compatible with the previous version, or it is required to manage versions of an identical module separately, different IDs should be assigned so that the modules are practically treated as if they were different.

12.1.4 Accessing contents of application module

12.1.4.1 Introduction

The content of an application module can be read by the following two methods:

- *getResourceAsStream(String) of java.lang.Class*: Applicable only to the application modules in ZIP format. Application modules in ZIP format are added to application's classpath when they are used as part of Java applications. Therefore files in application modules can be read using this method in this case.
- *Use of application module URL*: By passing a URL to an application module or to a file within it to an open method of javax.microedition.io.Connector as a parameter, a javax.microedition.io.InputConnection may be obtained. In turn, a java.io.InputStream for reading the content of the application module or a file within it can be obtained from the connection. But if the designated application module is not fully received, an IOException is thrown from Connector.open. Here, available() method of the InputStream obtained as described above shall return the total size in bytes of the application module or a file within it.

If an application module URL is used, it is possible to read the content of an application module that is not a part of the application using the URL. When reading application modules belonging to other applications, the application reading them must have the following permission:

```
dmb.module.read.<module_id>
```

```
where <module_id> is the ID of the application module to read.
```

12.1.4.2 URL to an application module

The format of a URL locating an application module is as follows:

```
module://<module_id>[!<path>]
```

module_id: application module ID.

path: in the case of an application module in ZIP format, this is the name of a file within the body of the application module. Thus it points to a specific file within the ZIP file. If the name does not begin with "/", no "/" shall be prefixed.

An application module URL can be in one of the following two formats:

- When designating the body of an application module:
  - *Format:* module://<module_id>
  - *Example:* module://xyz.com
- When designating a file within the body of an application module, and the format of the body is ZIP:
  - *Format:* module://<module_id>!<path>
  - *Example:* module://xyz.com/moduleA!com/xyz/image.gif

12.1.5 Compression of application module

Basically compressions performed in the protocol level of the transport network (e.g. MOT in case of terrestrial DMB) are not used, but when the body of an application module is in ZIP format, compressions can be performed on the files within it as designated in the ZIP specification. On the other hand, no compression mechanism is defined in the present document for application modules in application-defined formats.

12.1.6 Transport of application module

There is no restriction in the kind of networks and protocols that are used to transport application modules. The location where an application module resides is designated with a URL as defined in clause-12.3.1.2. Each platform standard should define the types of URLs that can be specified there.

In addition to having URLs to an application module specified, the present document defines a mechanism to specify time intervals during which an application module can be downloaded from an associated URL. Therefore it is made possible to operate bandwidth in time divided manners by transmitting different modules in different time intervals.

12.1.7 Signing application module

An application module can selectively be signed. For details on the digital signing, refer to clause-9.2.1 "Application Signing." If an application module is signed, the ID of the application module must begin with a "!" to indicate the fact.

12.2 Application ID

An application is identified by an ID of the following format:

```
<domain_name>/<application_specific_name>
```

domain_name: An internet domain name owned by the application provider. Due to this scheme, unique IDs may be assigned to applications without a separate registry for IDs.

application_specific_name: A name identifying each application among those provided by the same application provider.

An application ID shall not contain any character violating RFC 2396 [14]. The followings are examples of the application ID:

```
foo.com/game/soccer (domain_name: foo.com, application_specific_name: game/soccer)
bar.com/epg
```

12.3 Application signalling

In this clause, a mechanism is defined for announcing presence of applications on broadcast networks so that receivers can download and launch them via various networks including broadcast networks. The signalling defined in this clause-is irrelevant to a specific application type, and may be applied to application types other than Java applications. This clause-designates overall rules for the application signalling, and the detailed format of the messages is defined in clause-12.5.

12.3.1 Signalling structure

Five kinds of signalling messages are defined for the application signalling in the present document. All five kinds of messages can be associated with each other. A set of signalling messages in such association is called a signalling message group. An ensemble may have one or more signalling message groups. Those messages must be transmitted repeatedly with the cycle times designated in the present document, using appropriate mechanisms available in the underlying network. Figure 6 shows the relationships among services, application modules, and the 5 signalling messages.

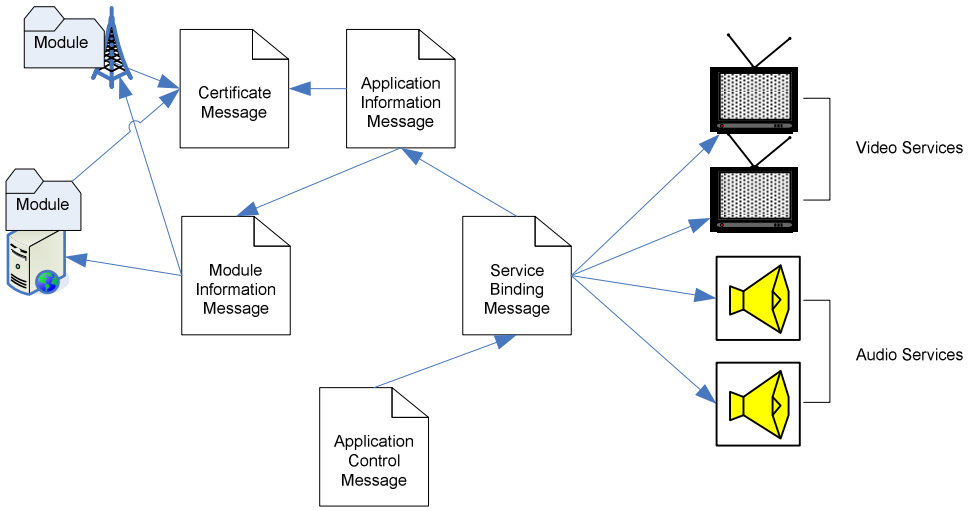


Figure 6: Signalling structure

12.3.1.1 Application information message

An application information message contains the following information on an application. Each application information message may define multiple applications:

- The application ID, the version, and the type.
- The profile and its version for receivers capable of running the application.
- Dependencies on other applications.
- Application modules constituting the application.
- Other attributes associated with the application (e.g. whether it will launch automatically).
- Information intended to be shown to users including icons, names, descriptions, and so on.
- Signatures to prevent forgery of the above mentioned contents.

In addition to those listed above, additional information may be added per application type, and in the form of descriptors.

12.3.1.2 Module information message

A module information message contains information on the modules referenced by applications defined in the corresponding application information message. That is, an application module appeared in an application information message must also appear in the associated module information message. A module information message contains the following information for each module described there:

- The application module ID and the version.
- Size of the application module.
- URLs pointing to locations where the application module can be downloaded.
- Time intervals per URL during which the application module can be downloaded from the location represented by the URL.

For each application module, more than one URL may be specified. In this case, it depends on the receiver's internal policy which URL is used to download the application module. The present document does not designate any specific rule on this matter.

Also there is no restriction on the URLs that can be specified. An application module may be downloaded via a communication channel using HTTP, or via the same broadcast channel as the one conveying the signalling messages or via a different broadcast channel.

12.3.1.3 Service binding message

A service binding message associates each service with applications relevant to the service. An application may be designated to launch when its associated service is selected, or additionally designated to do so via the corresponding application control message.

By referencing a service binding message, applications associated with each service can be identified even when they are not to launch immediately. Therefore when a user selects a service, the user can see the list of applications associated with the service prior to their launch time, and designate applications to download.

As an example, if a user knows the presence of an application providing additional information on a show that is on-air on Saturday afternoon and initiates its download in advance, the constituting application modules will be downloaded during the whole week.

12.3.1.4 Application control message

An application control message conveys control signals to launch or terminate applications associated with a service by its corresponding service binding message. Via the application control message, applications can be launched and terminated upon a specific event within the service.

An application control message is very small in size because it refers to binding tags defined in the corresponding service binding message.

12.3.1.5 Certificate message

A certificate message contains certificates for digital signatures in the corresponding application information message and application modules. For the detail, refer to clause-9.4.1.

12.3.2 Message transport

The messages defined in clause-12.3.1 are transported via a network-specific mechanism. The application control message shall be transmitted at least every 1 second. Other messages shall be transmitted at least every 10 seconds.

Within a single ensemble, more than one signalling message group may be transmitted. Additional restrictions may be defined for each network type. For example, if an ensemble is fully controlled by an operator, only a single signalling message group may be permitted, but if more than one independent service operator is managing an ensemble, a signalling message group per service is designated to be transmitted.

12.3.3 Message monitoring

A receiver must monitor version changes in the signalling messages within the current ensemble it is tuned to as described below:

- It is recommended that a receiver monitors all the signalling messages within the current ensemble.
- At least the signalling messages associated with the service that is currently selected shall be monitored.
- A receiver shall be able to detect a version change within a second.

12.4 Application state control

Application signalling messages affect storage, update, removal, and state transition of applications.

12.4.1 Application download

Download of an application is, in principle, initiated by a user's request, and upon the completion of the download, the application is installed in the receiver via an installation procedure. But if the autodownload descriptor (refer to clause-12.6.5.4) is present in signed_descriptor_loop or unsigned_descriptor_loop of application_definition() within the application information message, download of the corresponding application is automatically initiated. But receivers may still ignore it depending on their policy or user settings.

12.4.2 Application update

Upon detection of a change in the version of an application, download of new application modules is immediately initiated. When this is the case, the previous version may be designated to be either invalidated immediately or retained for execution until the new version is downloaded completely.

If invalidate_previous_version field in application_definition() is set to 0, then the previous version can be executed while the new version is being downloaded. On the other hand, if it is 1, the previous version cannot be executed until the new version is downloaded completely, and the update is completed.

12.4.3 Application removal

In general, applications are removed on requests from a user. But if an application_definition() includes a signal bound descriptor defined in clause-12.5.6.5, the application may automatically be removed when the application_definition() disappears.

12.4.4 Application execution

An application can be launched by user's requests. Besides, it may be launched in the following cases when it is associated with a service via a service binding message:

- If event_bound bit is set to 0, the application is automatically launched upon the selection of the associated service.
- If event_bound bit is set to 1, the receiver launches the application associated with the user selected service only when the binding_tag corresponding to the binding of (service, application) pair is present in the corresponding application control message.

12.4.5 Application termination

An application may be terminated voluntarily or upon requests from user via appropriate user interfaces. Besides it may be terminated in the following cases:

- An application automatically launched as a result of the selection of the current service is terminated when a different service is selected and the application is not bound to the service switched to.
- With the event_bound bit set to 1, if a different service is selected, the application is bound to the service switched to, and the binding_tag for the binding of (new service, existing application) pair is absent in the corresponding application control message, the application is terminated.

If an application is a MIDlet, the destroyApp(boolean) method is invoked upon its termination. Here, if kill bit in the service binding message is set to 1, a true is passed to the method, and if 0, a false is passed.

12.5 Application module and message formats

12.5.1 Relationship with platform standards

A platform standard based on the present document is recommended to follow the message formats defined in this clause-as they are, and to define only protocols used to transport the messages. But modifications specific to a platform is permitted.

12.5.2 Message version

The version of each message is a 32-bit integer large enough to allow testing of the validity of the cached version of the message by comparing the version of the cached message and that of the message being broadcast, even when the message is not monitored for relatively long period of time. The version is incremented by 1 for every change, but permitted to increase by more than 1 when it is required for managerial reasons.

A receiver repeatedly checks and receives new message. When a version of a message is received 720 hours (1 month) ago, it is always considered invalid. Therefore, at any time interval of 720 hours, the same version must not be reused.

In general, merely a change in the version of a message is considered to signify the transmission of a new version. But when it is important to know which one is newer as in the case of the service binding message, the following additional rule is applied. That is, if the absolute value of the difference between two versions exceeds 0x7FFFFFFF, then a version with a smaller value is considered newer than the other. Therefore within any 720-hour interval, any two versions of a message must not differ by a half of 32-bit value range.

12.5.3 Common data format

This clause-defines common data types intended to be used in defining the formats of the application module and the messages.

12.5.3.1 UTF-8 string

A string encoded in UTF-8 format as defined in ISO 10646-1 [21].

Table 5: Format of a UTF-8 string

Syntax	No. of Bits	Mnemonic
utf8 {		
Length	16	
for(i=0;i<length;i++) {		
char_byte	8	bslbf
}		
}		

length: length of the string in bytes.

char_byte: a byte constituting the string.

12.5.3.2 Descriptor

A descriptor used to insert additional information to each message.

Table 6: Format of a descriptor

Syntax	No. of Bits	Mnemonic
descriptor(){		
tag	16	uimsbf
length	16	uimsbf
for(i=0;i<length;i++) {		
data_byte	8	bslbf
}		
}		

tag: the identifier designating the type of the descriptor.

length: length of the content of the descriptor in bytes.

data_byte: a byte constituting the content of the descriptor.

12.5.3.3 Descriptor loop

A descriptor loop is used where one or more descriptors can be inserted.

Table 7: Format of a descriptor loop

Syntax	No. of Bits	Mnemonic
descriptor_loop{		
descriptor_count	16	uimsbf
for(i=0;i<descriptor_count;i++){		
descriptor()		
}		
}		

descriptor_count: number of descriptors.

12.5.3.4 Digital signature

A digital signature is represented as signature type. The signature type is defined in clause-9.4.2 of the present document.

12.5.3.5 Credentials

A credential for delegating permissions to an application is represented as credential type. The credential type is defined in clause-9.4.3 of the present document.

12.5.4 Application module format

The format of the application module is as follows. Additional information to an application module is always appended at the end of the application module. This scheme has a benefit of being able to treat stored application modules, as they are, as plain ZIP files. This additional information has to be treated as a comment from the view point of a ZIP file. Therefore, "zipfile comment length" in the ZIP file must be set to regard this additional information as "zipfile comment" in "End of central dir record" at the end of the ZIP file in accordance with the ZIP format [20].

Table 8: Format of an application module

Syntax	No. of Bits	Mnemonic
module(){		
module_payload	N	Bslbf
id		utf8
version	32	Uimsbf
type	8	Uimsbf
module_descriptor_loop		descriptor_loop
signature_count	8	Uimsbf
for(i=0;i<signature_count;i++){		
module_signature		Signature
}		
footer_length	16	Uimsbf
}		

module_payload: the data constituting the body of the application module.

id: ID of the application module.

version: version of the application module. The larger is the newer.

type: type of the module_payload. Each constant defined per-type is shown in table 9.

Table 9 - Types of the application module

Value	Description
0x00	Unused
0x01	ZIP format
0x02~0xFE	Reserved for future extensions
0xFF	Application defined format

module_descriptor_loop: a descriptor loop for describing additional information on the application module.

signature_count: number of module_signatures.

module_signature: a digital signature for the application module. All the content above signature_count exclusive of the signature_count is signed together with module_payload.

footer_length: size of the additional information on the application module. Number of bytes from right above footer_length to right before module_payload.

12.5.5 Format of application information message

Table 10: Format of an application information message

Syntax	No. of Bits	Mnemonic
application_information_message(){		
message_version	32	Uimsbf
application_count	16	Uimsbf
for(i=0;i<application_count;i++){		
application_definition_length	16	uimsbf
application_definition()		
}		
common_descriptor_loop		descriptor_loop
}		

*message_version: **version of the message.***

application_count: number of applications described in the application information message.

application_definition_length: length of application_definition().

application_definition(): defines each application. The format is defined in table 11.

common_descriptor_loop: descriptors applied to all applications listed in the application information message.

Table 11: Application definition

Syntax	No. of Bits	Mnemonic
application_definition(){		
application_id		utf8
application_version	32	uimsbf
invalidate_previous_version	1	bslsf
reserved	7	"1111111"
application_type	8	uimsbf
application_priority	8	uimsbf
profile_count	8	uimsbf
for(i=0;i<profile_count;i++){		
profile	8	uimsbf
profile_version	24	uimsbf
profile_descriptor_loop		
}		
visibility	8	uimsbf
module_count	8	uimsbf
for(i=0;i<module_count;i++){		
module_id		utf8
module_min_version	32	uimsbf
}		
signed_descriptor_loop		descriptor_loop
signature_count	8	uimsbf
for(i=0;i<signature_count;i++){		
application_signature		signature
}		
unsigned_descriptor_loop		descriptor_loop
}		

application_id: ID of the application being defined.

application_version: ***version of the application. The larger is the newer.***

invalidate_previous_version: if this is set to 1, it means that the current version of the application should not be used upon detection of a new version. If the current version is running, it should be terminated immediately. On the other hand, if 0, the current version can be run while downloading the new version.

application_type: ***application type.*** Among the types of applications, a data-only application cannot be executed, and such an application type is added to treat a set of data as a separate application to be able to reference by other applications as a depending application. For data-only applications, it is possible to access the content of the modules belonging to them with the application module URLs.

Table 12: Application types

Value	Description
0x00	Unused
0x01	MIDlet
0x02~0xFF	Reserved for future extensions

application_priority: priority of the application. It is to assist receivers in judging relative importance of the application. Receivers may give priorities to some applications based on their priority when the remaining memory space is low, or some other resources are insufficient to run all the applications. A larger value means a higher priority.

profile_count: number of platform profiles against which the application can run.

profile: platform profile ID on the platform of the profile the application can run. Specific values to be specified here are supposed to be defined in platform standards. A profile corresponding to a receiver must be shown here to be able to run the application on the receiver.

profile_version: a version of the profile. If the version of a receiver is greater than this, it can run the application.

profile_descriptor_loop: ***a descriptor loop containing additional information on the profile***

visibility: visibility of the application from the viewpoint of both user interfaces and application control API. The list of values eligible for this field is defined in table 13.

Table 13: Application visibility

Value	Description
0x00	Unused
0x01	User visible
0x02	API visible
0x03~0xFE	Reserved for future extensions
0xFF	Hidden. Used when applications are transmitted for testing. Thus normal receivers must treat it as non-existent

module_count: ***number of application modules constituting the application.***

module_id: the ID of an application module constituting the application.

module_min_version: the minimum version of the application module.

signed_descriptor_loop: ***a descriptor loop containing additional information on the application. Since the content of this loop is signed when this application definition is signed, descriptors that must not be forged must be listed here.***

signature_count: ***number of digital signatures for this application definition.*** If one of them is successfully verified, this application definition is considered as valid.

application_signature: ***a signature signing from*** application_id to signed_descriptor_loop inclusive. For details, refer to the security model (clause-9.2).

unsigned_descriptor_loop: ***a descriptor loop containing additional information on the application. This loop is excluded when the application definition is signed. Therefore, descriptors that need to be referred to before the signature verification should be placed here.***

12.5.6 Application related descriptors

This clause-defines descriptors that can be placed in descriptor_loops within the application information message.

12.5.6.1 Module download descriptor

A module download descriptor is used to designate application modules that contain icons or some other data referenced by the application definition itself. If this descriptor is placed in common_descriptor_loop of an application information message, then the application modules are automatically downloaded after receiving the application information message. If an application module is removed from the application information message in the next version, the corresponding application module is removed either.

Table 14: Module download descriptor

Syntax	No. of Bits	Mnemonic
module_download_descriptor(){		
tag	16	uimsbf
length	16	uimsbf
module_count	8	uimsbf
for(i=0;i<module_count;i++){		
module_id		utf8
module_min_version	32	uimsbf
}		
}		

tag: 0x0001

length: length of the content of the descriptor following this field in bytes.

module_count: number of application modules to download.

module_id: ID for an application module to download.

module_min_version: the minimum version of the application module to download.

12.5.6.2 Application description descriptor

An application description descriptor contains the names and the descriptions of an application, which are intended to be presented to end users. This descriptor can be placed in signed_descriptor_loop or unsigned_descriptor_loop of application_definition(). For each application, more than one descriptor can be specified per each language.

Table 15: Application description descriptor

Syntax	No. of Bits	Mnemonic
application_description_descriptor(){		
Tag	16	uimsbf
Length	16	uimsbf
language_code	n*8	bslbf
Null	8	"00000000"
Name		utf8
Description		utf8
}		

tag: 0x0002

length: length of this descriptor from right after this field to the end in bytes.

language_code: a language code defined in RFC 3066 [15].

null: a terminator to mark the end of the language_code field above.

name: ***name of the application in the language represented by*** language_code.

description: ***description of the application in the language represented by*** language_code.

12.5.6.3 Application icon descriptor

An application icon descriptor designates the location of an icon to be presented to end users. This descriptor may be placed either signed_descriptor_loop or unsigned_descriptor_loop of application_definition(). The icon is assumed to be inside an application module, must be encoded in PNG format and at least 16 by 16 pixels in size though there is no designated limit on the size of the icon. Receivers may expand or shrink the size if necessary.

This descriptor may be listed more than once for an application, and in that case, any of them may be used. Therefore each descriptor would be preferable to indicate a distinct location but the same icon.

Table 16: Application icon descriptor

Syntax	No. of Bits	Mnemonic
application_icon_descriptor(){		
Tag	16	uimsbf
icon_locator		utf8
}		

tag: 0x0003

icon_locator: an application module URL (clause-12.1.4.2) excluding the "module://" part.

12.5.6.4 Autodownload descriptor

An autodownload descriptor designates the corresponding application to be downloaded without user request. This descriptor is placed into signed_descriptor_loop or unsigned_descriptor_loop of application_definition(). If placed, the application corresponding to the application_definition() is automatically downloaded. A receiver may ignore this descriptor depending on its policy. This descriptor is valid once specified until detecting that it disappeared.

Table 17: Autodownload descriptor

Syntax	No. of Bits	Mnemonic
autodownload_descriptor(){		
Tag	16	uimsbf
Length	16	uimsbf
}		

tag: 0x0004

length: length of the content in the descriptor following this field. For this descriptor, this is always 0.

12.5.6.5 Signal bound descriptor

A signal bound descriptor is for designating that an application should be removed when it is no longer present in the signalling. This descriptor is placed in signed_descriptor_loop or unsigned_descriptor_loop of application_definition().

Table 18: Signal bound descriptor

Syntax	No. of Bits	Mnemonic
signal_bound_descriptor(){		
Tag	16	uimsbf
Length	16	uimsbf
}		

tag: 0x0005

length: length of the content in the descriptor following this field. For this descriptor, this is always 0.

12.5.6.6 MIDlet descriptor

A MIDlet descriptor describes additional information on a MIDlet. If the type of an application is MIDlet, this descriptor must be present in signed_descriptor_loop of application_definition().

Table 19: MIDlet descriptor

Syntax	No. of Bits	Mnemonic
midlet_descriptor(){		
Tag	16	uimsbf
Length	16	uimsbf
initial_class		utf8
parameter_count	16	uimsbf
for(i=0;i<parameter_count;i++){		
parameter_name		utf8
parameter_value		utf8
}		
permission_count	16	uimsbf
required_permission_count	16	uimsbf
for(i=0;i<permission_count;i++){		
Permission		utf8
}		
credential_count	8	uimsbf
for(i=0;i<credential_count;i++){		
application_credential		credential
}		
}		

tag: 0x0006

length: length of the content of this descriptor following this field.

initial_class: ***name of the*** MIDlet class (fully qualified name).

parameter_count: ***number of parameters to be read with*** MIDlet.getAppProperty(String).

parameter_name: name of a parameter. It corresponds to a name passed to MIDlet.getAppProperty(String).

parameter_value: value of a parameter. It corresponds to a return value from MIDlet.getAppProperty(String).

permission_count: number of permissions the application requests.

required_permission_count: number of permissions that are required for the operation of the application from the beginning of the whole list of permissions requested by the application. That is, those permissions designated by this count must be granted for the application to execute.

permission: a permission the application requests for its operation. The format of this permission string is defined in clause-9.3.2.

credential_count: number of credentials for the application.

application_credential: an application credential. For the details, refer to clause-9.4.3.

12.5.6.7 Profile extension descriptor

A profile extension descriptor describes a profile extension to designate additional characteristics of the receiver eligible to running the corresponding application. It is placed in profile_descriptor_loop of application_definition().

Table 20: Profile extension descriptor

Syntax	No. of Bits	Mnemonic
profile_extension_descriptor(){		
Tag	16	Uimsbf
Length	16	Uimsbf
profile_extension_class_id	16	Uimsbf
for(i=0;i<N;i++){		
profile_extension	8	Uimsbf
}		
}		

tag: 0x0007

length: length of the content of this descriptor following this field.

profile_extension_class_id: an ID representing the class of information included in this descriptor.. The details meaning of each ID is supposed to be defined in platform standards based on the present document.

profile_extension: a profile extension. The meaning of each profile extension is defined by each platform standard based on the present document. The length of the profile_extension can be deduced by subtracting 2 bytes corresponding to profile_extension_class_id from length.

12.5.6.8 Application expiration descriptor

An application expiration descriptor designates a time point from which an application becomes invalid because it is expired. This descriptor is placed in unsigned_descriptor_loop of application_definition().

Table 21: Application expiration descriptor

Syntax	No. of Bits	Mnemonic
application_expiration_descriptor(){		
Tag	16	uimsbf
Length	16	uimsbf
Expiration	64	uimsbf
}		

tag: 0x0008

length: length of the content of this descriptor following this field.

expiration: the time when the corresponding application expires represented with the difference between January 1st, 2000 UTC and the time in seconds. Passing this point of time, the corresponding application is no longer valid, meaning it may be removed.

12.5.7 Format of module information message

The format of the module information message is defined in table 22.

Table 22: Format of module information message

Syntax	No. of Bits	Mnemonic
module_information_message(){		
message_version	32	uimsbf
module_count	16	uimsbf
for(I = 0;i<module_count;i++){		
module_locator()		
}		
common_descriptor_loop		descriptor_loop
}		

message_version: version of the message.

module_count: number of application modules listed in this message.

module_locator: represents a location and time interval, where and when an application module can be downloaded. This structure is defined in table 23.

common_descriptor_loop: descriptors to be applied to all application modules in this message.

Table 23: Module locator

Syntax	No. of Bits	Mnemonic
module_locator(){		
module_id		utf8
module_version	32	uimsbf
module_size	32	uimsbf
locator_version	32	uimsbf
url_count	8	uimsbf
for(i=0;i<url_count;i++){		
url		utf8
schedule_count	8	uimsbf
for(j = 0;j<schedule_count;j++){		
start_time	64	uimsbf
Duration	32	uimsbf
}		
url_descriptor_loop		descriptor_loop
}		
locator_descriptor_loop		descriptor_loop
}		

module_id: ID of the application module.

module_version: version of the application module. A larger value means a newer version.

module_size: size of the application module in bytes. This value must coincide with the size of the actual application module.

locator_version: ***version of the whole*** module_locator.

url_count: number of URLs locating the application module.

url: a URL locating the application module. The types of URLs supported are different by the underlying network.

schedule_count: number of time intervals during which the application module may be downloaded from the associated URL. 0 in this filed means that the application module is always downloadable.

start_time: the start of the time interval where the application module can be downloaded from the corresponding URL. It is the difference between the start time and January 1st, 2000 UTC in seconds.

duration: the length of the time interval in seconds from the start time during which the application module can be downloaded from an associated URL.

url_descriptor_loop: descriptors for additional information on the URL.

locator_descriptor_loop: descriptors for additional information on this module_locator.

12.5.8 Format of service binding message

The format of the service binding message is defined in table 24.

Table 24: Format of service binding message

Syntax	No. of Bits	Mnemonic
service_binding_message(){		
message_version	32	uimsbf
service_count	8	uimsbf
for (i=0;i<service_count;i++){		
service_locator()		
bound_app_count	8	uimsbf
for(j=0;j<bound_app_count;j++){		
binding_id	32	uimsbf
Start	1	
Kill	1	
event_bound	1	
Reserved	5	"11111"
if (event_bound=="1") {		
binding_tag	8	uimsbf
}		
application_id		utf8
app_min_version	32	uimsbf
binding_descriptor_loop		descriptor_loop
}		
service_descriptor_loop		descriptor_loop
}		
common_descriptor_loop		descriptor_loop
}		

message_version: version of this message.

service_count: number of services.

service_locator(): locator pointing to a service. The format of the locator is defined by each platform standard.

bound_app_count: number of applications bound to a service.

binding_id: an ID identifying an association between a service and an application. If there is any change in the specifics of a binding, so must the ID, and it shall be unique within the context of a service binding message.

start: if set to 1, the corresponding application shall be started when the event_bound bit is set, and the corresponding service is selected, or when the event_bound bit is cleared, the service is selected, and the corresponding binding_tag appears in the application control message. If cleared to 0, the application shall not be started in any of the above conditions. The reason for clearing this bit is mainly for cases where an application is not intended to be started from a service, but it should not terminate when the service is selected after the application is started from another service.

*kill**: if set to 1, the corresponding application is terminated unconditionally***. If cleared, the application is allowed to terminate voluntarily. More specifically, in the case of MIDlet, if this bit is 1, destroyApp method is passed a true. If it is 0, a false is passed to the method.

event_bound: if this bit is set to 1, the corresponding application is launched, when the associated service is selected, and at the same time, the binding_tag appears in the corresponding application control message. Once launched, it is terminated when the binding_tag disappears in the application control message.

binding_tag: a tag for designating a specific binding in the corresponding application control message. Within a service binding message, this tag shall be unique.

application_id: ID of an application to be associated with the associated service.

app_min_version: minimum version of the application bound to the service.

binding_descriptor_loop: descriptors describing additional information specific to each binding between a service and an application.

service_descriptor_loop: descriptors describing additional information on each service.

common_descriptor_loop: ***descriptors to be applied to the whole service binding message.***

12.5.9 Format of application control message

The content that must be conveyed within an application control message is as follows. The concrete format for the application control message is supposed to be defined in each platform standard.

service_binding_message_version: designates the version of the service binding message, to which the application control message refers for the binding_tags within it. Only when the version of the service binding message stored in the receiver is equal to or newer than this version, the binding_tags within the application control message is valid with respect to the stored service binding message.

binding_tag: an 8-bit unsigned integer which is a tag for a service-application binding. Within an application control message, multiple binding_tags may be included. Applications with their corresponding tags present in the message are automatically started, and they terminate when the tags disappear.

12.5.10 Format of certificate message

The format of the certificate message is defined in clause-9.4.1 of the present document.

13 Java environment

13.1 Introduction

The present document designates an environment consisting of a JVM and APIs, where Java applications may be executed. This clause-defines the Java environment required by the present document.

13.2 Requirements on Java environment

The present document requires MIDP 2.0 [1] at a minimum. But the present document is allowed to be implemented on Personal Basis Profile 1.1 [2], and therefore an implementation of the present document based on Java ME Personal Basis Profile 1.1 is perfectly legal.

13.3 DMB extensions

MATE is based on MIDP 2.0, but requires extensions to it. This clause-summarizes, in one place, the extensions required in addition to MIDP 2.0 or Java ME Personal Basis Profile to help understanding of the present document.

13.3.1 Standard optional packages

Among the standard Java APIs, MATE requires the following optional packages:

- FileConnection API, a part of JSR 75 PDA optional package [17] is required.
- A profile of JSR 135 Mobile Media API 1.1 [16] as designated in clause-13.8.1 is required.

13.3.2 Simultaneous execution of multiple applications

CLDC 1.1 and MIDP 2.0 specifications do not explicitly set any restriction on simultaneous execution of more than one application, but they are designed with an implicit assumption that an application at a time can be executed. Different from such an assumption, MATE requires simultaneous application of multiple applications.

13.3.3 Graphics extension

The LCD UI API defined by MIDP 2.0 lacks in the following two areas, thus requiring extensions. The extensions are defined in the dmb.ui package:

- A transparent graphics plane is overlaid atop a video plane and composed by the source over rule defined in Porter-Duff [19]. Therefore, it should be possible to designate alpha values and a composition rule for graphics operations. Composition rules of clear, source, and source over rules defined in Porter-Duff [19] should be supported.
- More than one application needs to display graphics on a screen at the same time. The current MIDP 2.0 specification permits only one application to be shown up on the screen at a time, so an extension is required to enable that functionality.

But the following options are available for the cases where strict conformance to the specification is too costly to implement:

- For graphics operations performed in source over mode, it is permitted to perform them in source mode with an exception of image drawing. Image drawing should always follow its original behaviour designated by MIDP 2.0.
- In all cases where alpha channel is involved, the minimum requirement is to respect fully-transparent pixels as they are.

13.4 Simultaneous execution of multiple applications

13.4.1 Requirements

It is recommended that MATE implementations are capable of simultaneously running as many applications as possible, but it is not required.

When an implementation can run only one application at a time, an application with the highest priority set in its application_definition (see clause-12.5.5) shall be chosen to run. But even when there is only one application running at a time, it is still required to use resource management framework since the underlying implementation may use it to arbitrate resources among it and other native entities such as native applications.

13.4.2 JVM implementation

An application is considered to run within a logically isolated JVM. But depending on Java ME configurations, the present document also permits an implementation running multiple applications at once in a JVM using class loaders. As a consequence, MATE implementations and applications should follow the following rules:

1) 1A MATE implementation may choose not to invoke finalizers on classes defined by an application upon termination of the application. JVM does not guarantee the execution of finalizers on exit. Therefore, applications must also not rely on it.

2) System classes and other classes shared among more than one application may be physically identical in the JVM level. This exception to the isolation among applications is to permit an implementation running more than one application within a JVM using the class loader mechanism. In this case, applications must not violate the following rules. And if they do, such applications may cause problems depending on MATE implementations:

   - Applications must not synchronize on system classes, other classes shared among applications, or objects shared among applications via static members.

   - Applications must not assume that any static member (variable or method) is shared with other application. As an example, it is not permitted to exchange data between applications using a static field.

13.5 Standard properties

The present document defines standard properties that can be read by applications. Some of them have different values per MIDlet, and some are receiver-wide. Such properties are used for applications to get information on themselves or on the receiver they are running on.

13.5.1 MIDlet properties

A MIDlet property may be obtained by invoking MIDlet.getAppProperty(String), and is information specific to an application. Property names beginning with "dmb." are reserved for use by the present document. Therefore, platform standards may not use a name beginning with "dmb.".

Table 25: Standard MIDlet Properties

Value	Description
dmb.app.id	The application ID of the MIDlet
dmb.app.dir	The path string representing the root directory of the file system the application owns (clause-13.17.2)

13.5.2 System properties

System properties may be retrieved using java.lang.System.getProperty(String), and provide information on the receiver. Property names beginning with "dmb." are reserved for use by the present document. Therefore, platform standards may not use a name beginning with "dmb.".

Table 26: Standard system properties

Value	Description
dmb.dev.id	The unique ID for identifying the receiver. The format of the ID is as follows: <domain_name>/<device_id> The above ID shall not include any character that violates RFC 2396 [14]. The meaning of each part of the ID is as follows: **domain_name**: the internet domain name owned by the manufacturer of the receiver. By using a domain name here, unique IDs may be assigned to manufacturers without a central registry of ID. **device_id**: ID for the receiver that is unique among the set of receivers from the receiver manufacturer. As far as it does not violate the URI format, any numeric and alphabetic character is allowed in this part.

13.6 Basic APIs

In MATE, to reduce the total size of the API set, and make it easily understandable, the following basic API and patterns are used.

13.6.1 AsyncResult/AsyncRequestor pattern

On a broadcast network, it is very common that the latency required for getting data is long and irregular, since such data are repeatedly retransmitted rather than directly accessible. Also, when controlling other applications from an application, the time to complete a change in the state of other application may vary depending on the operations of the controlled application.

Because of the nature of those operations, many APIs should be designed to work in an asynchronous way taking their result via separate listeners or callbacks. But asynchronous APIs are difficult to use, and in some cases, it is better to block threads invoking such APIs till they complete despite their long latency.

The pattern of AsyncResult/AsyncRequestor defined in dmb.util package enables an API to act in both asynchronous and synchronous ways, and widely used throughout MATE APIs.

A method following the pattern has, in general, the following form:

```
AsyncResult doSomething(int arg1, int arg2, AsyncRequestor requestor);
```

Such a method just requests an operation, and returns immediately without waiting for its completion. The progress of the operation triggered by the method call may be monitored using the returned AsyncResult object (polling), or a certain method may be invoked on the AsyncResult to wait for the result (synchronous call). Also, if permitted, such an operation may be cancelled.

In addition to polling or waiting for the completion, if the progress of the operation needs to be notified of via a different thread, an AsyncRequestor should be designated. resultUpdated(AsyncResult) method of AsyncRequestor is invoked every time there is some progress in the operation. If such notifications are not required, it is always possible to pass a null to the method triggering an operation.

Regardless of how to get to know the completion of an operation, get() method of an AsyncResult may be invoked to retrieve the result of the operation, and in case there is any exception thrown in the course of the operation, the exception is re-thrown when the get() is invoked.

Depending on APIs, the progress of an operation is reported through AsyncResult.getProgress()method as an integer between 0 and 100 inclusive. If there is no such information available, the method returns AsyncResult.PROGRESS_UNKNOWN instead. Therefore, within AsyncRequestor.resultUpdated(AsyncResult), AsyncResult.isDone() must be consulted to know whether the operation is completed or not.

13.6.2 AttributedObject pattern

AttributedObject defined in dmb.util package is to publish named data of various types to applications. The information conveyed via a broadcast network is of various types, and varying platform by platform. Instead of defining separate classes to retrieve such information per platform, MATE utilizes a more generic interface of AttributedObject throughout many parts of it.

An attribute is represented with a string, and its value may be of such types as java.lang.Object, java.lang.String, boolean, int, long, java.util.Date, byte[], and java.lang.Object[]. An API based on the AttributeObject pattern may define additional types as needed.

If an AttributedObject is asked to return a value of non-existing attribute, dmb.util.InvalidAttributeException is thrown. isValid(String) method may be used to query if an attribute exists prior to trying to retrieve its value, and getAttributes()method may be used to get a list of all attributes provided by an AttributedObject.

13.7 Graphic user interface API

The present document defines APIs in dmb.ui package by extending certain classes in javax.microedition.lcdui package of MIDP 2.0. While it follows the basic framework of MIDP 2.0, additional rules and APIs for managing multiple applications sharing a display are defined. DMBCanvas extends Canvas which is defined by MIDP 2.0, and is the basis for user interfaces of DMB applications. DisplayControl augments Display to provide additional means to manage the screen. AlphaAttribute is associated with each Graphics to provide functionalities such as specifying alpha values and composite operations. An abstract class, DMBItem, is to aid implementation of user interfaces, and TextItem extending DMBItem, is to provide an ability to get text inputs from users. FontLoader is for loading fonts dynamically into the system, and KeyLock is used to reserve some keys for exclusive use by an application.

13.7.1 Screen management

Different from Canvas that exclusively occupies the whole screen, DMBCanvas may use a certain portion or the whole area of a screen by designating a screen area to cover. By using DMBCanvases, more than one application may access and draw on a display device at the same time, and screen areas used by applications may overlap with one another. Each application is assigned a Z-order value, which represents how the application (or an active DMBCanvas) is close to the user, and when a screen is updated, applications are painted in the order of increasing Z-order.

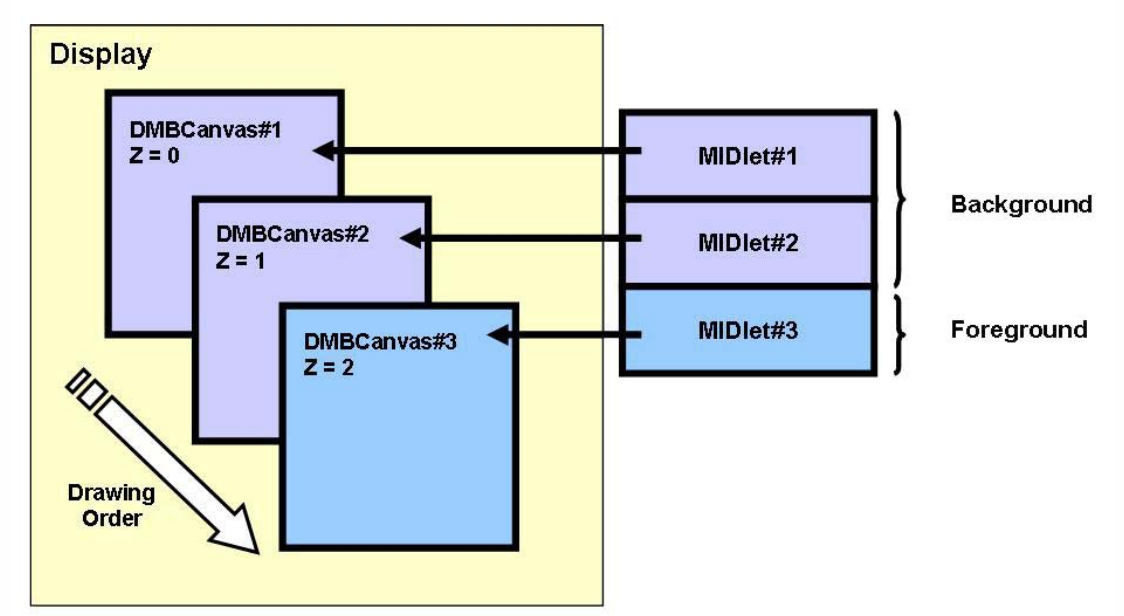


Figure 7: Management of z order

As in figure 7, an application with the largest z-order is drawn topmost, and as the foreground application, has the key focus. Except for the cases of using KeyLock as described below, basically all the key events generated are delivered to the DMBCanvas of the application owning the key focus. Pointer events, on the other hand, are delivered to a DMBCanvas that contains the position where the pressed event is generated and has the largest z-order. And the following drag and release events are delivered also to the same DMBCanvas. If a pressed event is generated at the point where no DMBCanvas covers, the events following it shall not be delivered to any DMBCanvas, and just ignored.

When an application needs to change its z-order, setPriority(int priority) of the DisplayControl can be invoked to set a priority. The possible priorities are HIGH, NORMAL, and LOW. An application with a higher priority shall be drawn over the application with a lower priority.

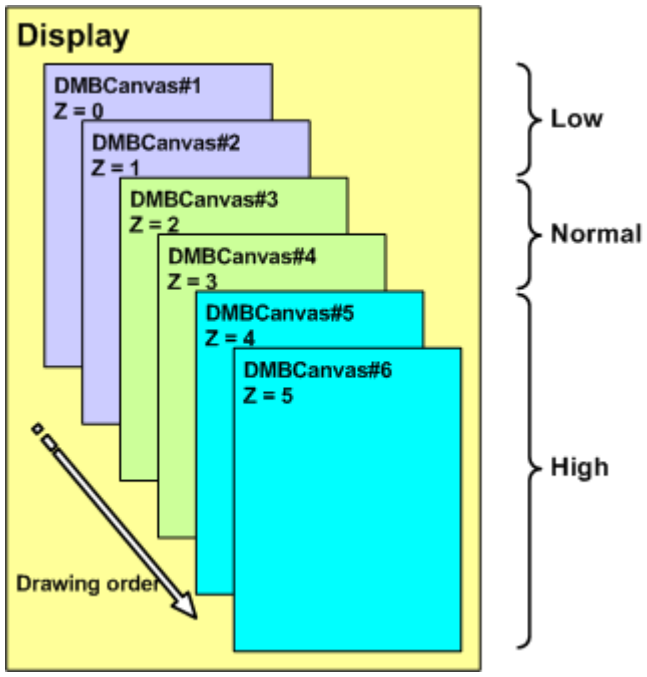


Figure 8: Priority in z order

Figure 8 shows the priorities and the z-orders assigned to DMBCanvases each of which is activated in an application owning it. When it is necessary to change z-orders of applications with a single priority, toFront()and toBack() methods of DisplayControl may be invoked, respectively resulting in sending an application to the bottom and bringing it to the front among those with the same priority.

To change the current owner of the key focus, an application may change its z-order resulting in getting the key focus or yielding it to other application. For example, if an application needs to display an important warning and get some input from the user, it may do so by setting its priority to HIGH, and calling toFront(). Then it may bring it to the topmost position among those with the HIGH priority. If an application does not require any key input, DisplayControl.setFocusable(false) may be invoked not to accept the key focus. Depending on implementations, a separate manager coordinating z-orders may exist to set the z-order of applications from outside the applications. But the present document only defines a means to change the z-order of an application from itself. Thus the operations of such an external manager are not restricted, and may coexist with an implementation of this API.

13.7.2 Processing alpha values

MIDP 2.0 only applies the source over rule defined in Porter-Duff [19], and assumes that the drawing surfaces do not an alpha channel. On the other hand, DMB applications require more advanced processing of alpha values to enable composition among them and between the graphics and the video planes. AlphaAttribute class is to enable setting of an alpha composite rule among CLEAR, SRC, and SRC_OVER when a drawing surface has an alpha channel. CLEAR rule is to clear the colour and the alpha value of the destination to 0, SRC to set both the colour and the alpha value of the destination as the current values, and SRC_OVER is to composite the source and the destination considering their alpha values to determine the final value in the destination. setComposite(int compRule) method may be invoked to set the composite rule. Once set, the rule is applied to all of drawing operations on the relevant Graphics object. Note that the original alpha processing in MIDP 2.0 is identical to the case where the composite rule is set to SRC_OVER and the current alpha value is set fully opaque.

13.7.3 User interface elements

DMBItem is a user interface element that is similar to an Item managed by Form class, and may be added to a DMBCanvas. A user interface element may be implemented by extending the class. A DMBItem, much like DMBCanvas, designates a screen area it manages, and receives key, pointer, and paint events from the DMBCanvas containing it. If more than one DMBItem is used within a DMBCanvas, it is assigned a z-order value as it is added to the containing DMBCanvas, and one of the DMBItems owns the key focus resulting in receiving the key events from the containing DMBCanvas. Each DMBItem has methods invoked when there are key/pointer/paint events, it has gained or lost the key focus, and it has been added to or removed from the containing DMBCanvas. DMBCanvas maintains methods for adding and removing DMBItems, and changing the focus.

Different from a Displayable in MIDP 2.0 that occupies the whole screen, DMBCanvases and DMBItems occupy a certain portion of a screen by nature, and have a local coordinate, the origin of which is their top-left corner.

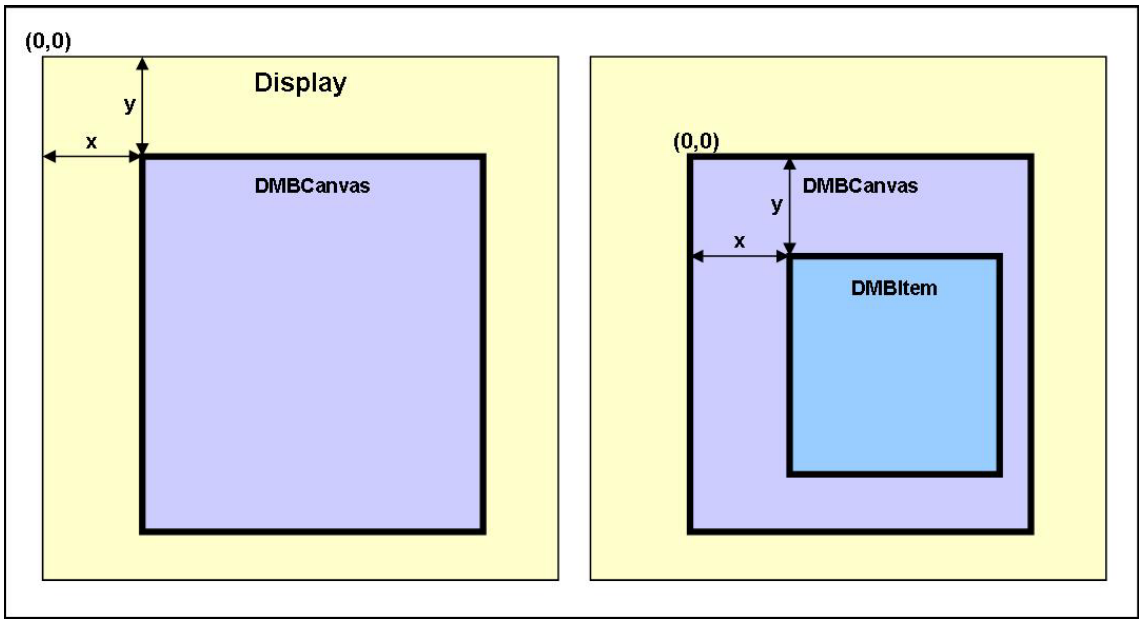


Figure 9: Coordinate systems of DMBCanvas and DMBItem

When key/pointer/paint events are delivered, any coordinate associated with them is described in the local coordinate. On the other hand, when positioning a DMBCanvas within a display device, or DMBItem within its containing DMBCanvas using setBounds(int,int,int,int), the local coordinate system of its parent entity is used.

13.7.4 Key mapping

DMB receivers may have certain keys dedicated to DMB watching and applications. To support such keys, DMBCanvas defines DMBAction similar to GameAction in Canvas. Each key may be mapped to a DMBAction. getDMBAction(int) method can be used to retrieve a DMBAction value corresponding a key, and getDMBKeyCode(int) to get a key code mapped to a given DMBAction. As in the case of GameAction, more than one key code can be mapped to a DMBAction, the following equation may not evaluate to true:

```
keyCode == getDMBKeyCode(getDMBAction(keyCode))
```

The present document defines DMBActions of VOLUME_UP, VOLUME_DOWN, MUTE, CHANNEL_UP, CHANNEL_DOWN, RECORD, GUIDE, and INFO.

13.7.5 Reserving keys for exclusive use

Within a broadcast receiver, it is common that an application, without any GUI component being presented on the screen, needs to get key inputs, or reserve certain hot keys for its exclusive use. For example, an EPG application may reserve the EPG in the receiver for its exclusive use, and react to the key by showing up in the screen.

To support this kind of scenarios, MATE provides KeyLock API. An application may create a KeyLock object, and acquire its ownership via the resource manager (clause-13.16), to reserve the keys of the specified codes for exclusive use by it. Key events corresponding to the key codes may directly be processed by subclassing KeyLock class, or may be designated to be delivered to the application in the same way as in a focused application.

13.7.6 Loading fonts dynamically

FontLoader class can be used to load and create fonts, that were absent in the receiver, from the data either received from a broadcast or return channel, or retrieved from the application resources. Dynamic creation of fonts is an optional feature, and the present document does not designate any specific format for such fonts.

13.8 Media control API

MATE utilizes Java ME MMAPI 1.1 [16] for playback of audio and video clips, and controling presentation of broadcast audio and video. This clause-defines parts of MMAPI elements that must be supported in MATE, and their relationship with other APIs such as those for service selection.

13.8.1 A MMAPI 1.1 profile

MATE requires all the controls designated with "SHOULD" and "MUST" in the "Sampled Audio" and "Video" parts of MMAPI 1.1 according to the rules defined in "Optionality and Implementation Requirements" section of the same specification. But in the case of a broadcast stream, StopTimeControl and FramePositioningControl need not be supported. And when presenting broadcast videos, dmb.media.BackgroundVideoControl must be supported in place of VideoControl. Optionally, RecordControl may be supported for broadcast videos.

13.8.2 Player creation

Players cannot be created via javax.microedition.media.Manager for the playback of broadcast streams. Such Players are created internally by dmb.service.ServiceManager, and accessible via getPlayer(String) and getPlayers() methods of the ServiceManager. For audio and video clips stored in storage devices, Players may directly be created via Manager.

13.9 Broadcast data access API

As defined in clause-7.1, MATE supports three types of broadcast channel protocols. In this clause, APIs for accessing files, packets, and triggers are defined respectively. Those APIs are defined in dmb.io package and utilizes the GCF(Generic Connection Framework) defined in Java ME MIDP 2.0 [1].

13.9.1 File access API

13.9.1.1 Creation of file objects

A BroadcastFileConnection object is used to access a broadcast file. It may be obtained by passing an appropriate locator to open(String) method of javax.microedition.io.Connector. Here, such a locator may locate a file within a broadcast file system, or the file system itself. In the latter case, the returned BroadcastFileConnection is a directory returning true from BroadcastFileConnection.isDirectory() method.

When a BroadcastFileConnection object is no longer required, it is a good practice to invoke close() on it. If so, any resource associated with the BroadcastFileConnection may be reclaimed as soon as possible.

13.9.1.2 Directory

If a directory containing list of files is transported within a file system, a BroadcastFileConnection object corresponding to it may be obtained. With the BroadcastFileConnection object, list of files within the directory can be obtained, and a file or directory under the directory can be open by specifying a path relative to it.

13.9.1.3 Metadata

In general, a file or directory may entail various metadata such as its MIME type. Since the BroadcastFileConnection is a dmb.util.AttributedObject, such data may be attached to it as attributes. The types of metadata supported for files and directories differ from a platform to another.

13.9.1.4 File access

To read the content of a file, a java.io.InputStream should be obtained by calling openInputStream() method. When accessing a file system containing a file, the content of a file may already be loaded into memory, but otherwise, it may take arbitrary time to read data from the InputStream. In such a case, BroadcastFileSystem.load(AsyncRequestor) method may be used to get notified without waiting when the content of the corresponding file is fully loaded into memory. If load(AsyncRequestor) is invoked on multiple files, and the files are processed when their loading is completed, it is possible to increase the overall throughput by overlapping I/O and CPU-bound jobs.

13.9.1.5 File update

When a file is updated, an application can get notified of the fact by registering a BroadcastFileListener. Upon such a notification, an application can invoke BroadcastFileConnection.flush() and read the content of the new version.

13.9.2 Packet access API

To receive packets, javax.microedition.io.DatagramConnection API is used as it is. But since the target stream is a broadcast stream, packets can only be read from the stream.

Therefore among the methods of DatagramConnection, only getMaximumLength(), newDatagram(byte[], int), newDatagram(int), and receive(Datagram) are supported. Among other methods, newDatagram(byte[], int, String), newDatagram(int, String), and send(Datagram) result in an IOException, setAddress(Datagram), setAddress(String), and setData(byte[], int, int) cause a RuntimeException, and getNominalLength() always returns 0.

13.9.3 Trigger API

To receiver triggers, the same javax.microedition.io.DatagramConnection is used as in the case of receiving packets. The methods supported for receiving triggers are same as those supported for packet receiving. If a locator passed to Connector locates a trigger stream, the returned DatagramConnection shall create dmb.io.Triggers instead of Datagrams, and those trigger objects must be used to receive triggers.

Each trigger has an ID, and as far as there is no change in the content of a trigger assigned an ID, it is immediately passed to an application only once. In that case, the getState() method of the received Trigger object returns Trigger.RECEIVED, and the receiving application may prepare to perform an action designated by the trigger responding to the trigger.

After receiving a trigger, the receiver tracks the current time to see when the time designated by the trigger comes. At the designated time, the application is passed the same trigger, where the getState() method of the trigger returns Trigger.TRIGGERED, meaning the application should perform the designated action immediately.

If there is any discontinuity or drift in the AV clock, a receiver may fail to detect the time designated in a trigger, and needs to cancel the trigger. If this happens, the same trigger is delivered to the application once more, but this time, Trigger.CANCELED is returned from getState().

Note that more than one trigger with the same ID may be delivered to an application if there was a change in the designated time or the content of the trigger.

13.10 Service information API

13.10.1 Introduction

The service information refers to the information on channel configurations and/or program schedules in Ensembles. Applications like EPG require presenting such service information received by the receiver to the user. To provide a level of compatibilities among various differing broadcast network specifications, the present document defines basic elements available in the most of broadcast networks, and provides a means to query such information by specifying a condition to meet by the target information. Additionally the service information API utilizes dmb.util.AttributedObject pattern, to facilitate the addition of new concepts.

13.10.2 Service information object

MATE considers that the service information is composed of certain kinds of service information objects (hereafter, referred to as SI objects). Those objects refer to various entities described by the service information. Each platform standard defines SI objects in detail, and attributes are defined for each of them. The type of an SI object is represented by the value of its SIAttribute.TYPE attribute. A set of objects supported by a platform, and their attributes are defined in detail by a platform standard.

13.10.2.1 SI database

Though an SI database (dmb.si.SIDatabase) itself is not an SI object, it is an AttributedObject meaning it may also have its own attributes. The attributes associated with an SI database are usually applicable to all of SI objects.

13.10.3 SI query and view

A query may be submitted against the values of attributes of SI objects, and the result may be retrieved as a list of SI objects of the same type. Here, it is possible to get the result of the query as a view rather than a list of SI objects. Different from the list, a view is updated automatically based on the query used to create it when there is any change in the underlying SI information.

Once created, a view may be submitted another query, and a listener may be added to one to get notified of additions, removals, or updates occurring in the view.

13.11 Tuning API

A tuner is a device to read the content of an ensemble by tuning to it. MATE provides two classes in dmb.tuning package to control the tuner.

13.11.1 Tuner

dmb.tuning.Tuner object is for activating, deactivating, and getting the signal quality of a tuner in the receiver.

13.11.2 TunerLock

dmb.tuning.TunerLock is an abastract concept of sharing a Tuner. It permits applications tuning to an ensemble to share a tuner. A TunerLock is a resource, and acquiring the ownership of it means that the associated tuner is tuned to an ensemble designated by the TunerLock. Once acquired, the ownership to the TunerLock is retained unless another application acquires a TunerLock for the same tuner by specifying a higher priority, the tuning fails, or the signal is too weak to keep tuning to the ensemble.

13.12 Service selection API

For service selection MATE defines ServiceManager in dmb.service package. It provides a means to select a service, and to add and remove service components from it. Since a ServiceManager is a resource, it must be acquired via the resource manager (clause-13.16) before selecting a channel with it. For details, refer to the API documentation.

13.13 CAS API

MATE defines a set of APIs for purchasing paid contents, and controlling the process in dmb.ca package.

13.13.1 Communication with CA module

There are occasions where it is required to directly exchange messages with a CA module. Reading the smartcard number, or changing the PIN (Personal Identification Number) requires such a direct communication.

dmb.ca.CAModule provides a direct interface to a CA module. Once a CASession is created, with CAModule.openSession method, an application may send requests in the form of CARequest to the CAModule. Events or responses to CARequests from the CAModule are delivered to an application via CAEventReceiver that was passed to the openSession method. In this case, general events are represented with CAEvent, and on the other hand, responses to CARequest are represented with CAResponse, a subclass of CAEvent.

Each platform should define concrete subclasses of CARequest, CAEvent, and CAResponse for each CAS, since the detailed protocol between an application and a CA module differs by CAS. Therefore the present document does not define such concrete subclasses.

13.13.2 Purchasable entities

dmb.ca.Purchasable is intended to be implemented by an object representing a purchasable entity, and defines the methods for retrieving information on the entity, and other ones to be implemented by such an object. Any type of objects may implement this interface, but in most cases, SI objects representing services or program locations, which are purchasable, implement this interface.

To purchase an entity implementing this interface, purchase or openPurchaseSession methods may be invoked. When purchase method is invoked, the receiver implementation is responsible for managing the whole process of the purchasing the entity providing appropriate user interfaces. Thus applications such as EPG, which directly use CA API, do not need to do something while a purchase is in progress.

On the other hand, openPurchaseSession creates a CASession like CAModule.openSession. Thus it enables the direct control over the purchase procedure by permitting direct communication between a CA module and an application. As in the case of CAModule, concrete subclasses of CARequest, CAEvent, and CAResponse should be defined depending on platform standards and/or CASes.

13.14 Application control API

To enable control of other applications from an application, MATE defines a set of APIs in dmb.app package. First of all, AppControls for applications may be obtained via AppManager. An AppControl may be used to control the lifecycle, and installation and removal of the associated application.

AppControl represents an application of any type, which conforms to the application signalling and transport specification defined by the present document. A subclass called MIDletControl is defined to specifically support MIDlets.

13.15 Inter-application communication API

The present document defines a means to communicate among MIDlet, which is based on message queue. Message queue is easy to use, and may be used to synchronize multiple applications in various ways, in addition to allowing exchange of data.

13.15.1 Messages

A datum to be exchanged among applications is represented by an object implementing dmb.messaging.Message interface, and can be created with Port.newMessages. Similar to javax.microedition.io.Datagram object, this object provides a similar means to read, write, send, and receive data.

13.15.2 Port

For multiple applications to exchange messages, they must share a Port. Messages are sent to and received from a port.

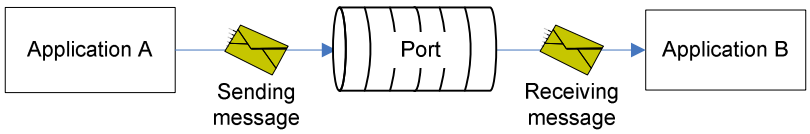


Figure 10: Concept of port

13.15.3 Sending messages

If a message is sent to a port (via send or sendFirst method of dmb.messaging.Port), the message is immediately stored in the port. Messages are processed by a port in a FIFO (First-In, First-Out) way, meaning that they are stored in their order of sending, and retrieved by receivers in the same order. But if sendFirst of Port class is used, a message may be put to the front of a port.

When creating a port, the maximum number of messages that can be stored within the port may be specified. It is also possible to set the maximum to 0. When there is no free space within a port, a message can be sent. In that case, depending on the setting of the blocking mode on the port, the sender may block until a space becomes available by other applications' retrieval of messages from the port, or return immediately without sending it.

13.15.4 Receiving messages

Messages stored in a port may be retrieved with receive method. Upon reception of a message, it is removed from the port, and copied to the buffer inside a message object of the receiving application.

If it is requested to receive a message when the port is empty, depending on the blocking mode setting, the application either waits for a message is available in the port, or immediately returns without a message.

13.16 Resource manager API

13.16.1 Introduction

Resources within a receiver such as video decoders may be used by an application at a time, and tuners may be shared among only applications that want to read the same ensemble. As such, there is a rule in sharing such resources of a receiver among multiple applications.

Also, there should be priorities among applications requiring the same resource. Otherwise, an application, which is important from user's standpoint, may stop its operation by losing its resources to other applications performing trivial tasks.

The resource manager API is for representing such rules for sharing resources in an appropriate way, and trading ownerships to the shared resources among the receiver implementation and applications, according to the priorities specified by applications.

13.16.2 Resource objects

In MATE, the objects implementing dmb.resources.Resource interface are considered as resources. Such resource objects include dmb.tuning.TunerLock and dmb.ui.KeyLock. A resource may be a hardware entity such as dmb.ca.CAModule, but it may also represent an abstract concept like dmb.tuning.TunerLock.

13.16.3 Resource group and choice

When an application needs resources, it is common to require more than one resource at the same time rather than a single resource at a time. For instance, let's say that an application reads data from an ensemble, does playback of an audio clip, and reserves a key for its exclusive use in a certain section of an application. In that case, it is of no use to acquire only parts of the resources required to perform all of the intended actions. MATE provides a means to represent and acquire the required resources as a unit. If any of the resources may not be acquired, the ownerships to all of the resources are given up to let other applications make use of them. The benefits gained by such a policy are as follows:

- *Increased utilization of resources within a receiver:* There are occasions where multiple applications need to acquire more than one resource to continue to proceed. In such cases, each application may acquire a resource but not others, and therefore no application can proceed. Or an application may lose the ownership to a resource that was already acquired while acquiring other resources. If these kinds of inefficiencies are avoided, the overall utilization of the receiver resources can be increased.

- *Simple implementation of applications and receivers:* Since an application does not need to coordinate multiple resources explicitly, requesting resources and coordinating their ownership become simple. Applications may delegate to the receiver implementation all the chores involved in the coordination of the resource acquisition, and as the result, their implementation is made simple. Also in the receiver side, the coordination of the resource acquisition may be made simpler than when dealing with each resource separately, by processing aggregated resources at once as a unit.

To group resources as a unit, two types of resource sets called resource group and resource choice are defined.

13.16.4 Resource group

A resource group represents a set of resources that must be acquired. If any resource within a resource group cannot be acquired, all the resources within the group are given up together. And even if the ownership to all the resources within a resource group was acquired, and later one of them becomes lost to other application, the ownership to all other resources are also given up together.

A resource group is represented by a dmb.resources.ResourceGroup object, and can group a set of resources.

13.16.5 Resource choice

A resource choice is also a collection of more than one resource like a resource group. But rather than being a unit to be acquired and given up together, a resource choice represents a requirement of acquiring any single resource among those in the resource choice.

For instance, when a receiver has multiple tuners, and an application is to acquire one of them, all the tuners may be grouped in a resource choice, and then acquired. Depending on the current condition of the receiver, the application can acquire either the ownership to a tuner or nothing.

A resource choice is represented by a dmb.resources.ResourceChoice object, and can group a set of resources, only one of which is required to be acquired.

13.16.6 Nesting resource groups and choices

Both resource group and choice are a set of resources, but they are also a resource themselves. Therefore, resource groups or choices include other groups and/or choices. By nesting groups and choices in such a way, complex requirements on the acquisition of resources can be specified.

13.16.7 Rule for determining resource ownership

To acquire ownership to a resource, acquire method of dmb.resources.ResourceManager should be invoked. And to release the ownership of a resource when an application finishes with it, release method of ResourceOwnership should be invoked.

Once a resource is requested, free resources are checked first, and if there is no free resource, resources owned by other applications are checked. When the resource is in use by other application, whether it can be taken over from the application or not should be determined. This decision is based on the priority set on each resource by each application.

That is, if an application specifies a higher priority for a resource than that specified by the current owner, then the resource can be taken over from the current owner. If there is more than one resource satisfying a request from an application, then one with lowest priority among them is picked.

The priority can be set when acquiring a resource, and for the resources that are already acquired, it can be changed with ResourceOwnership.setPriority(int). The method should frequently be invoked to reflect priorities varying according to the state of an application.

After checking that all the resources can be acquired, the original owners are given an opportunity to do cleanups for the resources. The opportunity is notified via ResourceOwner object specified when acquiring the resources. prepareRelease(ResourceOwnership) method is invoked on the object, but when it was inevitable to release the resources before invoking prepareRelease, or it took too much time within the method, then the resources may forcibly be revoked, and notifyRelease(ResourceOwnership) method be invoked instead.

13.17 Storage API

MATE requires the FileConnection API of JSR 75 PDA Optional Package [17] to support I/O with FLASH memory, disks, and other storage devices. The content stored in a storage device should persist even when there is no power supplied to the device.

13.17.1 Implementation requirements

MATE does not require a general purpose file system for the implementation of FileConnection API. For each application, only a directory must be accessible, and within the directory, only plain files shall be able to be created. Support for nested directories is not required.

13.17.2 Per-application storage

When an application is granted appropriate permissions, a directory is created for the application, for which the application has all the authorities. The root of the directory can be obtained by reading a MIDlet property named "dmb.app.dir". The directory can be accessed via FileConnection API.

When an application is removed from the receiver, the corresponding directory is also removed. Therefore, to protect application-specific data in such a case, it must be stored in a separate storage or server.

For an application to access a directory owned by other application, the owner needs to delegate appropriate permissions to the application. For details on authority delegation, refer to clause-9.3.5.

13.17.3 Permissions

In MATE, MIDP 2.0 style permissions defined in FileConnection API cannot be used as they are. Therefore, the following permissions are additionally defined:

- dmb.io.file.<operation>.<path>: a permission to perform the designated <operation> to the designated <path>. Here, <operation> is one of read, write, create, and remove, and <path> represents a path where the designated operation is permitted.

An application must request appropriate permissions to use storage. Also, the permission string defined above may be used in credentials.

13.18 Communication channel API

MATE supports all the APIs defined in javax.microedition.io package of Java ME MIDP 2.0 [1] to support use of communication channels. Note that all the APIs, defined in MIDP 2.0 [1] and relevant to communication channels, are also supported by Personal Basis Profile 1.1 [2].

Annex A (informative): Automated test environment for receiver certification

The purpose of this annex is to provide a recommended practice for the structure of an automated test environment, and a testing method. Since MATE and its base platform, MIDP 2.0, define all the elements required for building an automated test environment, no additional API is defined in the present document. And because an application implementing a set of tests may be downloaded to receivers, no separate communication protocol is also defined.

To certify an implementation of MATE to conform to the present document, an automated test environment and a receiver with the implementation should be connected, and the test cases for the certification should be run, where the automated test environment and the receiver should be able to exchange appropriate control signals and messages according to the progress of the test.

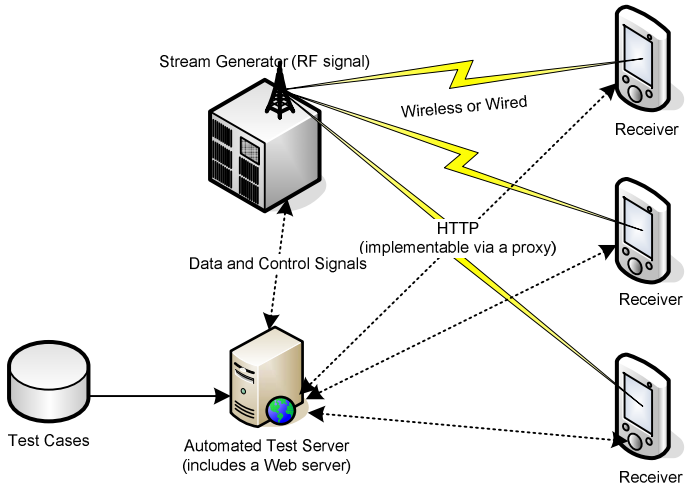


Figure A.1: A configuration of an automated test environment

Figure A.1 represents an automated test environment. Given a list of test cases, the automated test server performs each test, and logs the result. The automated test server controls the stream generator to create test programs and test data based on each test case, where the stream generator generates streams to be transmitted via broadcast interfaces.

The procedure by which each test is performed is as follows:

1) The automated test server reads a test case.

2) According to the test case, the followings are transferred to the stream generator: the test program to be run on the receiver, the communication module to be used by the program to communicate with the automated test server to proceed and log the test, and the associated test data. In this step, the test program is signaled to automatically launch:

   - The receiver runs the test program as soon as it is fully received.
   - While the test program runs, it communicates with the automated test server via HTTP using the communication module to get control commands, or sending out the test results. MATE is based on MIDP 2.0, which requires HTTP. That is why HTTP is used here.
   - Upon the completion of a test case, the automated test server logs the test result, and the test program on the receiver terminates.
   - Returns to 1) and repeats.

Figure A.2 depicts the interactions between a receiver and an automated test environment in more detail.

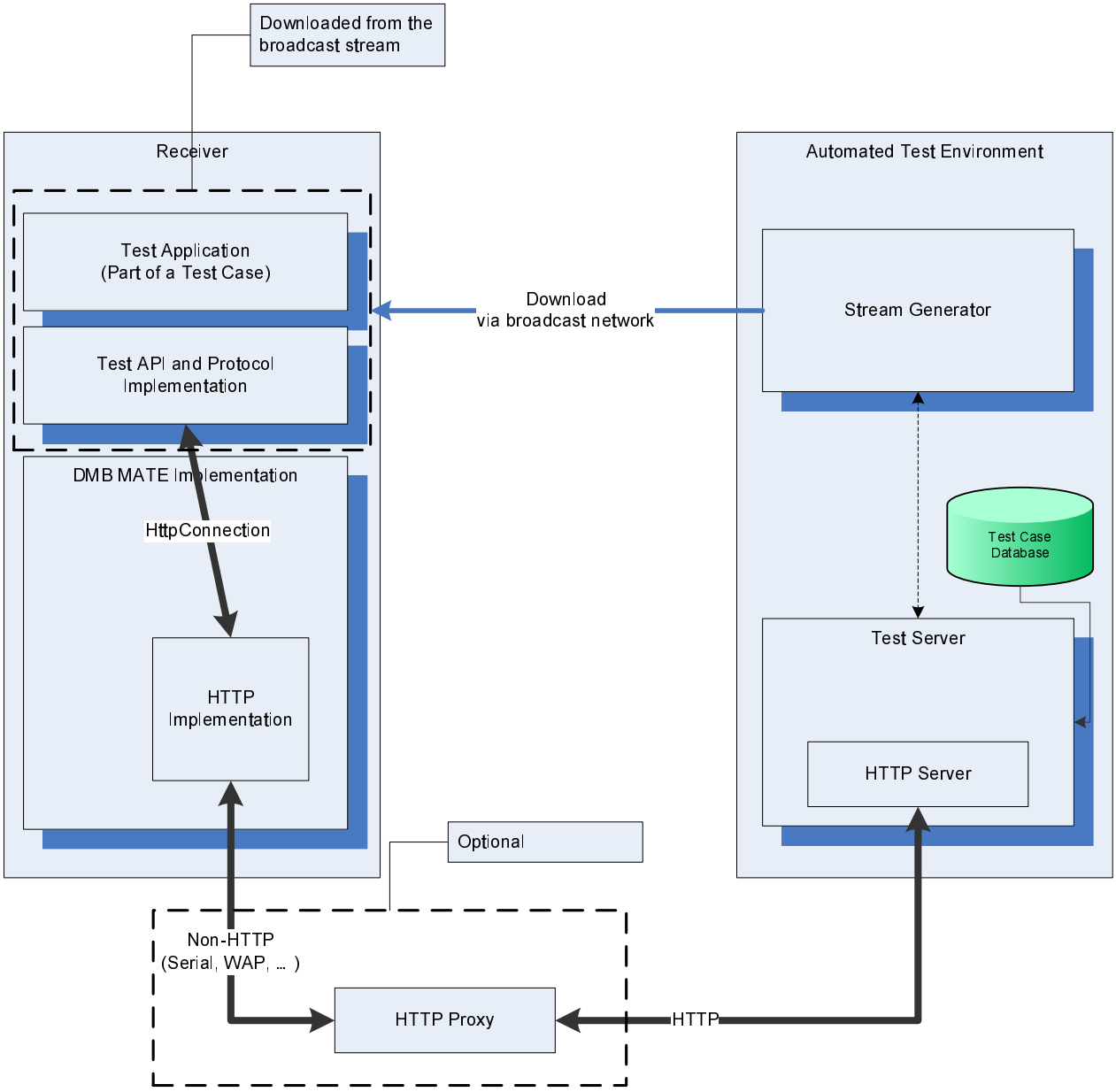


Figure A.2: Interactions within an automated test environment

Annex B (informative): Delivery and processing of key events among embedded applications and MATE

B.1 Introduction

All the types of applications defined in the present document (hereafter, referred to as MATE applications) can receive key inputs, and if a Java API, dmb.ui.KeyLock, is used, the corresponding application can intercept key inputs going to other applications.

But depending on receiver implementations, various applications may be embedded. For example, an application dealing with basic channel zapping may be implemented in C language, and embedded within a receiver. Such embedded applications need to get key inputs, and the issue of coordinating those applications with MATE applications in terms of their key inputs is raised here.

The purpose of this clause-is to provide a recommended practice in distributing and processing key events among both embedded and MATE applications, as a reference for the implementation of receivers conforming to the present document.

B.2 Key processing of embedded applications

Embedded applications should follow the following basic rules in processing key events:

- The embedded applications may exclusively use certain keys if they are not designated to be delivered to MATE applications by the platform standard. As an example, keys for activating receiver-specific menus or resetting the receiver may present. In this case, those keys are not delivered to MATE applications no matter whether one of the embedded applications has the key focus or not.
- The embedded applications may receive keys other than those mentioned in the right above, but if MATE applications reserve some of them using KeyLock (clause-13.7.5), the keys should be delivered to the corresponding MATE applications. A key to activate an EPG is a good example of such keys. But it is recommended to minimize the number of keys corresponding to this case. Also if it is possible to restrict use of the keys to the moments when one of the embedded applications has the key focus, then it is recommended to do so.
- With the exception of the above two cases, the embedded applications may process any key when they have the key focus.

B.3 Key focus management of MATE applications

When the embedded applications run simultaneously with MATE applications, the MATE applications are recommended to obey the following rules to make users comfortable:

- From the user's point of view, unless a MATE application is activated and considered to exclusively interact with the user, it should remove its dmb.ui.DMBCanvas by invoking removeDisplayable() of dmb.ui.DisplayControl, or avoid gaining the key focus by passing a false to setFocusable(boolean) of the same class. By doing so, the embedded applications may operate without unnecessary restrictions, when the MATE applications are not interacting with the user.
- As an exception to the case above, when a MATE application is not interacting with the user, and still needs to react to certain keys, such keys should be clearly specified using KeyLocks. By clearly specifying keys used by MATE applications, the embedded applications may get key inputs without unnessary restrictions if compared to the case when a MATE application has the key focus.

Annex C (informative): Accessing location information from Java applications

Information on the physical location of a receiver is quite useful enabling various location-based services. Though not mandating, the present document recommends use of JSR 176 [22] when such functionality is available on a specific receiver.

Presence of an implementation JSR 176 may be identified by examining a system property with a key "microedition.location.version."

Annex D (normative): API specification

Package Summary		Page
dmb.app	Defines classes and interfaces to get the list of applications known to receiver, and control each of them.	*59*
dmb.ca	Defines an interface to conditional access system in the receiver.	*82*
dmb.io	Defines APIs for accessing data coming from the broadcast channel.	*95*
dmb.media	As an extension to MMAPI, this package contains additional classes and interfaces.	*103*
dmb.messaging	Defines an API for applications to communicate with one another.	*109*
dmb.resources	Defines the basic framework to share resources among the receiver implementation and the aplications.	*118*
dmb.service	Defines APIs for service selection.	*131*
dmb.si	Provides APIs for giving access to service information managed by the underlying receiver implementation.	*159*
dmb.tuning	Defines a set of APIs for controlling tuners available in the receiver.	*183*
dmb.ui	This package provides a UI extension to `javax.microedition.lcdui` to handle the peculiarities in DMB environment such as support for transparent graphics plane.	*192*
dmb.util	Defines common interfaces and classes used in other packages.	*233*

Package dmb.app

Defines classes and interfaces to get the list of applications known to receiver, and control each of them.

See:

Description

Interface Summary		Page
AppControl	Represents an application known to the receiver.	59
AppListListener	An interface to be implemented by an object that needs to listen to changes in the application list maintained by `AppManager`.	72
AppStateListener	An interface to be implemented by an object that needs to listen to changes in the state of applications.	76
MIDletControl	Provides a means to control an `MIDlet` associated with an instance of this interface.	79

Class Summary		Page
AppManager	Manages applications known to the receiver.	73

Package dmb.app Description

Defines classes and interfaces to get the list of applications known to receiver, and control each of them. The list and each application may be monitored for changes. By using facilities defined in this package, an application may install, remove, execute, and terminate any application programatically.

`AppManager` may be used to obtain `AppControl`s for the applications of interest. In turn, `AppControl` provides a mean to control download and run states of the corresponding application.`AppControl` is the common base interface for all application models that are and will be designated in DMB MATE specification. So in the case of `javax.microedtion.midlet.MIDlet`, `MIDletControl` is defined as an extension to `dmb.app.Appcontrol` to deal with specifics of `MIDlet`. When there is another application model defined in the specification, then an additional subinterface of `AppControl` may be introduced in the future.

Interface AppControl

dmb.app

All Known Subinterfaces:

MIDletControl

---

public interface **AppControl**

Represents an application known to the receiver. Through this interface other applications may initiate download and control execution of the downloaded application. Later the application may be marked for removal through this interface.

Download States

Download of an application may be initiated by calling `download()`. It requests download of an application for either its initial installation or its upgrade. It does not protect the application from being removed. Thus the download may be canceled or, if it is not running, the application may be removed without prior notice.

The current download state may be obtained with `getDownloadState()`. Also changes in the download state are reported via any registered `AppStateListener` with DOWNLOAD_STATE_CHANGED event. When the event is delivered, the fourth argument to `AppStateListener.stateChanged(AppControl,String,int,String)` is set to the reason of the state change. The reason codes are declared in `AppStateListener`, and they all have `REASON_` prefix. This reason code is usually useful when the download state is changing from `DOWNLOADING` to `NOT_STORED` and from `UPDATING` to `STORED`. Such changes is the result of some error occurred while trying to download an application.

Run States

An application can be executed when its download state is either `STORED` or {@linK #UPDATING}. Here, note that a copy of an application is fully downloaded and ready when it is in `UPDATING` state. The update is in progress in the background without affecting the current version. When an application is directed to run in a download state other than those, an `IllegalStateException` is thrown. The current run state of an application may be queried with `getRunState()` method. Initially an application is in `NOT_RUNNING` state. With `init`, `start`, and `pause` methods, the run state of an application may be controlled. When a call to those methods implies multiple state transitions and the whole transitions are not completed, then the call is considered canceled. Thus a call to `AsyncResult.isCanceled()` returns `true` in that case.

When there is a change in the run state, a RUN_STATE_CHANGED event is delivered to the registered listeners. In this case, a relevant reason code is passed to the listener as the fourth argument to `AppStateListener.stateChanged(AppControl,String,int,String)`. For reason codes, refer to constants beginning with `REASON_` in `AppStateListener`.

Montoring Download Progress

The download progress of an application may be obtained via `getProgress()`. It reports progress with an integer ranging from 0 to 100, where 0 means nothing is downloaded and 100 fully downloaded. When there is some progress it is reported to any registered listeners via `AppStateListener` with DOWNLOAD_PROGRESSED event.

When an Application is no longer signaled

The receiver implementation shall go through the following process, when it notices that an application is no longer signaled:

- If the current download state is `DOWNLOADING`, the download is canceled automatically.
- If it is either the above mentioned case or one where the application's download state is in `NOT_STORED` state, the `AppControl` instance for the application becomes `INVALID` and the registered listeners are notified. Once it is in `INVALID` state, any method incurs a state change throws `IllegalStateException`. Normally an application may be controlled with prior knowledge on the state of it. Thus in most cases, such `IllegalStateException`s do not matter. But any general application manager application must process `IllegalStateException` since it deals with any application it does not have any knowledge on. Other methods in `AppControl` that are not marked to throw `IllegalStateException` must return their prior value, since such behaviour simplifies implementation of applications displaying information on applications in such a exceptional case.

Permissions

Permissions affecting operatons of `AppControl` are as follows:

- `dmb.app.control.<app_id>`: permission to control the download and the run states of an application through methods in `AppControl`. This is required to invoke `download()`, `remove()`,

init(AsyncRequestor), start(AsyncRequestor), pause(AsyncRequestor), stop(boolean, AsyncRequestor), and switchTo().

Field Summary		Page
int	**DOWNLOADING** Represents a download state where an application is being downloaded, or updated while its previous version is already invalidated.	63
int	**INVALID** Represents a download state where this AppControl instance is invalid, since the corresponding application is disappeared while it is not stored.	63
int	**NOT_RUNNING** Represents a run state where an application is not running.	64
int	**NOT_STORED** Represents a download state where data constituting an application is not stored in the receiver side.	63
int	**OUTDATED** Represents a download state where an application is fully downloaded, but a newer version is known to be available.	64
int	**PAUSED** Represents a run state where an application is loaded and initialized.	64
int	**STARTED** Represents a run state where an application is activated and doing what it is designated to do.	64
int	**STORED** Represents a download state where an application is fully downloaded and ready to run.	64
int	**UPDATING** Represents a download state where an application is being updated while the current version is still valid and can be run.	63

Method Summary		Page
void	**addAppStateListener**(AppStateListener listener) Adds an AppStateListener to monitor changes in the state of this application.	69
void	**download**() Requests download of this application represented by this AppControl.	66
String	**getDescription**() Returns the description of this application.	71
String	**getDescription**(String lang) Returns the description of this application in the designated language.	72
String[]	**getDescriptions**() Returns all the descriptions of this application represented in different languages together with the corresponding language code.	72

int	**getDownloadState**() Returns the current download state of this application.	*65*
AsyncResult	**getIcon**(AsyncRequestor requestor) Gets the icon associated with this application.	*71*
String	**getID**() Returns the ID of this application.	*65*
String	**getName**() Returns the name of this application.	*70*
String	**getName**(String lang) Returns the name of this application represented in the given language.	*70*
String[]	**getNames**() Returns a list of the application names in all languages available in the signalling message.	*71*
int	**getProgress**() Returns the current progress in downloading this application as an int ranging from 0 to 100.	*66*
int	**getRunState**() Returns the current run state of this application.	*66*
int	**getVersion**() Returns the version of this application.	*65*
AsyncResult	**init**(AsyncRequestor r) Initializes this application.	*67*
boolean	**isAutoDownload**() Returns whether this application represented by this AppControl is to be automatically downloaded or not.	*66*
boolean	**isVisible**() Returns whether this application should be visible to users via UI or so.	*70*
AsyncResult	**pause**(AsyncRequestor r) Pauses this application.	*68*
void	**remove**() Designates this application should be removed.	*67*
void	**removeAppStateListener**(AppStateListener listener) Removes the designated AppStateListener from this application.	*70*
AsyncResult	**start**(AsyncRequestor r) Starts this application.	*68*
AsyncResult	**stop**(boolean forced, AsyncRequestor r) Stops this application.	*69*
void	**switchTo**() Switches the calling application to an application represented by this AppControl.	*69*

Field Detail

INVALID

```
public static final int INVALID = -1
```

Represents a download state where this `AppControl` instance is invalid, since the corresponding application is disappeared while it is not stored. All the methods for changing the download and the run states throw `IllegalStateException` when an `AppControl` is in this state.

NOT_STORED

```
public static final int NOT_STORED = 0
```

Represents a download state where data constituting an application is not stored in the receiver side. In this state, the application is known to the receiver through the application signalling mechanism, but either its download is not initiated or it was downloaded and stored, but later removed. If download of the application is requested, then the download state changes to `DOWNLOADING` and the receiver begins to download data constituting the application. In this state, `getProgress()` returns 0.

See Also:

`getDownloadState()`

DOWNLOADING

```
public static final int DOWNLOADING = 1
```

Represents a download state where an application is being downloaded, or updated while its previous version is already invalidated. When the download completes, the download state is automatically changed to `STORED` state. In this state, applications can be run, and `getProgress()` method returns the current download progress.

See Also:

`getDownloadState()`

UPDATING

```
public static final int UPDATING = 2
```

Represents a download state where an application is being updated while the current version is still valid and can be run. This state may be entered while an application is previously in `OUTDATED` state. Note that the application can be run in this state, and `getProgress()` reflects the current progress of the update.

See Also:

`getDownloadState()`

STORED

```
public static final int STORED = 3
```

Represents a download state where an application is fully downloaded and ready to run. In this state, getProgress() always returns 100.

See Also:

getDownloadState()

OUTDATED

```
public static final int OUTDATED = 4
```

Represents a download state where an application is fully downloaded, but a newer version is known to be available. In this state, getProgress() always returns 100.

NOT_RUNNING

```
public static final int NOT_RUNNING = 0
```

Represents a run state where an application is not running.

See Also:

getRunState()

PAUSED

```
public static final int PAUSED = 1
```

Represents a run state where an application is loaded and initialized. When an application is started and then paused, it also enters this state. In this state, an application should occupy receiver resources as less as possible, but be ready to start quickly.

See Also:

getRunState()

STARTED

```
public static final int STARTED = 2
```

Represents a run state where an application is activated and doing what it is designated to do.

See Also:

getRunState()

Method Detail

getID

```
public String getID()
```

Returns the ID of this application.

Returns:

application ID

---

getVersion

```
public int getVersion()
```

Returns the version of this application. The version is meaningful only when it is in STORED state. When it is not yet stored, this returns -1.

Returns:

if this application is in STORED state, then returns its version. Otherwise, returns -1.

---

getDownloadState

```
public int getDownloadState()
```

Returns the current download state of this application.

Returns:

the current download state. One of the following:

- NOT_STORED
- DOWNLOADING
- UPDATING
- STORED
- OUTDATED
- INVALID

---

getRunState

```
public int getRunState()
```

Returns the current run state of this application.

Returns:

the current run state. One of the following:

- NOT_RUNNING
- PAUSED
- STARTED

---

getProgress

```
public int getProgress()
```

Returns the current progress in downloading this application as an `int` ranging from 0 to 100. Here, 0 means nothing is downloaded yet, and 100 fully downloaded. Note that this returns a number between 0 and 100 only when this application is in either DOWNLOADING or UPDATING state.

Returns:

the current progress in downloading this application

---

isAutoDownload

```
public boolean isAutoDownload()
```

Returns whether this application represented by this `AppControl` is to be automatically downloaded or not.

Returns:

`true` if it is to be automatically downloaded, `false` otherwise.

---

download

```
public void download()
              throws SecurityException,
                     IllegalStateException
```

Requests download of this application represented by this `AppControl`. When this application is in the NOT_STORED or OUTDATED state and this method is called, the application is entered to the DOWNLOADING or UPDATING state, respectively. When the download is completed, it automatically moves into the STORED state. Such state changes are reported through registered AppStateListeners.

Throws:

`SecurityException` - When caller does not have a permission to control the state of this application

`IllegalStateException` - When this application is in `INVALID` state

---

remove

```
public void remove()
            throws SecurityException,
                   IllegalStateException
```

Designates this application should be removed. The call to this method changes the state of this application to `NOT_STORED`. If the application is no longer signaled too, then its state is automatically changed to `INVALID` state.

Throws:

`SecurityException` - When caller does not have a permission to control the state of this application

`IllegalStateException` - When this application is in `INVALID` state

---

init

```
public AsyncResult init(AsyncRequestor r)
                throws SecurityException,
                       IllegalStateException
```

Initializes this application. It enters `PAUSED` state after a sucessful initialization. Otherwise it remains in `NOT_RUNNING` state. This method is invoked when this application is not in `STORED` or `UPDATING` state, an `IllegalStateException` is thrown. Calling this method is meaningful only when it is in `NOT_RUNNING` state, but it is legal to call it in other states and the call is silently ignored in that case. This method returns immediately without blocking. The actual progress may be obtained via `AsyncRequestor` and `AsyncResult`.

Parameters:

`r` - `AsyncRequestor` to get the progress. Specify `null` if no report is required

Returns:

`AsyncResult` object to query the current progress. `AsyncResult.get()` returns `null` if this application was successfully initialized. Otherwise, it will throw an exception representing the cause of the failure. In the case of `MIDlet`, it rethrows an exception thrown by `MIDlet` in the course of its initialization

Throws:

`SecurityException` - the calling application does not have proper permission to call this method

`IllegalStateException` - this application was in `INVALID`, `NOT_STORED`, or `DOWNLOADING`

---

start

```
public AsyncResult start(AsyncRequestor r)
                  throws SecurityException,
                         IllegalStateException
```

Starts this application. As the result, it enters STARTED state. If it is in NOT_RUNNING before calling this method, the receiver behaves as if init(AsyncRequestor) was first called and completed. As in the case of calling init, this method must be called in STORED or UPDATING state. Otherwise, an IllegalStateException is thrown. In terms of the run state, this method is meaningful only in NOT_RUNNING or PAUSED state. But it is legal to call this method in other states, and such attempt will silently be ignored. If starting is not successful, it remains in PAUSED state. Note that this method returns immediately without blocking, so AsyncRequestor and AsyncResult should be used to track the progress of the operation.

Parameters:

r - AsyncRequestor to get the progress. Specify null if no report is required

Returns:

AsyncResult object to query the current progress. AsyncResult.get() returns null if this application was successfully started. Otherwise, it will throw an exception representing the cause of the failure. In the case of MIDlet, it rethrows an exception thrown by MIDlet while trying to start it

Throws:

SecurityException - the calling application does not have proper permission to call this method

IllegalStateException - this application was in INVALID, NOT_STORED, or DOWNLOADING

---

pause

```
public AsyncResult pause(AsyncRequestor r)
                  throws SecurityException,
                         IllegalStateException
```

Pauses this application. As the result, the run state of this application is changed to PAUSED state upon success. This method may be invoked only when this application is in STARTED state, but doing so in other state does not incur any error and is silently ignored. Note that this method returns immediately without blocking. The actual progress can be monitored with AsyncRequestor and AsyncResult.

Parameters:

r - AsyncRequestor to get the progress. Specify null if no report is required

Returns:

AsyncResult object to query the current progress. AsyncResult.get() returns null if this application was successfully paused. Otherwise, it will throw an exception representing the cause of the failure. In the case of MIDlet, it rethrows an exception thrown by MIDlet while trying to pause it

Throws:

SecurityException - the calling application does not have proper permission to call this method

IllegalStateException - this application was in INVALID, NOT_STORED, or DOWNLOADING

---

stop

```
public AsyncResult stop(boolean forced,
                        AsyncRequestor r)
                 throws SecurityException,
                        IllegalStateException
```

Stops this application. Upon successful termination, this application is entered STARTED state. This method is meaningful in STARTED or PAUSED state. But calling it in other states is legal and silently ignored without throwing an exception. Note that this method returns immediately without blocking. To monitor the progress of this operation, AsyncRequestor and AsyncResult should be used appropriately.

Parameters:

forced - if true, this application should be terminated unconditionally. Otherwise, this application have a choice of vetoing the termination request

r - AsyncRequestor to get the progress. Specify null if no report is required

Returns:

AsyncResult object to query the current progress. AsyncResult.get() returns null if this application was successfully stopped. Otherwise, it will throw an exception representing the cause of the failure. In the case of MIDlet, it rethrows an exception thrown by MIDlet while trying to stop it

Throws:

SecurityException - the calling application does not have proper permission to call this method

IllegalStateException - this application was in INVALID, NOT_STORED, or DOWNLOADING

---

switchTo

```
public void switchTo()
```

Switches the calling application to an application represented by this AppControl. This means that the calling application is destroyed first, and then the designated application is launched. Note that failing to launch the second application does not cause relaunching of the first application.

This method is useful in a resource constrainted environment, since the two applications are never active at the same time.

Throws:

IllegalStateException - if an application represented by this AppControl is in the INVALID, NOT_STORED, or DOWNLOADING state

SecurityException - the calling application does not have a proper permission to call this method

---

addAppStateListener

```
public void addAppStateListener(AppStateListener listener)
```

Adds an AppStateListener to monitor changes in the state of this application. If listener is null, the call is silently ignored.

Parameters:

`listener` - the listener

---

removeAppStateListener

`public void` **`removeAppStateListener`**`(AppStateListener listener)`

Removes the designated `AppStateListener` from this application. If `listener` is `null` or the listener was not added previously, the call is silently ignored.

Parameters:

`listener` - listener to remove

---

isVisible

`public boolean` **`isVisible`**`()`

Returns whether this application should be visible to users via UI or so.

Returns:

`true` if this application should be presented to end users. `false` otherwise

---

getName

`public String` **`getName`**`()`

Returns the name of this application. If there is a default language of choice, then the returned name shall be in the language. Otherwise, the underlying implementation should try its best to return an appropriate representation. If there was no name signaled, then this method returns a String of zero-length (that is, an empty string).

Returns:

the name of this application

---

getName

`public String` **`getName`**`(String lang)`

Returns the name of this application represented in the given language. If there is no representation in that language, returns an empty string.

Parameters:

`lang` - a language code designated in RFC 3066

Returns:

the name of this application in the given language

---

getNames

`public String[] getNames()`

Returns a list of the application names in all languages available in the signalling message. When there is no name signaled, this returns an array of zero-length.

Returns:

a list of the application names. A string at zero or even index is RFC 3066 [15] language code, and the corresponding entry at odd index contains the name of this application represented in the language designated with the language code

---

getIcon

`public AsyncResult getIcon(AsyncRequestor requestor)`

Gets the icon associated with this application. This method returns immediately without blocking, and the download of the icon progresses in the background if it is not fully loaded. The progress and the actual icon can be obtained via `AsyncRequestor` and `AsyncResult` interfacese. When completed, `AsyncResult.get()` returns `Image` for the loaded icon. If no icon is designated for this application, it returns `null`.

Parameters:

`requestor` - `AsyncRequestor` to get notified of the download progress. If such report is not required, `null` may be specified

Returns:

`AsyncResult` to track the download of the icon, and get the result

---

getDescription

`public String getDescription()`

Returns the description of this application. When there are more than one description each of which is represented in a different language, the name in the default language is returned. If no name represented in the default language, this returns an empty string.

Returns:

the description of this application

---

getDescription

`public String` **`getDescription`**`(String lang)`

Returns the description of this application in the designated language. If there was no description represented in the designated language, this method returns an empty string.

Parameters:

`lang` - a language code designated in RFC 3066

Returns:

the description of this application in the designated language if any. Otherwise, returns an empty string

---

getDescriptions

`public String[]` **`getDescriptions`**`()`

Returns all the descriptions of this application represented in different languages together with the corresponding language code. If there is no description signaled, returns an array of zero-length.

Returns:

descriptions of this application. 0 and even index contains a language code in RFC 3066 [15], and the corresponding odd index contains the description of this application represented in the designated language

Interface AppListListener

dmb.app

---

public interface **AppListListener**

An interface to be implemented by an object that needs to listen to changes in the application list maintained by `AppManager`. An event is delivered to this listener when there is any application added and/or removved to/from the list, and information on any application has been updated. Note that these changes do not include the download and the run state changes. They are delivered through `AppStateListener`.

Method Summary		Page
`void`	**`appListUpdated`**`()` Called when there is any change in the application list managed by `AppManager` and/or the information associated with any application.	*73*

Method Detail

appListUpdated

```
public void appListUpdated()
```

Called when there is any change in the application list managed by AppManager and/or the information associated with any application.

Class AppManager

dmb.app

```
java.lang.Object
  └ dmb.app.AppManager
```

---

final public class **AppManager**

extends Object

Manages applications known to the receiver. Applications are represented with AppControls, and they may be used to control and get information on the applications.

The list of applications maintainaed by this class contains all the applications known to the receiver while surfing channels and those being downloaded or stored in the receiver.

If there is any new application, or information on any of applications is changed, it is notified to the registered AppListListeners. In addition to that, the current states of applications may be monitored through AppStateListeners added directly to AppManager. It has the same effect as registering AppStateListeners with all the applications separately.

Method Summary		Page
static void	**addAppListListener**(AppListListener listener) Adds a listener to monitor changes in the list of applications.	75
static void	**addAppStateListener**(AppStateListener listener) Adds the given listener for monitoring state changes of any of applications known to the receiver.	76
static AppControl	**getAppControl**(String id) Returns an AppControl corresponding to an application of the given ID.	74
static AppControl[]	**getAppControls**() Returns a list of all the applications known to the receiver.	74
static AppControl[]	**getEventBoundApps**(String service) Returns a list of AppControls for the applications bound to one or more events in the service represented by the given locator.	75

static AppControl[]	**getServiceBoundApps**(String service) Returns a list of AppControls for the applications bound to the service represented by the given locator.	*74*
static void	**removeAppListListener**(AppListListener listener) Removes the given listener from the list of registered listeners.	*75*
static void	**removeAppStateListener**(AppStateListener listener) Removes the given listener from this AppManager.	*76*

Method Detail

getAppControl

public static AppControl **getAppControl**(String id)

Returns an AppControl corresponding to an application of the given ID.

Parameters:

id - application ID

Returns:

an AppControl for the application with the given ID. If there is no application with the given ID, then returns null

Throws:

NullPointerException - if the given ID is null.

---

getAppControls

public static AppControl[] **getAppControls**()

Returns a list of all the applications known to the receiver.

Returns:

a list of AppControls for all the applications known to the receiver

---

getServiceBoundApps

public static AppControl[] **getServiceBoundApps**(String service)

Returns a list of AppControls for the applications bound to the service represented by the given locator. The list only includes AppControl only for service-bound applications with no ones for event-bound ones.

Parameters:

service - a service locator

Returns:

a list of AppControls for the applications bound the given service. If there is no application bound to the given service, returns an array of zero-length

Throws:

NullPointerException - thrown if the given locator string is null

IllegalArgumentException - thrown if the given locator string is invalid. The locator may be invalid in terms of its format, or since designating no valid service

---

getEventBoundApps

```
public static AppControl[] getEventBoundApps(String service)
```

Returns a list of AppControls for the applications bound to one or more events in the service represented by the given locator. The list only includes AppControl only for event-bound applications with no ones for service-bound ones.

Parameters:

service - a service locator

Returns:

a list of AppControls for the applications bound the given service. If there is no application bound to the given service, returns an array of zero-length

Throws:

NullPointerException - thrown if the given locator string is null

IllegalArgumentException - thrown if the given locator string is invalid. The locator may be invalid in terms of its format, or since designating no valid service

---

addAppListListener

```
public static void addAppListListener(AppListListener listener)
```

Adds a listener to monitor changes in the list of applications. If listener is null, the call is silently ignored.

Parameters:

listener - a listner to add

---

removeAppListListener

```
public static void removeAppListListener(AppListListener listener)
```

Removes the given listener from the list of registered listeners. If it was not added or listener is null, the call is silently ignored.

Parameters:

`listener` - the listner to remove

---

addAppStateListener

`public static void` **`addAppStateListener`**`(AppStateListener listener)`

Adds the given listener for monitoring state changes of any of applications known to the receiver. If `listener` is `null`, no exception is thrown and the call is sliently ignored.

Parameters:

`listener` - a listener to add

---

removeAppStateListener

`public static void` **`removeAppStateListener`**`(AppStateListener listener)`

Removes the given listener from this `AppManager`. If it was not added or `listener` is `null`, the call is silently ignored.

Parameters:

`listener` - the listener to remove

Interface AppStateListener

dmb.app

---

public interface **AppStateListener**

An interface to be implemented by an object that needs to listen to changes in the state of applications.

Field Summary		Page
`String`	**`DOWNLOAD_PROGRESSED`** A constant string designating that there was a change in the progress reported by `AppControl.getProgress()` when an application is in DOWNLOADING or `AppControl.UPDATING` state.	*77*
`String`	**`DOWNLOAD_STATE_CHANGED`** A constant string designating that download state has been changed.	*77*

String	**REASON_APP_REQUESTED** Designates that the current state change is incurred by a request from an application.	*78*
String	**REASON_NETWORK_UNAVAILABLE** Designates that network status is bad or there is no network interface available in the device.	*78*
String	**REASON_OUT_OF_STORAGE** Designates that there is not enough storage for an application.	*78*
String	**REASON_UNKNOWN** Designates that the reason is unknown.	*78*
String	**REASON_VOLUNTEERED** Designates that the current state change is requested by the application itself.	*78*
String	**RUN_STATE_CHANGED** A constant string designating that run state has been changed.	*77*

Method Summary		*Page*
void	**stateChanged**(AppControl control, String event, int state, String reason) Called when the run or download state of an application has been changed.	*78*

Field Detail

DOWNLOAD_STATE_CHANGED

```
public static final String DOWNLOAD_STATE_CHANGED = "downloadStateChanged"
```

A constant string designating that download state has been changed. Given to stateChanged(AppControl, String, int, String) method as the second argument.

RUN_STATE_CHANGED

```
public static final String RUN_STATE_CHANGED = "runStateChanged"
```

A constant string designating that run state has been changed. Given to stateChanged(AppControl, String, int, String) method as the second argument.

DOWNLOAD_PROGRESSED

```
public static final String DOWNLOAD_PROGRESSED = "downloadProgressed"
```

A constant string designating that there was a change in the progress reported by AppControl.getProgress() when an application is in DOWNLOADING or AppControl.UPDATING state. Given to stateChanged(AppControl, String, int, String) method as the second argument.

REASON_OUT_OF_STORAGE

```
public static final String REASON_OUT_OF_STORAGE = "outOfStorage"
```

Designates that there is not enough storage for an application. This reason code is specified only when the download state of an application is changed to NOT_STORED state.

REASON_NETWORK_UNAVAILABLE

```
public static final String REASON_NETWORK_UNAVAILABLE = "networkUnavailable"
```

Designates that network status is bad or there is no network interface available in the device. This reason code is specified only when the download state of an application is changed to NOT_STORED state.

REASON_APP_REQUESTED

```
public static final String REASON_APP_REQUESTED = "appRequested"
```

Designates that the current state change is incurred by a request from an application.

REASON_VOLUNTEERED

```
public static final String REASON_VOLUNTEERED = "volunteered"
```

Designates that the current state change is requested by the application itself.

REASON_UNKNOWN

```
public static final String REASON_UNKNOWN = "unknown"
```

Designates that the reason is unknown.

Method Detail

stateChanged

```
public void stateChanged(AppControl control,
                         String event,
                         int state,
                         String reason)
```

Called when the run or download state of an application has been changed.

Parameters:

`control` - AppControl representing an application the state of which has been changed

`event` - kind of the event

`state` - the final state as the result of the change notified by this event. Depending on the kind of the event, this is set to a constant designating either a run or download state. When `event` is `DOWNLOAD_PROGRESSED` this should be `AppControl.DOWNLOADING` or `AppControl.UPDATING`.

`reason` - the reason of the state change. One of the constants defined in this interface prefixed with `REASON_`. When event is `DOWNLOAD_PROGRESSED`, this is always set to null

Interface MIDletControl

dmb.app

All Superinterfaces:
AppControl

---

public interface **MIDletControl**

extends AppControl

Provides a means to control an `MIDlet` associated with an instance of this interface. This interface is an extension to `AppControl` providing `MIDlet` specific features.

Fields inherited from interface dmb.app.AppControl
DOWNLOADING, INVALID, NOT_RUNNING, NOT_STORED, OUTDATED, PAUSED, STARTED, STORED, UPDATING

Method Summary		***Page***
String	**getProperty**(String key) Returns the same property value that may be retrieved by the application represented by this control via `MIDlet.getAppProperty(String)`.	*80*
AsyncResult	**init**(String[] props, AsyncRequestor r) Initializes this application with the given properties.	*80*
void	**switchTo**(String[] props) Switches the calling application to an application represented by this `AppControl` passing the given properties.	*80*

Methods inherited from interface dmb.app.AppControl
addAppStateListener, download, getDescription, getDescription, getDescriptions, getDownloadState, getIcon, getID, getName, getName, getNames, getProgress, getRunState, getVersion, init, isAutoDownload, isVisible, pause, remove, removeAppStateListener, start, stop, switchTo

Method Detail

init

```
public AsyncResult init(String[] props,
                        AsyncRequestor r)
                 throws SecurityException,
                        IllegalStateException
```

Initializes this application with the given properties. By an invocation of this method, the corresponding application should begin to move to PAUSED state.

Parameters:

`props` - List of properties, 0 and even index contains name of a property, and the corresponding value of the property is stored at the next index.

Returns:

`AsyncResult` object to query the current progress. `AsyncResult.get()` returns `null` if this application was successfully initialized. Otherwise, it will throw an exception representing the cause of the failure. In the case of `MIDlet`, it rethrows an exception thrown by `MIDlet` in the course of its initialization

Throws:

`SecurityException` - the calling application does not have proper permission to call this method

`IllegalStateException` - this application was in `INVALID`, `NOT_STORED`, or `DOWNLOADING`

---

switchTo

```
public void switchTo(String[] props)
```

Switches the calling application to an application represented by this `AppControl` passing the given properties. This method is identical to `switchTo()` except that this method allows for additional properties to be passed to the application being switched to. The properties can be retrieved with `MIDlet.getAppProperty(String)`.

Parameters:

`props` - List of properties, 0 and even index contains name of a property, and the corresponding value of the property is stored at the next index.

Throws:

`IllegalStateException` - if an application represented by this `AppControl` is in the `INVALID`, `NOT_STORED`, and `DOWNLOADING` state.

`SecurityException` - the calling application does not have proper permission to call this method

---

getProperty

```
public String getProperty(String key)
```

Returns the same property value that may be retrieved by the application represented by this control via `MIDlet.getAppProperty(String)`. If there is no such property, then `null` is returned.

Parameters:

`key` - property key

Returns:

the value corresponding to the specified key. If no such property exists, returns `null`

Package dmb.ca

Defines an interface to conditional access system in the receiver.

See:

Description

Interface Summary		Page
CAEventReceiver	Receives CA events from a CA module.	*83*
CASession	Represents a communication session between a CA module and an application.	*88*
Purchasable	Represents an entity that can be purchased via CA system.	*89*

Class Summary		Page
CAEvent	Represents an event generated by a CA module.	*82*
CAModule	Represents a CA module within the receiver.	*84*
CARequest	Represents a request to a CA module.	*87*
CAResponse	Represents an event generated by a CA module in response to a corresponding request sent to the module via `CASession.send(CARequest)`.	*88*

Exception Summary		Page
CARefusalException	An exception thrown when an operation is rejected by a CA system because an application performing it does not have enough right to do so.	*86*

Package dmb.ca Description

Defines an interface to conditional access system in the receiver. `Purchasable` may be implemented by objects representing products under control of the CA system. It provides a means to initiate a purchase procedure without dealing with the details of the procedure, and a separate means to open a message session with a CA module to directly control the purchase procedure by exchanging messages with the CA module and interacting with the user. Also `CAModule` is defined to open a message session for exchaning messages not related to a purchase.

Class CAEvent

dmb.ca

```
java.lang.Object
  └ dmb.ca.CAEvent
```

Direct Known Subclasses:

CAResponse

abstract public class **CAEvent**

extends Object

Represents an event generated by a CA module. This is the base class of all of the classes representing CA events.

Constructor Summary	Page
protected **CAEvent**() Creates an instance of CAEvent.	83

Constructor Detail

CAEvent

protected **CAEvent**()

Creates an instance of CAEvent. This constructor is provided for implementation convenience, and evolution of the specification. Therefore, applications are not supposed to use this constructor.

Interface CAEventReceiver

dmb.ca

public interface **CAEventReceiver**

Receives CA events from a CA module. An object interested in such events should implement this interface.

Method Summary	Page
Void **receive**(CAEvent e) Called when a CA event is sent from a CA module via a CA session.	83

Method Detail

receive

public void **receive**(CAEvent e)

Called when a CA event is sent from a CA module via a CA session.

Parameters:

e - the event object

Class CAModule

dmb.ca

java.lang.Object

└ **dmb.ca.CAModule**

All Implemented Interfaces:
Resource

---

abstract public class **CAModule**

extends Object

implements Resource

Represents a CA module within the receiver. To exchange messages with a CA module, a CASession must be open, and prior to doing so, the ownership of the r73CAModule must be acquired. Otherwise, a ResourceNotOwnedException is thrown.

To open a session with a CA module, the application must have an appropriate permission as follows:

- dmb.ca.session.<ca_id>: required to open a session to a CA module with an ID designated in <ca_id>

Constructor Summary	Page
protected **CAModule**() Creates an instance of CAModule.	84

Method Summary	Page
static CAModule **getDefault**() Returns the default CA module in the receiver.	85
int **getModuleID**() Returns the ID for this CA module.	85
static CAModule[] **list**() Returns all CA modules in the receiver.	85
CASession **openSession**(CAEventReceiver r) Opens a session to communicate with this CA module.	85

Constructor Detail

CAModule

protected **CAModule**()

Creates an instance of `CAModule`. This constructor is provided for implementation convenience, and evolution of the specification. Therefore, applications are not supposed to use this constructor.

Method Detail

getDefault

```
public static CAModule getDefault()
```

Returns the default CA module in the receiver.

Returns:

the default CA module

---

list

```
public static CAModule[] list()
```

Returns all CA modules in the receiver.

Returns:

list of all CA modules

---

getModuleID

```
public int getModuleID()
```

Returns the ID for this CA module.

Returns:

the CA ID

---

openSession

```
public CASession openSession(CAEventReceiver r)
                      throws ResourceNotOwnedException,
                             SecurityException
```

Opens a session to communicate with this CA module.

Parameters:

`r` - a `CAEventReceiver` to receive events originating from this CA module

Returns:

a CASession object that provides a means to communicate with this CA module. If it is possible to open a session for a reason not signaled with exceptions that may be thrown from this method, null may be returned

Throws:

ResourceNotOwnedException - thrown when the ownership of this module is not owned by the calling application

SecurityException - thrown when the calling application does not have permissions to call this method

Class CARefusalException

dmb.ca

java.lang.Object

  └ java.lang.Throwable

    └ java.lang.Exception

      └ **dmb.ca.CARefusalException**

All Implemented Interfaces:

Serializable

---

public class **CARefusalException**

extends Exception

An exception thrown when an operation is rejected by a CA system because an application performing it does not have enough right to do so.

Constructor Summary	*Page*
CARefusalException() Creates an instance of this exception with no detail message.	*86*
CARefusalException(String reason) Creates an instance of this exception with the given detail message.	*87*

Constructor Detail

CARefusalException

public **CARefusalException**()

Creates an instance of this exception with no detail message.

CARefusalException

`public` **`CARefusalException`**`(String reason)`

Creates an instance of this exception with the given detail message.

Class CARequest

dmb.ca

`java.lang.Object`

└ **`dmb.ca.CARequest`**

---

abstract public class **CARequest**

extends Object

Represents a request to a CA module. This is the base class for all the classes representing such requests.

Constructor Summary		Page
protected	**CARequest**() Creates an instance of `CARequest`.	*87*

Constructor Detail

CARequest

`protected` **`CARequest`**`()`

Creates an instance of `CARequest`. This constructor is provided for implementation convenience, and evolution of the specificiation. Therefore, applications are not supposed to use this constructor.

Class CAResponse

dmb.ca

```
java.lang.Object
  └ dmb.ca.CAEvent
      └ dmb.ca.CAResponse
```

---

abstract public class **CAResponse**

extends CAEvent

Represents an event generated by a CA module in response to a corresponding request sent to the module via `CASession.send(CARequest)`. All the classes representing such responses are extended from this class.

Constructor Summary		Page
protected	**CAResponse**() Creates an instance of `CAResponse`.	88

Constructor Detail

CAResponse

protected **CAResponse**()

Creates an instance of `CAResponse`. This constructor is provided for implementation convenience, and evolution of the specification. Therefore, applications are not supposed to use this constructor.

Interface CASession

dmb.ca

---

public interface **CASession**

Represents a communication session between a CA module and an application.

Method Summary		Page
void	**close**() Closes this session.	89

void	**send**(CARequest r) Sends a request to the CA module for which this session is established.	*89*

Method Detail

send

```
public void send(CARequest r)
          throws IOException
```

Sends a request to the CA module for which this session is established.

Parameters:

`r` - the CA request to send to the CA module

Throws:

`IOException` - thrown when this session is already closed

---

close

```
public void close()
```

Closes this session. Upon the termination of this session, the `CAEventReceiver` registered with the corresponding CA module is automatically removed.

If the application that acquired the ownership of the associated CA module loses its ownership, then the session established with the CA module is automatically closed.

Interface Purchasable

dmb.ca

---

public interface **Purchasable**

Represents an entity that can be purchased via CA system. This interface is implemented by objects representing such entities. For example, some of `SIObject`s retrieved via `dmb.si` API may implement this interface if it is not free, and should be purchased to consume. Also `AppControl` may do so if the application is not free.

For purchasing an entity, this interface provides two different ways of interacting with a CA module. First of all, an application can initiate a purchase session, and let the receiver and the CA module implementations deal with the remaining details. And an application can also open a purchase session and directly communicate with a CA module. In that case, all the interaction with the user must be handled by the application.

Field Summary		Page
String	**CURRENCY_CODE** A currency code defined by ISO 4217 [25].	91
String	**DESC** The description on the product in a purchase information.	92
String	**LONG_DESC** The long description on the product in a purchase information.	92
String	**LONG_NAME** The long name of the product in a purchase information.	92
String	**NAME** The name of the product in a purchase information.	91
String	**PRICE_FRACTION** The fraction part of the price in a purchase information.	91
String	**PRICE_FRACTION_DIGITS** The number of digits in the fraction part of the price in a purchase information.	91
String	**PRICE_INT** The integer part of the price in a purchase information.	91
String	**PRICE_SYMBOL** The currency symbol in a purchase information.	91
String	**PURCHASE_WINDOW_END** The end time of the purchase window during which a relevant product can be purchased.	92
String	**PURCHASE_WINDOW_START** The start time of the purchase window during which a relevant product can be purchased.	92

Method Summary		Page
AttributedObject	**getPurchaseInfo**() Returns an AttributedObject containing information on the product to be purchased.	93
boolean	**isAvailable**() Returns whether the product represented by this object may be purchased or not.	93
boolean	**isPurchased**() Returns whether this product is in purchased state or not.	93
CASession	**openPurchaseSession**(CAEventReceiver r) Opens a CASession to purchase a product represented by this object.	93
AsyncResult	**purchase**(AsyncRequestor r) Purchases the product represented by this object.	92

Field Detail

PRICE_INT

```
public static final String PRICE_INT = "priceInt"
```

The integer part of the price in a purchase information. The type of its value is `int`.

PRICE_FRACTION

```
public static final String PRICE_FRACTION = "priceFraction"
```

The fraction part of the price in a purchase information. The type of its value is `int`.

PRICE_FRACTION_DIGITS

```
public static final String PRICE_FRACTION_DIGITS = "priceFractionDigits"
```

The number of digits in the fraction part of the price in a purchase information. The type of its value is `int`.

CURRENCY_CODE

```
public static final String CURRENCY_CODE = "currencyCode"
```

A currency code defined by ISO 4217 [25]. The type of its value is `String`.

PRICE_SYMBOL

```
public static final String PRICE_SYMBOL = "priceSymbol"
```

The currency symbol in a purchase information. The type of its value is `String`.

NAME

```
public static final String NAME = "name"
```

The name of the product in a purchase information. The type of its value is `String`.

LONG_NAME

```
public static final String LONG_NAME = "longName"
```

The long name of the product in a purchase information. The type of its value is `String`.

DESC

```
public static final String DESC = "desc"
```

The description on the product in a purchase information. The type of its value is `String`.

LONG_DESC

```
public static final String LONG_DESC = "longDesc"
```

The long description on the product in a purchase information. The type of its value is `String`.

PURCHASE_WINDOW_START

```
public static final String PURCHASE_WINDOW_START = "purchaseWindowStart"
```

The start time of the purchase window during which a relevant product can be purchased. The type of its value is `Date`.

PURCHASE_WINDOW_END

```
public static final String PURCHASE_WINDOW_END = "purchaseWindowEnd"
```

The end time of the purchase window during which a relevant product can be purchased. The type of its value is `Date`.

Method Detail

purchase

```
public AsyncResult purchase(AsyncRequestor r)
```

Purchases the product represented by this object. This method returns immediately without blocking, and the actual progress can be monitored and controlled with `AsyncResult` returned from this method. And whenever possible, the purchase can be canceled by canceling the `AsyncResult`. If a purchase fails, `AsyncResult.complete()` throws a `CARefusalException`.

Parameters:

`r` - `AsyncRequestor` to be notified of the progress of the purchase. If such notification is not anticipated, then `null` may be specified

Returns:

`AsyncResult` to monitor and control the purchase initiated by this method

---

openPurchaseSession

```
public CASession openPurchaseSession(CAEventReceiver r)
```

Opens a `CASession` to purchase a product represented by this object. Opening a session does not actually begin the purchase procedure, but the application should exchange messages with the CA module to progress. The application may need to present UIs to provide relevant information and/or get input from the user in response to messages exchanged with the CA module.

Returns:

A `CASession` used to communicate with a CA module. If a session cannot be open for some reason, this returns `null`

---

isPurchased

```
public boolean isPurchased()
```

Returns whether this product is in purchased state or not.

Returns:

if purchased, `true`. Otherwise, `false`

---

isAvailable

```
public boolean isAvailable()
```

Returns whether the product represented by this object may be purchased or not. If it is out of the purchase window, then it may not be possible to purchase the product.

Returns:

if the product can be purchased, returns `true`. Otherwise, returns `false`

---

getPurchaseInfo

```
public AttributedObject getPurchaseInfo()
```

Returns an `AttributedObject` containing information on the product to be purchased. The standard set of attributes is defined in this class. But since some of them may not be supported depending on implementations, `AttributedObject.isValid(String)` must be consulted before getting attributes from the returned object.

Returns:

an `AttributedObject` containing information on the product represented by this object

Package dmb.io

Defines APIs for accessing data coming from the broadcast channel.

See:

Description

Interface Summary		Page
BroadcastFileConnection	Represents a connection to a broadcast file.	95
BroadcastFileListener	An interface to be implemented by an object to get notified of changes in broadcast files.	99
Trigger	Represents a datagram containing an associated timing when the data contained within the datagram will be used for some action.	100

Package dmb.io Description

Defines APIs for accessing data coming from the broadcast channel. The APIs defined in this package are extensions to "Generic Connection Framework" defined in `javax.microedition.io` package of Java ME CLDC 1.1.

This first provides an API to access broadcast files, which are repeatedly transmitted for reliable delivery. It also notifies changes in the files via a listener mechanism. A `BroadcastFileConnection` can be open with `Connector.open(String)` by specifying a locator to the file. Via the interface, the file system containing the file is mounted, and the contents of the file can be loaded and accessed. If a `BroadcastFileListener` is added, it can get notified of version changes of the file.

And `DatagramConnection` is used for receiving packets being broadcast, and for streams carrying triggers, the `DatagramConnection` returns `Trigger` that is an extension of `Datagram` to provide additional timing information.

Interface BroadcastFileConnection

dmb.io

All Superinterfaces:
AttributedObject, Connection, InputConnection

---

public interface **BroadcastFileConnection**

extends InputConnection, AttributedObject

Represents a connection to a broadcast file. If a URL to a broadcast file is given to `Connector`, and the file actually exists, a `BroadcastFileConnection` is returned. If the specified file does not exist, then an `IOException` is thrown.

Also to ease access to files within a broadcast file system, `open(String)` method is provided. Once a r113BroadcastFileConnection is obtained for a directory, `open(String)` may be called on it with relative paths to files to get another r113BroadcastFileConnections for files under the directory. If there is no such file corresponding to the specified path, then an `IOException` is thrown.

For reading the content of the file, then the methods defined in `InputConnection` may be used to get `InputStream`s. And other methods for getting size, name, and URL for the file are provided.

Broadcast files may be updated in the transmission side, and those updates can be monitored by adding a listener with addListener(BroadcastFileListener). Then the listener is notified when there is any change in the file.

Method Summary		Page
void	**addListener**(BroadcastFileListener listener) Adds a listener to get notified of changes in the content of this file.	98
long	**fileSize**() Returns the size of this file.	96
void	**flush**() Removes the cached content of this file if any.	98
String	**getName**() Returns the name of this file.	97
String	**getPath**() Returns the path to this file within the broadcast file system containing this file.	97
boolean	**isDirectory**() Reports whether this connection represents a directory or not.	97
String[]	**list**() Lists all the files contained in this connection in case this is a diectory.	97
AsyncResult	**load**(AsyncRequestor requestor) Starts the loading of the content of this file.	98
BroadcastFileConnection	**open**(String relPath) Opens a r113BroadcastFileConnection located at the specified path relative to this directory.	97
void	**removeListener**(BroadcastFileListener listener) Removes a BroadcastFileListener that was previously added to this file.	99

Methods inherited from interface dmb.util.AttributedObject
getAttributes, getBoolean, getBooleanList, getBytes, getDate, getDateList, getInt, getIntList, getLong, getLongList, getObject, getObjectList, getString, getStringList, isValid

Method Detail

fileSize

public long **fileSize**()

Returns the size of this file.

Returns:

the size in bytes

getName

```
public String getName()
```

Returns the name of this file.

Returns:

the name of this file

---

getPath

```
public String getPath()
```

Returns the path to this file within the broadcast file system containing this file.

Returns:

the file path

---

isDirectory

```
public boolean isDirectory()
```

Reports whether this connection represents a directory or not.

Returns:

if this represents a directory, returns `true`. Otherwise returns `false`

---

list

```
public String[] list()
```

Lists all the files contained in this connection in case this is a diectory. This method is only meaningful when `isDirectory()` returns `true`. If this is not a directory, then returns `null`. Note that this returns 0 if this directory is empty.

Returns:

list of all the files and directories within this directory. If this is not a directory, then returns `null`

---

open

```
public BroadcastFileConnection open(String relPath)
                            throws IOException
```

Opens a r113BroadcastFileConnection located at the specified path relative to this directory.

Parameters:

`relPath` - the relative path to the file to open

Returns:

the connection object

Throws:

`IOException` - thrown when the file does not exist

---

flush

```
public void flush()
```

Removes the cached content of this file if any. After invoking this method, the receiver loads the content of this file directly from broadcast stream, and all the `load` requests that were in progress are canceled. And if there was an open `InputStream` obtained from this connection, then any attempt to retrieve data from the stream will result in an `IOException`.

---

load

```
public AsyncResult load(AsyncRequestor requestor)
```

Starts the loading of the content of this file. This method returns immediately. And the actual progress can be monitored and controlled via `AsyncResult` returned from this method.

Parameters:

`requestor` - `AsyncRequestor` to get notified of progresses of the loading. `null` may be specified if such notification is not required

Returns:

An `AsyncResult` object for controlling the loading process. If the loading is failed, `AsyncResult.complete()` throws an `IOException`. `AsyncResult.get()` returns `null` upon successful loading

---

addListener

```
public void addListener(BroadcastFileListener listener)
                 throws IOException
```

Adds a listener to get notified of changes in the content of this file. A notification is delivered only when this file is fully loaded, and the version of this file in the broadcast stream is compared with the version of the loaded content. Once a notification is delivered, additional version changes are simply ignored if a new version is not loaded. Whether a file that is fully loaded is `flush()`"ed or not does not affect the monitoring of the version.

If the given `listener` parameter is `null`, it is silently ignored without throwing an exception.

Parameters:

`listener` - the listener to add

Throws:

`IOException` - thrown when there is not enough resource to monitor the underlying stream

---

removeListener

`public void` **`removeListener`**`(BroadcastFileListener listener)`

Removes a `BroadcastFileListener` that was previously added to this file. If the given `listener` is `null`, or it was never added to this file, then the call is silently ignored without incurring an exception.

Parameters:

`listener` - the listener to remove

Interface BroadcastFileListener

dmb.io

---

public interface **BroadcastFileListener**

An interface to be implemented by an object to get notified of changes in broadcast files. Upon notification, the content of the changed file is not automatically loaded. To load the new content, `BroadcastFileConnection.flush()` must be called first to clear the cache, and then `BroadcastFileConnection.load(AsyncRequestor)` need to be called.

See Also:

`BroadcastFileConnection`

Method Summary		Page
void	**`fileUpdated`**`(BroadcastFileConnection conn)` Called when a broadcast file has been updated.	*99*

Method Detail

fileUpdated

`public void` **`fileUpdated`**`(BroadcastFileConnection conn)`

Called when a broadcast file has been updated.

Parameters:

conn - BroadcastFileConnection object for accessing the broadcast file

Interface Trigger

dmb.io

All Superinterfaces:
Datagram, DataInput, DataOutput

---

public interface **Trigger**

extends Datagram

Represents a datagram containing an associated timing when the data contained within the datagram will be used for some action. This interface extends `javax.microedition.Datagram` to provide access to the timing information. `Trigger` conveys the timing information to synchronize the action triggered by it with a specific point in the associated A/V. Each `Trigger` has an associated trigger ID, and the timing to trigger an action associated with itself.

If more than one trigger is received, and their trigger IDs are identical, all of them are considered same, and reported to the receiving application only once. The reason for sending more than one trigger with the same ID is to increase the possibility of receiving a trigger. In addition to that, the trigger ID may be used to change the timing associated with a trigger. If a trigger is processed, then its ID may be reused over time for other triggers.

If an application is blocked for receiving a trigger, and a trigger of a specific ID is successfully received, then the `Trigger` object specified by the application is filled with the contents of the received trigger, and the calling thread returns immediately. If the time associated with the trigger has not come, `getState()` of the `Trigger` returns `RECEIVED`. If more than one trigger with the same ID is returned, then it means that the data associated with the trigger has been changed. Otherwise, no trigger with the same ID shall be returned more than once before the time associated with it passes. After the reception of a trigger, the receiver keeps track of the time associated with the trigger to notify a receiving application when the time has come. At that time, the application receives the same trigger (even though the corresponding trigger packet is not received again) with its state set to `TRIGGERED`. Responding to the reception, the application should perform an action associated with the trigger. As an exception, if the receiver could not track the time associated with the trigger, and it is sure that the time has passed already, then the trigger is simply canceled, and the same trigger is returned to the application with its state set to `CANCELED` this time. Once a trigger of a specific ID is delivered with either `TRIGGERED` or `CANCELED` state, then other triggers delivered after it with the same trigger ID are different ones.

When a stream conveying triggers is open with `Conector`, a `DatagramConnection` is returned and its `newDatagram` method shall return `Trigger` instead of `Datagram`.

See Also:

`DatagramConnection`

Field Summary		Page
int	**CANCELED** Represents that this trigger has been received before, and the receiver has failed to track the time associated with it.	101

int	**RECEIVED** Represents that this trigger is just received, and the time associated with it has not come yet.	*101*
int	**TRIGGERED** Represents that this trigger has been received before, and the time associated with it has just come.	*101*

Method Summary		*Page*
int	**getID**() Returns the ID for this trigger.	*102*
int	**getState**() Returns the state of this trigger.	*102*
long	**getTime**() Returns the time point associated with this trigger.	*102*

Field Detail

RECEIVED

```
public static final int RECEIVED = 0
```

Represents that this trigger is just received, and the time associated with it has not come yet. If there is more than one trigger with the same ID, only the first one will actually be delivered to the application, and others will be discared if it does not signal changes in the contents of the trigger.

See Also:

getState()

TRIGGERED

```
public static final int TRIGGERED = 1
```

Represents that this trigger has been received before, and the time associated with it has just come. The application receiving this trigger is considered to perform some action associated with this trigger. Note that a trigger in this state does not mean a new trigger packet is received. A trigger packet may be reported more than once with different states.

See Also:

getState()

CANCELED

```
public static final int CANCELED = 2
```

Represents that this trigger has been received before, and the receiver has failed to track the time associated with it.

See Also:

`getState()`

Method Detail

getState

```
public int getState()
```

Returns the state of this trigger. Once a trigger packet is received, it is delivered to an application in `RECEIVED` state. And if the time associated with it has come, the same trigger is delivered again in `TRIGGERED` state. As an exception, if the receiver has failed to track the time, and it is sure that the time has already passed, then the trigger is delivered in `CANCELED` state instead of `TRIGGERED` state.

Returns:

the state of this trigger

---

getID

```
public int getID()
```

Returns the ID for this trigger.

Returns:

the trigger ID

---

getTime

```
public long getTime()
```

Returns the time point associated with this trigger. This time is not exact, so should be used just as a hint. Especially, it is fairly difficult to estimate the exact time when there is a discontinuity in the media time of A/V streams associated with this trigger.

Returns:

The trigger time. Elapsed time in milliseconds since midnight, January 1, 1970 UTC. If it is impossible to estimate the time then returns `-1`

Package dmb.media

As an extension to MMAPI, this package contains additional classes and interfaces.

See:

Description

Interface Summary		Page
BackgroundVideoControl	A `Control` providing a means to control video presentation laid under the graphics plane.	*103*

Package dmb.media Description

As an extension to MMAPI, this package contains additional classes and interfaces. `BackgroundVideoControl` may be obtained from `Player`s presenting a broadcast video, and can be used to control the size and the location of the area where the video is displayed.

Interface BackgroundVideoControl

dmb.media

All Superinterfaces:
Control

---

public interface **BackgroundVideoControl**

extends Control

A `Control` providing a means to control video presentation laid under the graphics plane. `Player`s obtained from a `ServiceManager` and resposible for video presentation must return this control. And in the case of `Player`s created via `Manager` should also return this control if it supports presentation of video under the graphics plane. If both `javax.microedition.media.control.VideoControl` and this control are returned from a `Player`, the successful initialization of one of them must disable the other, resulting in the initialization failure of the disabled one.

Field Summary		Page
`String`	**`SIZE_CHANGED`** An event fired when the display size of a background video has been changed.	*104*

Method Summary		Page
`int`	**`getDisplayHeight`**`()` Returns the height of the area where the video is displayed.	*107*

int	**getDisplayWidth**() Returns the width of the area where the video is displayed.	*107*
int	**getDisplayX**() Returns the x coordinate of the top-left corner of the area where the video is displayed.	*106*
int	**getDisplayY**() Returns the y coordinate of the top-left corner of the area where the video is displayed.	*107*
int	**getSourceHeight**() Returns the height of the source video.	*106*
int	**getSourceWidth**() Returns the width of the source video.	*106*
int	**getZOrder**() Returns the z-order value of the video controlled by this control. 0 is top most, that is, closest to the viewer, and as the value increases, it is farther from the viewer.	*108*
void	**init**(Display display) Initializes this control for the presentation of video on the given `Display`.	*104*
void	**setDisplayBounds**(int x, int y, int width, int height) Sets the bounds of the area where video will be displayed.	*105*
void	**setVisible**(boolean visible) Sets the video visible or invisible depending on the argument.	*105*
void	**setZOrder**(int zOrder) Sets the z-order of the video the presentation of which is controlled by this control.	*107*
AsyncResult	**takeSnapshot**(String path, AsyncRequestor r) Takes the snapshot of the current video frame, and stores it to the given file in JPEG format.	*105*

Field Detail

SIZE_CHANGED

```
public static final String SIZE_CHANGED = "backgroundVideoSizeChanged"
```

An event fired when the display size of a background video has been changed. When this event is delivered to `PlayerListener.playerUpdate(Player, String, Object)`, the third argument to the method shall be a `BackgroundVideoControl` object associated with the background video.

Method Detail

init

```
public void init(Display display)
          throws IllegalStateException,
                 NullPointerException
```

Initializes this control for the presentation of video on the given `Display`. This method must be invoked before presentation begins, and cannot be called more than once.

Parameters:

`display` - the `Display` to present video on

Throws:

`IllegalStateException` - thrown if called more than once

`NullPointerException` - thrown when the argument is `null`

---

setDisplayBounds

```
public void setDisplayBounds(int x,
                             int y,
                             int width,
                             int height)
```

Sets the bounds of the area where video will be displayed. The bounds are specified in the screen's coordinate system. If the specified size is smaller than that of the original video, then it is implementation dependent whether the video is scaled or the portion outside the area is clipped out.

Parameters:

`x` - the x coodinate of the top-left corner of the video area

`y` - the y coodinate of the top-left corner of the video area

`width` - the width of the video area

`height` - the height of the video area

---

setVisible

```
public void setVisible(boolean visible)
```

Sets the video visible or invisible depending on the argument.

Parameters:

`visible` - if `true`, the video is visible. If `false`, set invisible

---

takeSnapshot

```
public AsyncResult takeSnapshot(String path,
                                AsyncRequestor r)
                         throws IOException
```

Takes the snapshot of the current video frame, and stores it to the given file in JPEG format. This method returns immediately without blocking, and the progress may be monitored and controlled via `AsyncResult`

and AsyncRequestor associated with each call to this method. AsyncResult.get() returns null upon completion of snapshot taking. If there was any problem in storing the file an IOException may be thrown.

Parameters:

path - a file (a path in a file system) to store the snapshot

r - AsyncRequestor to get notified of the progress of the snapshot taking. In case such notification is not required, null may be specified

Returns:

AsyncResult to monitor and control the progress

Throws:

IOException - thrown when an error is immediately detected (for example, when the given path is invalid)

---

getSourceWidth

```
public int getSourceWidth()
```

Returns the width of the source video.

Returns:

the width of the source video

---

getSourceHeight

```
public int getSourceHeight()
```

Returns the height of the source video.

Returns:

the height of the source video

---

getDisplayX

```
public int getDisplayX()
```

Returns the x coordinate of the top-left corner of the area where the video is displayed. The coordinate is in Display's coordinate system.

Returns:

the x coordinate of the video

---

getDisplayY

```
public int getDisplayY()
```

Returns the y coordinate of the top-left corner of the area where the video is displayed. The coordinate is in `Display`'s coordinate system.

Returns:

the y coordinate of the video

---

getDisplayWidth

```
public int getDisplayWidth()
```

Returns the width of the area where the video is displayed.

Returns:

the width of the video

---

getDisplayHeight

```
public int getDisplayHeight()
```

Returns the height of the area where the video is displayed.

Returns:

the height of the video

---

setZOrder

```
public void setZOrder(int zOrder)
              throws IllegalArgumentException
```

Sets the z-order of the video the presentation of which is controlled by this control. The z-order value is relative to each other, therefore there may be no video associated with a specific z-order value even though there are for two z-order values respectively larger and smaller than the value. If there are more than one video with the same z-order value, then the actual z-order is implementation dependent. Z-order of zero represents top most video, that is, closest to the viewer, and as the value increases, the video is farther from the viewer.

Parameters:

`zOrder` - the z-order value for the video

Throws:

`IllegalArgumentException` - if z-order is negative

---

getZOrder

```
public int getZOrder()
```

Returns the z-order value of the video controlled by this control. 0 is top most, that is, closest to the viewer, and as the value increases, it is farther from the viewer. When a `Player` is first created, the default value is set to 0 meaning top-most.

Returns:

the z-order of the video

Package dmb.messaging

Defines an API for applications to communicate with one another.

See:

Description

Interface Summary		Page
Message	Represents a message that can be sent to or received from a `Port`.	*109*

Class Summary		Page
Port	Represents a port where more than one application can communicate with one another.	*112*

Package dmb.messaging Description

Defines an API for applications to communicate with one another. The communication model supported by the API defined in this package is the message queue. With this messaging model, applications can synchronize their executions with one another as well as exchaning data.

Applications may share a `Port` by specifying a common name agreed upon among them. And they may send and/or receive messages via the shared port. The message is abstracted by `Message` object. It is a kind of buffer for copying data in/out to ports.

Interface Message

dmb.messaging

All Superinterfaces:
DataInput, DataOutput

---

public interface **Message**

extends DataInput, DataOutput

Represents a message that can be sent to or received from a `Port`. It has a buffer inside, and can be configured either read or write mode. Therefore once created, a message may be used repeatedly for sending and receiving data via `Port`s.

Reading a message

To set a message for reading, `readFrom(int)` should be called. It sets the contents of the message to be read from the designated byte-offset from the beginning of the message buffer. Since this class implements `DataInput`, the methods defined in the interface may be used to read the contents. If read beyond the length of the data returned from `getLength()`, an `EOFException` is thrown. And in the read mode, calling methods defined in `DataOutput` that is also implemented by this class incurs an `IOException`. If called more than once, `readFrom(int)` can be used to read the same message more than once from different positions.

Writing to a message

As in reading, writeFrom(int) should be called to set the message for writing. Then the message is set for writing from the specified byte-offset from the beginning of the message buffer. The methods defined in DataOutput can be used to write data, and if data is written beyond the current length of the message, the length is increased automatically. Once set for writing, the methods defined in DataInput cannot be called. If called, an IOException is thrown. If writeFrom(int) is called more than once, part of the otherwise same message may be modified multiple times.

State of a message after sending or receiving data

A message is sent to or received from a Port. After sending a message, there is no change in the state of the message. That is, the read/write mode previously set is retained as it was, and getPosition() and getLength() return the same values as before sending its data.

In the case of receiving data, the original data contained in the message is deleted, and the newly received data is copied to the message from the beginning of the internal buffer of the message. The length is set to that of data received from the Port, and the message is automatically set for reading from the beginning of the message buffer. Thus getPosition() returns 0.

Method Summary		Page
void	**end**() Sets the current position returned by getPosition() to be the length of this message.	111
int	**getLength**() Returns the length of this message.	111
int	**getPosition**() Returns the next position to read or write data from.	111
void	**readFrom**(int pos) Sets this message for reading from the given byte-offset from the beginning of the internal buffer.	110
void	**setCapacity**(int capacity) Sets the capacity of this message in bytes.	112
void	**writeFrom**(int pos) Sets this message for writing from the given byte-offset from the beginning of the internal buffer.	111

Method Detail

readFrom

public void **readFrom**(int pos)

Sets this message for reading from the given byte-offset from the beginning of the internal buffer. Once called, the methods defined in DataInput work as specified, and data can be read to the length of this message in the internal buffer.

Parameters:

pos - the byte-offset from which to read data in the internal buffer of this message

Throws:

`IllegalArgumentException` - thrown when the given `pos` parameter is less than 0, or larger than or equal to `getLength()`

---

writeFrom

```
public void writeFrom(int pos)
```

Sets this message for writing from the given byte-offset from the beginning of the internal buffer. Once called, the methods defined in `DataOutput` work as specified, and data can be written to the internal buffer. If data is written beyond the current length of the message, the length is increased automatically.

Parameters:

`pos` - the byte-offset from which to write data in the internal buffer of this message

Throws:

`IllegalArgumentException` - thrown when the given `pos` parameter is less than 0, or larger than or equal to `getLength()`

---

getPosition

```
public int getPosition()
```

Returns the next position to read or write data from.

Returns:

the next position to read or write data. An byte-offset from the beginning of the internal buffer

---

end

```
public void end()
```

Sets the current position returned by `getPosition()` to be the length of this message.

---

getLength

```
public int getLength()
```

Returns the length of this message. When a message is created for the first time, this is set to 0.

Returns:

the length of the message

setCapacity

`public void` **`setCapacity`**`(int capacity)`

Sets the capacity of this message in bytes. This value is used as a hint of the preferred buffer space. By respecing this value, the underlying implementation may keep the allocated space minimum, or reduce reallocations of internal buffer. But the actual size of the internal buffer may differ from the value specified via this method depending on implementations.

Parameters:

`capacity` - the preferred size of the buffer in bytes

Throws:

`IllegalArgumentException` - thrown when the given `capacity` parameter is negative or less than the length of this message

Class Port

dmb.messaging

`java.lang.Object`

└ **`dmb.messaging.Port`**

---

abstract public class **Port**

extends Object

Represents a port where more than one application can communicate with one another. To send data to other applications, a `Message` should be filled with the data. On the other hand, for receiving data, a `Message` to copy the data into should be specified.

A `Port` is implemented as a queue that is a FIFO (First In First Out) data structure, so the order of messages received with `receive(Message)` is the same as that of messages sent with `send(Message)`.

A `Port` can be configured either blocking or non-blocking mode. In the blocking mode, methods for sending or receiving a message block if it cannot be sent or received immediately. Conversely in the non-blocking mode, such methods return immediately returning `false` without performing the designated operation if it can be done immediately. In the case of the blocking mode, `setTimeout(long)` may be used to specify the timeout for which those methods will block without completing the designated operation. If the timeout expires without completing the designated operation, the methods return `false`. Therefore return values must be checked in the non-blocking mode.

Permissions

Access to `Port`s by applications can be controlled with the following permission.

- `dmb.messaging.Port.open.<name>`: permission to open a port with the specified <name>

Constructor Summary		Page
protected	**Port**() Creates an instance of Port.	113

Method Summary		Page
abstract void	**clear**() Removes all the Messages pending in this port.	117
abstract void	**close**() Closes this Port.	114
abstract void	**configureBlocking**(boolean block) Sets the blocking mode of this port.	114
abstract long	**getTimeout**() Returns the timeout for sending or receiving a messsage when this port is set to the blocking mode.	115
abstract boolean	**isBlocking**() Checks if this port is set to the blocking mode or not.	115
abstract Message	**newMessage**(int size) Creates a new Message instance for use with this Port that has a buffer of at least the given size.	115
static Port	**open**(String name, int capacity, boolean createIfNecessary) Returns a Port with the given name.	114
abstract boolean	**receive**(Message msg) Receives a message from this port.	116
abstract boolean	**send**(Message msg) Sends a message to this port.	116
abstract boolean	**sendFirst**(Message msg) Puts the given message to the front of the queue within this port.	117
abstract void	**setTimeout**(long timeout) Sets the timeout when sending or receiving a message in the blocking mode.	115

Constructor Detail

Port

protected **Port**()

Creates an instance of Port. This constructor is for implementation convenience and evolution of the specification. Therefore, applications must not make use of this constructor.

Method Detail

open

```
public static Port open(String name,
                        int capacity,
                        boolean createIfNecessary)
                 throws IOException,
                        SecurityException
```

Returns a `Port` with the given name. If there is already a `Port` with the given name, returns the instance. But if it is not the case and the `createIfNessary` parameter is `true`, a new `Port` is created and then returned.

Parameters:

`name` - name of the port

`capacity` - the maximum number of messages that can be stored without blocking within this port. This is meaningful only when a new port is created

`createIfNecessary` - if `true`, a new port is created when one with the given name is absent in the receiver

Throws:

`IOException` - thrown when the `createIfNcessary` parameter is `false` and there is no `Port` with the given name

`SecurityException` - thrown when the caller does not have a permission to call this method with the given parameters

`IllegalArgumentException` - thrown when the `capacity` parameter is negative

`NullPointerException` - thrown when the `name` parameter is `null`

---

close

```
public abstract void close()
```

Closes this `Port`. If there is no application holding this port unclosed, this port is actually removed from the receiver. Closed `Port`s should not be used, and no method should be invoked on them.

---

configureBlocking

```
public abstract void configureBlocking(boolean block)
```

Sets the blocking mode of this port.

Parameters:

`block` - if `true`, this port is set to the blocking mode. Otherwise set to the non-blocking mode

---

isBlocking

```
public abstract boolean isBlocking()
```

Checks if this port is set to the blocking mode or not. Newly created ports are set to the blocking mode. If this method is called after `close()` is called, the return value is not defined.

Returns:

`true` if this port is in the blocking mode. `false` otherwise

---

setTimeout

```
public abstract void setTimeout(long timeout)
```

Sets the timeout when sending or receiving a message in the blocking mode. If the timeout is expired without sending or receiving a message, the corresponding method returns returning `false`. If the timeout value is set to `0`, it means that those methods wait forever until sending or receiving a message. When a `Port` is created, its timeout defaults to `0`.

Parameters:

`timeout` - the timeout value in milliseconds

Throws:

`IllegalArgumentException` - thrown when the `timeout` parameter is negative

---

getTimeout

```
public abstract long getTimeout()
```

Returns the timeout for sending or receiving a messsage when this port is set to the blocking mode. The port waits for this timeout until sending or receiving a message. After the timeout is expired, the methods for sending or receiving a message returns `false`. If 0 is given, it means that those methods wait forever util sending or receiving a message. When a `Port` is created, its timeout defaults to `0`.

Returns:

the timeout value in milliseconds. If this `Port` is closed, then returns `-1`

---

newMessage

```
public abstract Message newMessage(int size)
```

Creates a new `Message` instance for use with this `Port` that has a buffer of at least the given size.

Parameters:

`size` - the minimum size of the internal buffer

Returns:

the created Message

Throws:

IllegalArgumentException - thrown when the size argument is negative

---

receive

```
public abstract boolean receive(Message msg)
                         throws IOException
```

Receives a message from this port. If this port is empty, then the behaviour of this method differs depending on the blocking mode set to this port. If this port is in the blocking mode, waits for a message for at most the timeout set with setTimeout(long) or until this Port is closed. If a message is not obtained as the result, this method returns false. On the other hand, if this port is in the non-blocking mode and there is no message immediately available for receiving, this method returns immediately with false as the return value.

Parameters:

msg - a Message object to store the message data obtained from this port

Returns:

if received successfully, returns true. Otherwise false. Note that false is returned when timeout is expired or this Port is closed without retrieving a message

Throws:

IOException - thrown when there was an error while receiving a message, or this Port is already closed

InterruptedIOException - thrown when the calling thread has been interrupted while blocking on this port

NullPointerException - thrown when the msg parameter is null

---

send

```
public abstract boolean send(Message msg)
                      throws IOException
```

Sends a message to this port. If there is no available free slot in this port, the behaviour of this method is different depending on the blocking mode set to this port. If this port is set to the blocking mode, the calling thread blocks for at most the timeout set with setTimeout(long) or until this Port is closed. If the timeout is expired or this Port is closed without sending the message, this method returns false. On the other hand, in the non-blocking mode, this method returns immediately with false as the return value if it is not possible to send the message without blocking.

Parameters:

msg - a Message object containing data sent to this port

Returns:

returns `true` if the message is sent. Otherwise `false`. Note that `false` is returned when timeout is expired or this `Port` is closed without sending a message

Throws:

`IOException` - thrown when there was an error while receiving a message, or this `Port` is already closed

`InterruptedIOException` - thrown when the calling thread has been interrupted while blocking on this port successfully. Otherwise returns `false`

`NullPointerException` - thrown when the `msg` parameter is `null`

---

sendFirst

```
public abstract boolean sendFirst(Message msg)
                          throws IOException
```

Puts the given message to the front of the queue within this port. That is, the message is sent prior to other messages waiting to be taken in this port. If there is no free slot, then the behaviour of this method is different depending on the blocking mode set to this port. In the blocking mode, the thread calling this method blocks for at most the timeout set with `setTimeout(long)` or until this `Port` is closed. If it could not send the message within the timeout or until this `Port` is closed, this method just returns `false`. On the other hand, in the non-blocking mode, this method returns immediately with `false` as the return value if the message cannot be sent without blocking.

Parameters:

`msg` - a `Message` object containing data sent to this port

Returns:

returns `true` if the message is sent. Otherwise `false`. Note that `false` is returned when timeout is expired or this `Port` is closed without sending a message

Throws:

`IOException` - thrown when there was an error while receiving a message, or this `Port` is already closed

`InterruptedIOException` - thrown when the calling thread has been interrupted while blocking on this port successfully. Otherwise returns `false`

`NullPointerException` - thrown when the `msg` parameter is `null`

---

clear

```
public abstract void clear()
```

Removes all the `Message`s pending in this port.

Package dmb.resources

Defines the basic framework to share resources among the receiver implementation and the aplications.

See:

Description

Interface Summary		Page
Resource	A marker interface to mark resources managed by `ResourceManager`.	120
ResourceOwner	Represents the owner of the resource acquired via `ResourceManager`.	128
ResourceOwnership	Represents an ownership of single or a set of resources acquired by a call to `acquire` or `tryAcquire` methods of `ResourceManager`.	129

Class Summary		Page
ResourceChoice	Represents a set of choices for resources.	120
ResourceGroup	Represents a group of resources to be acquired as a unit.	122
ResourceManager	Represents the system-wide resource manager, which keeps track of ownerships of various resources in the system.	123

Exception Summary		Page
InsufficientResourceException	An exception thrown when an operation cannot be performed because there is not enough resource.	119
ResourceNotOwnedException	An exception thrown when an application tries to call methods on a resource object it does not have the ownership.	127

Package dmb.resources Description

Defines the basic framework to share resources among the receiver implementation and the aplications. All the objects that are considered as resources are marked with `Resource` interface. Resources may be a hardware entity such as `Tuner`, or some abstract permission such as `KeyLock`.

`ResourceManager` is used to acquire the ownership of resources. Each application may specify the priority associated with each resource acquisition to express the priority of its operation using those resources. The receiver assigns resources based on these priorities. Each application may set a higher priority to retain resources of their concern, and may be notified when resources it owns are taken away by the receiver implementation or other applications.

Class InsufficientResourceException

dmb.resources

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └dmb.resources.InsufficientResourceException
```

All Implemented Interfaces:
Serializable

---

public class **InsufficientResourceException**

extends RuntimeException

An exception thrown when an operation cannot be performed because there is not enough resource.

Constructor Summary	*Page*
`InsufficientResourceException()` Creates an instance of this exception without specifying the reason.	*119*
`InsufficientResourceException(String desc)` Creates an instance of this exception with a string describing the reason.	*119*

Constructor Detail

InsufficientResourceException

```
public InsufficientResourceException()
```

Creates an instance of this exception without specifying the reason.

InsufficientResourceException

```
public InsufficientResourceException(String desc)
```

Creates an instance of this exception with a string describing the reason.

Interface Resource

dmb.resources

All Known Implementing Classes:

CAModule, KeyLock, ResourceChoice, ResourceGroup, ServiceManager, Tuner, TunerLock

---

public interface **Resource**

A marker interface to mark resources managed by `ResourceManager`.

Class ResourceChoice

dmb.resources

```
java.lang.Object
  └ dmb.resources.ResourceChoice
```

All Implemented Interfaces:

Resource

---

public class **ResourceChoice**

extends Object

implements Resource

Represents a set of choices for resources. Acquiring a `ResourceChoice` means that only one of the resources contained in it is required. Once a `ResourceChoice` is acquired, `getAcquired()` may be consulted to get the resource that is actually acquired among those within this `ResourceChoice`.

Note that this class is different from other `Resource`s, in that an instance of it cannot be specified in a request for resource acquisition if it is already acquired in the context of other request. If it is specified in such a case, then an `IllegalArgumentException` is raised.

See Also:

`ResourceGroup`

Constructor Summary	Page
ResourceChoice(Resource[] res) Creates a ResourceChoice from the given resources.	121

Method Summary		Page
Resource	**getAcquired**() Returns the resource that is actually acquired.	121
Resource[]	**getResources**() Returns an array containing resources consisting of this ResourceChoice.	121

Constructor Detail

ResourceChoice

public **ResourceChoice**(Resource[] res)

Creates a ResourceChoice from the given resources. The specified array is copied, so changing it does not affect the created ResourceChoice.

Throws:

IllegalArgumentException

Method Detail

getResources

public Resource[] **getResources**()

Returns an array containing resources consisting of this ResourceChoice.

Returns:

an array containing the resources. It is a copy of the array kept in this object

---

getAcquired

public Resource **getAcquired**()

Returns the resource that is actually acquired. If this ResourceChoice is not acquired, then returns null.

Returns:

the resource that is actually acquired. If this choice is not acquired, then returns null

Class ResourceGroup

dmb.resources

```
java.lang.Object
  └ dmb.resources.ResourceGroup
```

All Implemented Interfaces:

Resource

---

public class **ResourceGroup**

extends Object

implements Resource

Represents a group of resources to be acquired as a unit. If a `ResourceGroup` is submitted to `ResourceManager` for acquisition, the acquisition succeeds only when all the resources contained in it may be obtained at the same time. Otherwise, it is not acquired.

`ResourceGroup` may recursively contain `ResourceChoice`s. But r173ResourceGroup may not be nested directly or indirectly within another r173ResourceGroup. If used together with `ResourceChoice`, `ResourceGroup` may be used to specify various resource requirements.

See Also:

ResourceChoice

Constructor Summary	Page
ResourceGroup(Resource[] res) Creates an instance of `ResourceGroup` consisting of the given resources.	122

Method Summary		Page
Resource[]	**getResources**() Returns the resource consisting of this group as an array.	123

Constructor Detail

ResourceGroup

```
public ResourceGroup(Resource[] res)
```

Creates an instance of `ResourceGroup` consisting of the given resources.

Throws:

`IllegalArgumentException`

Method Detail

getResources

`public Resource[] getResources()`

Returns the resource consisting of this group as an array. This array is a copy of the array kept within this group.

Returns:

the resource array

Class ResourceManager

dmb.resources

`java.lang.Object`

`└ dmb.resources.ResourceManager`

---

abstract public class **ResourceManager**

extends Object

Represents the system-wide resource manager, which keeps track of ownerships of various resources in the system. Application may acquire a resource by calling `acquire(Resource,int,ResourceOwner, AsyncRequestor)`, and releases an acquired resource by calling `ResourceOwnership.release()` on the `ResourceOwnership` object obtained as the result of successfully gaining the ownership of the resource.

A resource may be single entity such as `Tuner`, but `ResourceGroup` may be used to acquire more than one resource at once. In this case, all of the resources represented by `ResourceGroup` must be able to be acquired as a whole, otherwise the request is failed. In addition to requesting a set of resources at once, `ResourceChoice` may be used to list a set of resources, only one of which should be acquired. `ResourceGroup` and `ResourceChoice` may be used together to specify complex requirements on resources.

The `ResourceOwner` given to `acquire(Resource,int, ResourceOwner,AsyncRequestor)` provides a means for this resource manager to communicate with the owner of the resource. That is, when other application or the underlying receiver implementation may need to take resources that are already owned by an application, the resource manager may communicate with the application via the methods defined in `ResourceOwner`. For details, refer to `ResourceOwner`.

`Resource`s in the receiver must be acquired before using via `ResourceManager`. Otherwise some methods of `Resource`s will throw `ResourceNotOwnedException`.

Permissions

Each application can specify a priority for the resources it acquires. But if any application can specify a priority as high as it wants, then any malfunctioning application may hamper proper operation of the receiver. So applications require the following permission to set an appropriate priority.

- `dmb.resources.priority.<priorityCeiling>`: A permission to set the priority to <priorityCeiling>. In other words, an application cannot specify a priority larger than <priorityCeiling>. When this permission is not specified at all, the corresponding application cannot specify a priority larger than `NORMAL`.

Field Summary		Page
static final int	**HIGH** A priority value meaning an application requires high prority in using the resources of its concern.	124
static final int	**LOW** A priority value meaning an application requires low prority in using the resources of its concern.	125
static final int	**NORMAL** A priority value meaning an application requires normal prority in using the resources of its concern.	125
static final int	**PRIVILEGED** A priority value meaning an application requires privileged right to the resources of its concern.	124

Constructor Summary		Page
protected	**ResourceManager**() Creates an instance of ResourceManager.	125

Method Summary		Page
abstract AsyncResult	**acquire**(Resource r, int priority, ResourceOwner owner, AsyncRequestor requestor) Definitely acquires the given resource by retrying until it is possible.	126
static ResourceManager	**getInstance**() Returns an instance of ResourceManager.	125
abstract AsyncResult	**tryAcquire**(Resource r, int priority, ResourceOwner owner, AsyncRequestor requestor) Tries to acquire the given resource only if it is possible.	125

Field Detail

PRIVILEGED

```
public static final int PRIVILEGED = 268435456
```

A priority value meaning an application requires privileged right to the resources of its concern. This value is usually specified by privileged applications such as EPG.

HIGH

```
public static final int HIGH = 536870912
```

A priority value meaning an application requires high prority in using the resources of its concern.

NORMAL

```
public static final int NORMAL = 805306368
```

A priority value meaning an application requires normal prority in using the resources of its concern.

LOW

```
public static final int LOW = 1073741824
```

A priority value meaning an application requires low prority in using the resources of its concern.

Constructor Detail

ResourceManager

```
protected ResourceManager()
```

Creates an instance of `ResourceManager`. This constructor is added for implementation convenience and easy evolution of the specification. Therefore applications are not supposed to use this constructor.

Method Detail

getInstance

```
public static ResourceManager getInstance()
```

Returns an instance of `ResourceManager`.

Returns:

`ResourceManager` instance

---

tryAcquire

```
public abstract AsyncResult tryAcquire(Resource r,
                                       int priority,
                                       ResourceOwner owner,
                                       AsyncRequestor requestor)
                                throws IllegalArgumentException,
                                       InsufficientResourceException,
                                       SecurityException
```

Tries to acquire the given resource only if it is possible. If it is possible to acquire the given resource, this method returns AsyncResult, otherwise null. This method returns immediately, and the returned AsyncResult may be used to monitor the progress of the acquisition process. When the ownership is acquired, then AsyncResult.get() returns a ResourceOwnership object representing the acquired ownership. When AsyncRequestor gets notified, it means that the resource is acquired. So a call to AsyncRequestor.resultUpdated(AsyncResult) may be considered as the completion of the resource acquisition without checking AsyncResult.get() and/or AsyncResult.getProgress().

Parameters:

r - the resource to try to acquire

priority - the priority associated with this acquisition of the resource. This priority is used to decide which application will own the resource. An application specifying the highest priority owns the resource

owner - the owner of the resource. If notification via ResourceOwner is not required, null may be specified

requestor - AsyncRequestor to get notified when the resource is actually acquired. If no notification is required, null may be specified

Returns:

if the resource cannot be acquired, returns null. Otherwise returns an AsyncResult object. When the resource is successfully acuiqred, AsyncResult.get() returns a ResourceOwnership object representing the ownership of the resource

Throws:

IllegalArgumentException - thrown when the specified resource represents or contains a resource unknown to the receiver or a ResourceChoice that is already acquired

InsufficientResourceException - thrown when this request cannot be satisfied, because there is not enough resource within the receiver regardless of the current ownership of them. For instance, it is impossible to prepare 3 tuners when there is only one in the receiver

SecurityException

Securityexception - thrown when the given priority is higher than the ceiling granted for the caller

---

acquire

```
public abstract AsyncResult acquire(Resource r,
                                    int priority,
                                    ResourceOwner owner,
                                    AsyncRequestor requestor)
                             throws IllegalArgumentException,
                                    InsufficientResourceException,
                                    SecurityException
```

Definitely acquires the given resource by retrying until it is possible. This method returns immediately, but the resource manager does the acquisition in the background. When the returned AsyncResult is done, it means that the resource is actually acquired (more specifically, AsyncResult.isDone() returns true, AsyncResult.isCanceled() false). When the ownership is acquired, AsyncResult.get() returns a ResouceOwnership object representing the acquired ownership of the resource.

Parameters:

r - the resource to acquire

`priority` - the priority associated with this acquisition of the resource. This priority is used to decide which application will own the resource. An application specifying the highest priority owns the resource

`owner` - the owner of the resource. If notification via `ResourceOwner` is not required, `null` may be specified

`requestor` - `AsyncRequestor` to get notified when the resource is actually acquired. If no notification is required, `null` may be specified

Returns:

returns an `AsyncResult` for tracking the acquisition. When the resource is acquired, `AsyncResult.get()` returns the `ResourceOwnership` representing the acquired ownership

Throws:

`IllegalArgumentException` - thrown when the specified resource represents or contains a resource unknown to the receiver or a `ResourceChoice` that is already acquired

`InsufficientResourceException` - thrown when this request cannot be satisfied, because there is not enough resource within the receiver regardless of the current ownership of them. For instance, it is impossible to prepare 3 tuners when there is only one in the receiver

`SecurityException`

`Securityexception` - thrown when the given `priority` is higher than the ceiling granted for the caller

Class ResourceNotOwnedException

dmb.resources

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └dmb.resources.ResourceNotOwnedException
```

All Implemented Interfaces:
Serializable

---

public class **ResourceNotOwnedException**

extends RuntimeException

An exception thrown when an application tries to call methods on a resource object it does not have the ownership. If it acquired a resource, and `ResourceOwner.notifyRelease(ResourceOwnership)` is called later, it means that the resource is already released, and this exception is thrown if methods on the resource object are called.

Constructor Summary	Page
ResourceNotOwnedException() Creates an instance of this exception with no detailed description on the reason.	128
ResourceNotOwnedException(String desc) Creates an instance of this exception with a string describing the reason.	128

Constructor Detail

ResourceNotOwnedException

`public ResourceNotOwnedException()`

Creates an instance of this exception with no detailed description on the reason.

ResourceNotOwnedException

`public ResourceNotOwnedException(String desc)`

Creates an instance of this exception with a string describing the reason.

Interface ResourceOwner

dmb.resources

---

public interface **ResourceOwner**

Represents the owner of the resource acquired via `ResourceManager`. For each acquisition of a resource, a separate `ResourceOwner` may be associated.

While an application owns a resource, and another application requests the same resource, then the resource manager compares the priority specified by the current resource owner with the new priority specified by the second application. If the second one specifies a higher priority, then it can take the ownership of the resource from the current owner.

When the current owner should release the ownership, `prepareRelease(ResourceOwnership)` of the `ResourceOwner` associated with the resource is called to give the application a chance to do some cleanup. Once the method returns, the ownership is actually transferred to the new owner. This method must return as soon as possible. If it takes too long before returning from this method, the ownership may be reclaimed forcibly. The amount of the time before forcing the reclamation is implementation-dependent.

If `prepareRelease(ResourceOwnership)` does not return too long, or the receiver need to reclaim the resource immediately without first notifying the resource manager, the resource may be reclaimed first and then `notifyRelease(ResourceOwnership)` may be called to notify the fact.

Method Summary		Page
void	**notifyRelease**(ResourceOwnership ownership) Called when the resource owned by this application has been reclaimed forcibly.	129
void	**prepareRelease**(ResourceOwnership ownership) Called when the resource manager requests the current owner of the resource to prepare for the release of the resource.	129

Method Detail

prepareRelease

```
public void prepareRelease(ResourceOwnership ownership)
```

Called when the resource manager requests the current owner of the resource to prepare for the release of the resource. Before this method returns, it must do anything required. This method must return as soon as possible, otherwise the receiver may reclaim the resource forcibly, and notify it with a call to notifyRelease(ResourceOwnership).

Parameters:

ownership - the ownership of the resource about to be relelased

---

notifyRelease

```
public void notifyRelease(ResourceOwnership ownership)
```

Called when the resource owned by this application has been reclaimed forcibly. Such cases include when prepareRelease(ResourceOwnership) does not return for a long time, or the receiver needs to reclaim the resource forcibly for some reasons.

Parameters:

ownership - the ownership of the resource that has been reclaimed

Interface ResourceOwnership

dmb.resources

---

public interface **ResourceOwnership**

Represents an ownership of single or a set of resources acquired by a call to acquire or tryAcquire methods of ResourceManager.

Method Summary		Page
void	**release**() Releases this ownership.	130
void	**setPriority**(int priority) Sets the priority associated with this ownership.	130

Method Detail

setPriority

```
public void setPriority(int priority)
```

Sets the priority associated with this ownership. If the previous priority and the new priority are identical, nothing happens. If there is a change in the priority, it works as if the ownership is first released, and then the resource is re-acquired with the new priority. But the process is atomic in the case of this method.

Parameters:

`priority` - the new priority

---

release

```
public void release()
```

Releases this ownership.

Package dmb.service

Defines APIs for service selection.

See:

Description

Interface Summary		Page
ServiceComponent	Represents a component within a service.	*131*
ServiceManagerListener	An interface to be implemented by an object to be notified of changes in `ServiceManager`.	*152*

Class Summary		Page
ServiceManager	Represents an entity managing the process of selecting a service at time, and monitoring the current selection for proper reaction to the changes in the environment related to the presentation of the current service.	*132*

Package dmb.service Description

Defines APIs for service selection. Service selection and control of components to be presented are done via `ServiceManager`. `dmb.service.ServiceManage` maintains the current service, and does all the complex tasks required for presentation of a service including control of tuner, CAS, A/V decoder, and so on.

Interface ServiceComponent

dmb.service

All Superinterfaces:
AttributedObject

---

public interface **ServiceComponent**

extends AttributedObject

Represents a component within a service. This may be an audio, video, or any other.

Field Summary		Page
`String`	**LANGUAGE** The language associated with this component.	*132*
`String`	**LOCATOR** The locator representing this component.	*132*

String	**MIME_TYPE** MIME type of this component.	*132*

Methods inherited from interface dmb.util.AttributedObject
getAttributes, getBoolean, getBooleanList, getBytes, getDate, getDateList, getInt, getIntList, getLong, getLongList, getObject, getObjectList, getString, getStringList, isValid

Field Detail

MIME_TYPE

```
public static final String MIME_TYPE = "mimeType"
```

MIME type of this component. A `String` value.

LANGUAGE

```
public static final String LANGUAGE = "language"
```

The language associated with this component. A language code designated in RFC 3066 [15]. A `String` value.

LOCATOR

```
public static final String LOCATOR = "locator"
```

The locator representing this component. A `String` value.

Class ServiceManager

dmb.service

```
java.lang.Object
  └ dmb.service.ServiceManager
```

All Implemented Interfaces:

Resource

abstract public class **ServiceManager**

extends Object

implements Resource

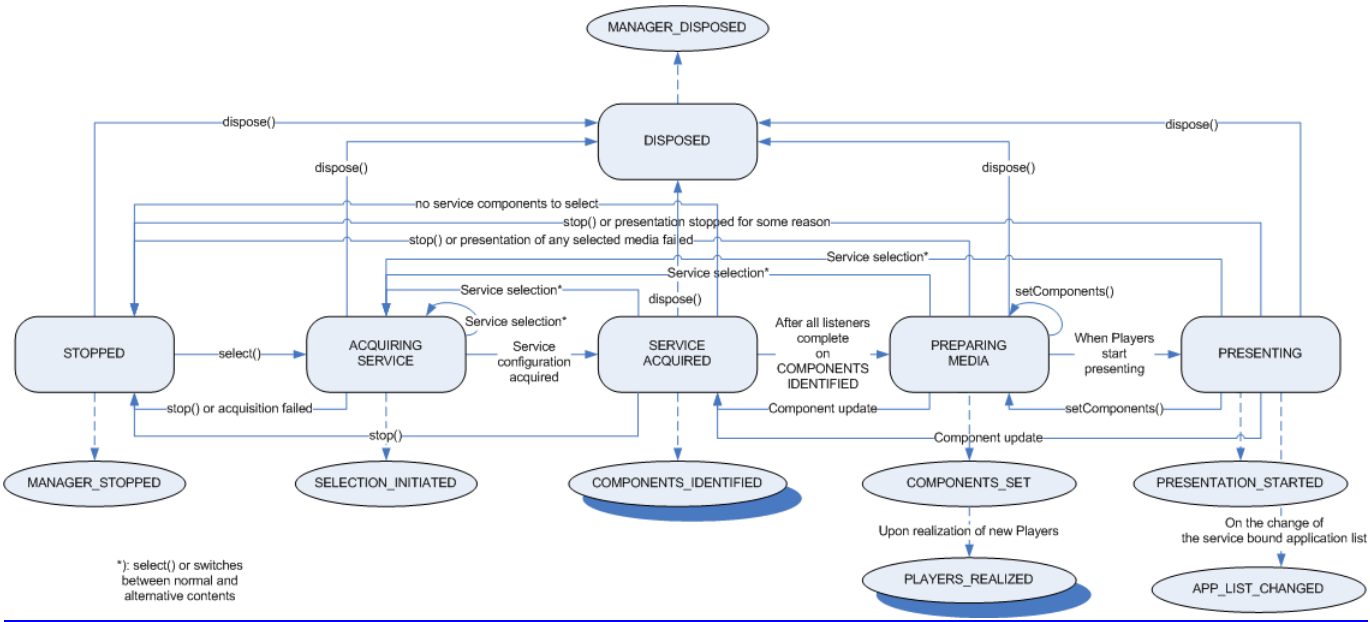
Represents an entity managing the process of selecting a service at time, and monitoring the current selection for proper reaction to the changes in the environment related to the presentation of the current service. `ServiceManager` does all the tasks required to present a selected service, that is, including retrieval of the list of components in the service, selecting the default components to be presented, acquiring resources required for the presentation, and beginning the presentation. In addition to those, `ServiceManager` monitors the current presentation and reacts to the various changes in the surrounding environment.

Besides selection of a service, `ServiceManager` provides a means to control presentation of a service component-by-component. It is possible to perform operations such as adding and removing components to/from the list of components being presented.

`ServiceManager` may be used by one application at a time, so it is considered as a resource. For applications to use a `ServiceManager` it should obtain the ownership for the `ServiceManager`. Otherwise, some methods in `ServiceManager` throw `ResourceNotOwnedException`.

State Model

`ServiceManager` can be in one of the six states - *`STOPPED`*, *`ACQUIRING_SERVICE`*, *`SERVICE_ACQUIRED`*, *`PREPARING_MEDIA`*, *`PRESENTING`*, *`DISPOSED`*. State transition among them is depicted in the following diagram.



Click the image to enlarge

In the diagram above, states are represented as rounded rectanges, transitions among them are as solid lines connecting the states where conditions for each transition are specified on the lines. Ovals are events that may be fired in the state to which it is connected via dotted lines. Shadows under some of the ovals represent that the corresponding events are intended to be delivered fully to the registered listeners before a transition to other state occurs. By "fully delivered", it means that all the listeners are notified of the events their event methods being called, and all those calls return to the caller.

STOPPED State

The initial state is STOPPED state. In this state, ServiceManager does not have a selected service, and therefore presents nothing. If there was a service being presented, the presentation is stopped, and all the Players used for the presentation of the service are closed with Player.close(). The following methods are worth noting in STOPPED state.

- getService(): returns null.
- getAllComponents(): returns an array of zero-length.
- getSelectedComponents(): returns an array of zero-length.
- getPlayer(ServiceComponent): Always throws IllegalArgumentException.
- getPlayers(): returns an array of zero-length.
- setComponents(ServiceComponent[], AsyncRequestor): Throws IllegalArgumentException. In this state, there is no selected service, so is no component to select.

ACQUIRING_SERVICE State

This state is entered when a service is selected by either a call to select(String,AsyncRequestor) or for some reason, a request from the underlying receiver implementation (e.g. when a preview period terminated). In this state, the receiver performs tasks required to access the service, and tries to obtain configuration of the service, that is, list of components with the service. If there was a service being presented, the presentation is stopped at some point in this state. If the acquisition fails, ServiceManager automatically moves into STOPPED state. The following methods are worth noting in this state.

- getService(): returns the selected service.
- getAllComponents(): returns an array of zero-length.
- getSelectedComponents(): returns an array of zero-length.
- getPlayer(ServiceComponent): always throws IllegalArgumentException.
- getPlayers(): returns an array of Players used for the presentation of the previous service but not closed yet. Those are in PREFETCHED or STARTED.
- setComponents(ServiceComponent[], AsyncRequestor): throws IllegalStateException. No service component exists for selection since the configuration of the selected service is not known yet in this state.

SERVICE_ACQUIRED State

When ServiceManager is ready to access a selected service and the configuration of the service is fully known, this state is entered. Upon entering this state, COMPONENTS_IDENTIFIED event is delivered to the registered listeners with a list of ServiceComponents selected for presentation by default. In case of no ServiceComponent to select, COMPONENTS_IDENTIFIED event is not delivered to the listeners and ServiceManager is changed to the STOPPED state. Any of the listeners may invoke setComponents(ServiceComponent[], AsyncRequestor) to change the default selection, and it will affect list of the initial components to be presented. But invoking setComponents in a listener will not affect the list of the default components passed to the other listeners. Only one of the listeners calling setComponents will affect the final list of components. But which one actually affects the selection of the initial components is implementation-dependent. After a call to setComponents, the result will immediately be effective making getSelectedComponents() returning the new list. When all the listeners are notified of the event, and returned from the event notification method, ServiceManager automatically enters PREPARING_MEDIA state. The following methods are worth noting in this state.

- getService(): returns the selected service.
- getAllComponents(): returns all the service components available in the service.

- getSelectedComponents(): returns the list of the components selected for presentation. Initially it returns the default selection passed to listeners, and reflects the components selected by any of listeners via setComponents method.
- getPlayer(ServiceComponent): throws IllegalArgumentException because no Player is allocated to ServiceComponents yet.
- getPlayers(): returns an array of Players used for the presentation of the previous service but not closed yet. Those are in PREFETCHED or STARTED.

PREPARING_MEDIA State

To control the presentation of the selected components, MMAPI is used. In this state, the receiver prepares Players to present selected service components. Players are newly created if required, and if some of Players used for the presentation of the previous service may be reconfigured and reused, then they are used.

Upon entering this state, COMPONENTS_SET event is delivered to registered listeners with a relevant reason code.

ServiceManager reuses Players if possible. Therefore, Players that are not closed are checked first to see if they may be reconfigured for the representation of new components. That is, Players used for rendering of audios may be reused only for audios, and those for videos may be only for videos. Remaining Players not selected for reuse are all closed. And for service components that are not allocated existing Players, new ones are created. If there is any newly created Player, then the application owning the ServiceManager via ResourceManager will receive PLAYERS_REALIZED event. The list of the newly created Players is passed to registered listeners with the event. All the Players are in REALIZED state. One of listeners receiving PLAYERS_REALIZED should initialize them appropriately. Especially those for presenting videos should be initialized by obtaining BackgroundVideoControl and calling init method on it. After all the listeners fully process PLAYERS_REALIZED event and return from the event notification method, the Players are automatically started, and the ServiceManager is moved into PRESENTING state.

If there is not enough resource for presenting the selected components, ServiceManager moves into STOPPED state.

The following methods are worth nothing in this state.

- getService(): returns the selected service.
- getAllComponents(): returns all the service components within the selected service.
- getSelectedComponents(): returns the list of components selected for presentation.
- getPlayer(ServiceComponent): returns a Player instance corresponding to the given component.
- getPlayers(): returns Players in REALIZED, PREFETCHED, or STARTED state that are allocated to service components to be presented.

PRESENTING State

This state is entered from PREPARING_MEDIA state after Players for presenting the selected components are properly created and initialized. Upon entering this state, PRESENTATION_STARTED event is delivered to registered listeners. In this state, all of methods defined in ServiceManager work as designated in the present document.

Even when a service is presented successfully, the presentation may stop automatically for some reasons. For instance, one of the resources used for the presentation may be revoked by the receiver implementation for some reason.

The following methods are worth nothing in this state.

- getService(): returns the selected service.
- getAllComponents(): returns all the service components within the selected service.
- getSelectedComponents(): returns the list of components selected for presentation.

- getPlayer(ServiceComponent): returns a Player instance corresponding to the given component.
- getPlayers(): returns Players in REALIZED, PREFETCHED, or STARTED state that are allocated to service components to be presented.
- getServiceBoundApps(): returns the list of AppControl for applications bound to the current service being presented.
- getEventBoundApps(): returns the list of AppControl for applications bound to one or more events in the current service being presented.

DISPOSED State

This state is entered when an application calls dispose(). Any method invoked in this state throws an IllegalStateException. Different from STOPPED state, this state guarantees that ServiceManager releases all the resources it held before.

Resource Acquisition

Depending on platforms, resources such as tuners are required for the presentation of a service. ServiceManager acquires appropriate service when selecting a service. The entity requesting and owning a resource at this time is not the application selecting a service but the ServiceManager itself. It obtains resources based on the priority set by setPriority(int). The priority affects the resource acquisition as if it was specified to ResourceManager when acquiring a resource. When created, a ServiceManager has NORMAL priority.

Default Instance

There is one ServiceManager instance that is automatically created by the underlying receiver implementation. It is the default instance, and shared among all the applications running on the receiver. The default instance reflects the current selection of service from the view point of the user. Therefore, EPG and channel navigator applications either select a service via the default instance or effectively behave as if they do so. The name of the default instance is DEFAULT. Besides the default instance, applications may create instances of ServiceManager and share them with other applications via export mechanism. For further details, see export(String).

A/V Presentation

If a selected service contains are A/V components, corresponding Players are created and set to REALIZED state. When the PLAYERS_REALIZED event is delivered, those Players are passed to registered listeners. Any of the listeners may fetch appropriate Controls from the Players, and set up them appropriately before presentation begins. For example, in the case of video components, BackgroundVideoControl may be fetched and used to control the size and position of the video presentation. Components sharing the same timeline shares the same Player.

If an application owning a ServiceManager has access to Players associated with the ServiceManager, all the Players are closed when the application loses the ownership of the ServiceManager.

Each Player is reused if possible. When selecting a service or changing components, unclosed Players are first checked if they are reusable. For new components are to be associated with a specific Player, the Controls obtainable from the Player must be identical to those required by the new components. If some of them are not reusable, new Players are created. Players that are not reusable will be closed with Player.close().

Preference Setting

Preferences may be specified that will affect service selection and selection of default components to be presented. For example, LANGUAGE preference may be specified to express the preference in the language associated with service cmponents. When selecting an audio component among multiple choices, the language preference will be respected. Preferences may be set with setPreference(String, Object).

Permissions

To invoke some methods on ServiceManager, the invoking application must have appropriate permissions. Otherwise, a SecurityException may be thrown. The followings are list of permissions relevant to ServiceManager.

- dmb.service.ServiceManager.setPriority.<name>: Permission to set the priority for resource acquisition on ServiceManager of the given name. Note that the maximum priority that can be set to ServiceManager is the priorityCeiling specified with another permission, dmb.resources.priority.<priorityCeiling>
- dmb.service.ServiceManager.createInstance: Permission to create an instance of ServiceManager (createInstance())
- dmb.service.ServiceManager.getManagingInstance: Permission to get the managing instance for the calling application (getManagingInstance())
- dmb.service.ServiceManager.getInstance.<name>: Permission to get an instance of ServiceManager with the given name (getInstance(String))
- dmb.service.ServiceManager.export.<name>: Permission to share a ServiceManager instance with other applications (export(String))
- dmb.service.ServiceManager.dispose.<name>: Permission to dispose a ServiceManager instance of the given name (dispose())
- dmb.service.ServiceManager.getPlayers.<name>: Permission to get Player for controlling the presentation of the current service selected to a ServiceManager of the given name (getPlayer(ServiceComponent), getPlayers())
- dmb.service.ServiceManager.stop.<name>: Permission to stop service presentation of a ServiceManager with the given name (stop(AsyncRequestor))
- dmb.service.ServiceManager.select.<name>.<locator>: Permission to select a service specified by the given locator to a ServiceManager with the given name (select(String,AsyncRequestor))

Field Summary		Page
static final int	**ACQUIRING_SERVICE** A constant indicating that this ServiceManager is in the *acquiring service* state.	140
static final String	**DEFAULT** Name of the default instance.	139
static final int	**DISPOSED** A constant indicating this ServiceManager is in *disposed* state.	140
static final String	**LANGUAGE** Represents preference on a language.	139
static final String	**MASK_APP** Represents whether service or event bound applications shall be launched or not when a service is selected.	140
static final int	**PREPARING_MEDIA** A constant indicating that this ServiceManager is in the *media prepared* state.	141
static final int	**PRESENTING** A constant indicating that this ServiceManager is in the *presenting* state.	141

static final int	**SERVICE_ACQUIRED** A constant indicating that this ServiceManager is in the *service acquired* state.	*141*
static final int	**STOPPED** A constant indicating this ServiceManager is in the *stopped* state.	*140*

Constructor Summary		Page
protected	**ServiceManager**() Creates an instance of ServiceManager.	*141*

Method Summary		Page
abstract void	**addListener**(ServiceManagerListener l) Adds a listener to be notified of changes in this ServiceManager.	*150*
static ServiceManager	**createInstance**() Creates a new instance.	*142*
abstract void	**dispose**() Disposes of this instance.	*143*
abstract void	**export**(String name) Exports this ServiceManager with the given name.	*143*
abstract ServiceComponent[]	**getAllComponents**() Returns all the components constituting the current service.	*145*
abstract ServiceComponent[]	**getComponents**(Player player) Returns ServiceComponents presented by the given Player.	*146*
abstract AppControl[]	**getEventBoundApps**() Returns the list of AppControls denoting applications which are bound to one or more events in the service being presented.	*147*
static ServiceManager	**getInstance**(String name) Returns a shared ServiceManager instance with the given name.	*142*
static ServiceManager	**getManagingInstance**() Returns the ServiceManager managing the calling application.	*142*
abstract Player	**getPlayer**(ServiceComponent component) Returns a Player responsible for the presentation of the given service component.	*145*
abstract Player[]	**getPlayers**() Returns all the Players created but not closed by this ServiceManager.	*144*
abstract Object	**getPreference**(String key) Returns the preference value associated with the given key.	*151*

abstract int	**getPriority**() Returns the priority this ServiceMananger uses when acquiring resources required for presentation of a selected service.	*144*
abstract ServiceComponent[]	**getSelectedComponents**() Returns the components current being presented among those constituting the current service.	*146*
abstract String	**getService**() Returns the locator of the service currently selected to this ServiceManager.	*148*
abstract AppControl[]	**getServiceBoundApps**() Returns the list of AppControls denoting applications which are bound to the service being presented.	*147*
abstract int	**getState**() Returns the current state of this ServiceManager.	*143*
abstract void	**removeListener**(ServiceManagerListener l) Removes the given listener registered previous.	*151*
abstract AsyncResult	**select**(String locator, AsyncRequestor r) Selects a service represented by the given locator.	*148*
abstract AsyncResult	**setComponents**(ServiceComponent[] components, AsyncRequestor r) Sets the components to be presented within the current service.	*149*
abstract Object	**setPreference**(String key, Object value) Sets a preference.	*151*
abstract void	**setPriority**(int priority) Sets the priority used when this ServiceManager acquires resources required for presentation of a selected service.	*144*
abstract AsyncResult	**stop**(AsyncRequestor r) Stops the presentation of the current service.	*150*

Field Detail

DEFAULT

```
public static final String DEFAULT = "default"
```

Name of the default instance.

See Also:

getInstance(String)

LANGUAGE

```
public static final String LANGUAGE = "language"
```

Represents preference on a language. This preference affects selection of service components presented to users by default. For example, if there are more than one audio streams in different languages, this preference is considered to select an audio track in the specified language. The value should be a language code defined in RFC 3066 [15].

See Also:

`setPreference(String,Object)`

MASK_APP

```
public static final String MASK_APP = "maskApp"
```

Represents whether service or event bound applications shall be launched or not when a service is selected. The value must be one of `Boolean.TRUE` and `Boolean.FALSE`. The former means that bound applications must not be launched, and the latter must be launched. The default value for this preference is `Boolean.FALSE` meaning applications must automatically launched.

See Also:

`setPreference(String,Object)`

DISPOSED

```
public static final int DISPOSED = 0
```

A constant indicating this `ServiceManager` is in *disposed* state.

See Also:

`getState()`

STOPPED

```
public static final int STOPPED = 1
```

A constant indicating this `ServiceManager` is in the *stopped* state.

See Also:

`getState()`

ACQUIRING_SERVICE

```
public static final int ACQUIRING_SERVICE = 2
```

A constant indicating that this `ServiceManager` is in the *acquiring service* state.

See Also:

getState()

SERVICE_ACQUIRED

```
public static final int SERVICE_ACQUIRED = 3
```

A constant indicating that this ServiceManager is in the *service acquired* state.

See Also:

getState()

PREPARING_MEDIA

```
public static final int PREPARING_MEDIA = 4
```

A constant indicating that this ServiceManager is in the *media prepared* state.

See Also:

getState()

PRESENTING

```
public static final int PRESENTING = 5
```

A constant indicating that this ServiceManager is in the *presenting* state.

See Also:

getState()

Constructor Detail

ServiceManager

```
protected ServiceManager()
```

Creates an instance of ServiceManager. This constructor is for implementation convenience and evolution of the specification. Therefore, applications must not make use of this constructor.

Method Detail

getManagingInstance

```
public static ServiceManager getManagingInstance()
                                        throws SecurityException
```

Returns the `ServiceManager` managing the calling application. The managing instance is the `ServiceManager` that launched the calling application in the context of a service which the application is bound to.

Returns:

the managing instance if there is one. If the calling application does not have a managing instance, returns `null`

Throws:

`SecurityException` - thrown when the caller does not have proper permission to call this method

---

getInstance

```
public static ServiceManager getInstance(String name)
                                  throws SecurityException,
                                         NullPointerException
```

Returns a shared `ServiceManager` instance with the given name.

Parameters:

`name` - the name of the instance to retrieve

Returns:

the instance with the given name. If there is no such instance, then returns `null`

Throws:

`SecurityException` - thrown when the calling application does not have proper permission to call this method

`NullPointerException` - thrown when `name` argument is `null`

---

createInstance

```
public static ServiceManager createInstance()
                                     throws SecurityException
```

Creates a new instance.

Returns:

the created instance

Throws:

`SecurityException` - thrown when the caller does not have a permission to create an instance

---

getState

```
public abstract int getState()
```

Returns the current state of this `ServiceManager`.

Returns:

a constant indicating the current state. One of `DISPOSED`, `STOPPED`, `ACQUIRING_SERVICE`, `SERVICE_ACQUIRED`, `PRESENTING`

---

export

```
public abstract void export(String name)
                    throws IllegalArgumentException,
                           IllegalStateException,
                           SecurityException,
                           NullPointerException
```

Exports this `ServiceManager` with the given name. An exported instance may be shared with other applications by allowing them to obtain the instance by specifying the given name to `getInstance(String)`. Note that if this instance is the managing instance of this application, it is impossible to export it even when it is not already exported. If tried, this method will throw `SecurityException`.

Parameters:

`name` - the name of this instance

Throws:

`IllegalArgumentException` - thrown when there is an instance exported with the same name as `name`

`IllegalStateException` - thrown when this instance is already exported, or it is in `DISPOSED` state

`SecurityException` - thrown when the caller does not have permission to call this method, or when the caller tries to export its managing instance

`NullPointerException` - thrown when the given `name` parameter is `null`

---

dispose

```
public abstract void dispose()
                     throws SecurityException
```

Disposes of this instance. Once diposed, applications can not make a call on this `ServiceManager`. If tried, it will result in an `IllegalArgumentException` thrown. And if this instance was exported, then the exported

instance will no longer be accessible with `getInstance(String)`. Applications can dispose of instances they created without any special permission. If this method is called more than once, it will be siliently ignored without an exception thrown.

Throws:

`SecurityException` - thrown when the calling application does not have permission to dispose of this instance

---

setPriority

```
public abstract void setPriority(int priority)
                         throws SecurityException,
                                IllegalStateException
```

Sets the priority used when this `ServiceManager` acquires resources required for presentation of a selected service.

Parameters:

`priority` - the priority. This is the priority given to `ResourceManager` when acquiring resources

Throws:

`SecurityException` - thrown when the calling application does not have proper permission to call this method

`IllegalStateException` - thrown when this instance is in `DISPOSED` state

---

getPriority

```
public abstract int getPriority()
```

Returns the priority this `ServiceMananger` uses when acquiring resources required for presentation of a selected service. The default value is `ResourceManager.NORMAL`.

Returns:

the current priority to be specified when acquiring required resources

Throws:

`IllegalStateException` - thrown when this `ServiceManager` has been disposed

---

getPlayers

```
public abstract Player[] getPlayers()
                             throws SecurityException,
                                    IllegalStateException
```

Returns all the `Player`s created but not closed by this `ServiceManager`. When this `ServiceManager` is in `STOPPED` state, returns an array of zero-length.

Returns:

an array containing `Player`s created for the presentation of the current service but not closed yet

Throws:

`SecurityException` - thrown when the calling application does not have permission to obtain `Player`s

`IllegalStateException` - thrown when this `ServiceManager` has been disposed

---

getPlayer

```
public abstract Player getPlayer(ServiceComponent component)
                         throws SecurityException,
                                IllegalArgumentException,
                                IllegalStateException,
                                NullPointerException
```

Returns a `Player` responsible for the presentation of the given service component.

Parameters:

`component` - the service component

Returns:

the `Player` responsible for the presentation of the given service component. If the component is not being presented, returns `null`

Throws:

`SecurityException` - thrown when the calling application does not have permission to call this method

`IllegalArgumentException` - thrown when the given component does not belong to the current service

`IllegalStateException` - thrown when this `ServiceManager` is in `DISPOSED` state

`NullPointerException` - thrown when the given argument, `component` is `null`

---

getAllComponents

```
public abstract ServiceComponent[] getAllComponents()
                                        throws IllegalStateException
```

Returns all the components constituting the current service. Returns an array of zero-length in the following cases.

1. `ServiceManager` is in `STOPPED` state
2. `ServiceManager` is in `ACQUIRING_SERVICE` state

Returns:

an array containing all the `ServiceComponent`s constituting the current service

Throws:

`IllegalStateException` - thrown when this `ServiceManager` is in `DISPOSED` state

---

getSelectedComponents

```
public abstract ServiceComponent[] getSelectedComponents()
                                                  throws IllegalStateException
```

Returns the components current being presented among those constituting the current service. This method returns an array of zero-length in the following conditions.

1. `ServiceManager` is in `STOPPED` state
2. `ServiceManager` is in `ACQUIRING_SERVICE` state
3. `ServiceManager` is in `SERVICE_ACQUIRED` state

Returns:

an array containing service components being presented

Throws:

`IllegalStateException` - thrown if this `ServiceManager` is in `DISPOSED` state

---

getComponents

```
public abstract ServiceComponent[] getComponents(Player player)
                                        throws IllegalArgumentException,
                                               NullPointerException,
                                               IllegalStateException
```

Returns `ServiceComponent`s presented by the given `Player`.

Parameters:

`player` - one of `Player`s returned from `getPlayers()`

Returns:

an array of `ServiceComponent`s for the presentation of which the given `Player` is resposible

Throws:

`IllegalArgumentException` - thrown if the given `player` is not related to this `ServiceManager`

`NullPointerException` - thrown when the given `player` parameter is `null`

`IllegalStateException`

`IllegalStatetException` - thrown when this `ServiceManager` is in `DISPOSED` state

---

getServiceBoundApps

```
public abstract AppControl[] getServiceBoundApps()
                                        throws IllegalStateException
```

Returns the list of AppControls denoting applications which are bound to the service being presented. The list only includes AppControl only for service-bound applications with no ones for event-bound ones. This method returns an array of zero-length in the following conditions.

1. ServiceManager is in STOPPED state
2. ServiceManager is in ACQUIRING_SERVICE state
3. ServiceManager is in SERVICE_ACQUIRED state
4. ServiceManager is in PRESENTING state and the first ServiceManagerListener.APP_LIST_CHANGED event is not yet delivered
5. ServiceManager is in PRESENTING state and no service-bound applications are found

Returns:

an array of AppControls for applications bound to the service being currently presented.

Throws:

IllegalStateException

IllegalStatetException - thrown when this ServiceManager is in DISPOSED state

---

getEventBoundApps

```
public abstract AppControl[] getEventBoundApps()
                                      throws IllegalStateException
```

Returns the list of AppControls denoting applications which are bound to one or more events in the service being presented. The list only includes AppControl only for event-bound applications with no ones for service-bound ones. This method returns an array of zero-length in the following conditions.

1. ServiceManager is in STOPPED state
2. ServiceManager is in ACQUIRING_SERVICE state
3. ServiceManager is in SERVICE_ACQUIRED state
4. ServiceManager is in PRESENTING state and the first ServiceManagerListener.APP_LIST_CHANGED event is not yet delivered
5. ServiceManager is in PRESENTING state and no event-bound applications are found

Returns:

an array of AppControls for applications bound to one or more events in the service being currently presented.

Throws:

IllegalStateException

IllegalStatetException - thrown when this ServiceManager is in DISPOSED state

getService

```
public abstract String getService()
                          throws IllegalStateException
```

Returns the locator of the service currently selected to this ServiceManager.

Returns:

the locator for the current service. Returns null when this ServiceManager is in STOPPED state

Throws:

IllegalStateException - thrown when this ServiceManager is in DISPOSED state

select

```
public abstract AsyncResult select(String locator,
                                   AsyncRequestor r)
                            throws ResourceNotOwnedException,
                                   IllegalArgumentException,
                                   SecurityException,
                                   NullPointerException,
                                   IllegalStateException
```

Selects a service represented by the given locator. This method returns immediately, and the exact progress of the selection process may be monitored with AsyncResult and AsycnRequestor associated with this method. When the selection completes, AsyncResult.get() returns the locator specified in this method. Otherwise, it throws the following exceptions depending on the cause of the failure. As a special case, if a selection is canceled by another selection, then the first selection is treated as if it were canceled, and AsyncResult.isCanceled() returns true.

1. TuningFailedException: when tuning failed
2. InsufficientResourceException: when some of the required resources could not be acquired
3. CARefusalException: when it is refused by CAS
4. MediaException: when there is no such service, or there is some problem with components within the selected service

Parameters:

locator - the locator representing the service to select

r - AsyncRequestor to get notified of the progress of the service selection. null may be given when no notification is required

Returns:

an AsyncResult to monitor the progress of the service selection

Throws:

ResourceNotOwnedException - thrown when the calling application does not have the ownership of this ServiceManager

IllegalArgumentException - thrown when the locator is in a wrong format, or does not designate a valid service

SecurityException - thrown when the calling application does not have permission to select the given service

NullPointerException - thrown when the locator argument is null

IllegalStateException - thrown when this ServiceManager is in DISPOSED state

---

setComponents

```
public abstract AsyncResult setComponents(ServiceComponent[] components,
                                          AsyncRequestor r)
                                   throws ResourceNotOwnedException,
                                          InsufficientResourceException,
                                          SecurityException,
                                          NullPointerException,
                                          IllegalStateException
```

Sets the components to be presented within the current service. When there is any change in the list of components to be presented, those being presented are kept as they are as much as possible. Thus the presentation of those remaining in the list is not affected, if possible.

This method returns immediately, and the compnoent setting is performed asynchronously. The actual progress can be monitored and controlled with AsyncResult returned from this method. Upon the completion of the component setting, that is, when the components are successfully displayed generating PRESENTATION_STARTED, AsyncResult.get() returns null. If there is not enough resource to present the selected components, and it can be detected immediately, InsufficientResourceException is thrown. Otherwise, AsyncResult.get() will throw one of the following exceptions depending on the cause of the failure.

1. InsufficientResourceException: when it was not possible to acquire all of the resources required for the presentation
2. CARefusalException: when CAS refused to present some of the selected components
3. MediaException: when there is any problem with the components set for presentation

When the component setting was failed, ServiceManager moves into STOPPED state.

If there is any update in the list of components in the current service, the component setting is canceled and COMPONENTS_IDENTIFIED event is delivered, whether the setting would be successful or not.

Parameters:

components - components to be presented

r - AsyncRequestor to be notified of the progress of the component setting. If no notification is required, null may be specified

Returns:

AsyncResult to track and control the progress of the component setting

Throws:

ResourceNotOwnedException - thrown when the calling application does not have the ownership of this ServiceManager

InsufficientResourceException - thrown when there is not enough resource for the compnoent setting

SecurityException - thrown when the calling application does not have permission to call this method

NullPointerException - thrown when the components given is null

IllegalStateException - thrown when this ServiceManager is in one of SERVICE_ACQUIRED, PREPARING_MEDIA, and PRESENTING states

---

stop

```
public abstract AsyncResult stop(AsyncRequestor r)
                         throws ResourceNotOwnedException,
                                SecurityException,
                                IllegalStateException
```

Stops the presentation of the current service. This method returns immediately, and the progress can be monitored and controlled with AsyncResult and AsyncRequestor associated with each call to this method. When completed, AsyncResult.get() returns null.

Parameters:

r - AsyncRequestor to be notified of the progress of the operation of this method. null may be specified if no notification is required

Throws:

ResourceNotOwnedException - thrown when the calling application does not have the ownership of this ServiceManager

SecurityException - thrown when the calling application does not have permission to stop the presentation of the current service

IllegalStateException - thrown when this ServiceManager has been disposed

---

addListener

```
public abstract void addListener(ServiceManagerListener l)
                         throws IllegalStateException
```

Adds a listener to be notified of changes in this ServiceManager. If null is given, it is silently ignored without throwing any exception.

Parameters:

l - the listener to add

Throws:

IllegalStateException - thrown when this ServiceManager has been disposed

---

removeListener

```
public abstract void removeListener(ServiceManagerListener l)
                            throws IllegalStateException
```

Removes the given listener registered previous. If `null` is specified or the listener was not added to this `ServiceManager`, it will be ignored silently without throwing any exception.

Parameters:

`l` - the listener to remove

Throws:

`IllegalStateException` - thrown when this `ServiceManager` has been disposed

---

setPreference

```
public abstract Object setPreference(String key,
                                     Object value)
                              throws NullPointerException,
                                     IllegalArgumentException,
                                     IllegalStateException
```

Sets a preference.

Parameters:

`key` - preference key

`value` - preference value

Returns:

returns the previous value for the preference if there was one

Throws:

`NullPointerException` - thrown when `key` is `null`

`IllegalArgumentException` - thrown the `key` parameter is not supported by this `ServiceManager`

`IllegalStateException` - thrown when this `ServiceManager` has been disposed

---

getPreference

```
public abstract Object getPreference(String key)
                              throws NullPointerException,
                                     IllegalArgumentException,
                                     IllegalStateException
```

Returns the preference value associated with the given key.

Parameters:

`key` - preference key

Returns:

the value associated with the given key

Throws:

`NullPointerException` - thrown when `key` is `null`

`IllegalArgumentException` - thrown the `key` parameter is not supported by this `ServiceManager`

`IllegalStateException` - thrown when this `ServiceManager` has been disposed

Interface ServiceManagerListener

dmb.service

---

public interface **ServiceManagerListener**

An interface to be implemented by an object to be notified of changes in `ServiceManager`.

Field Summary		Page
String	**APP_LIST_CHANGED** An event indicating that the list of service- or event-bound applications bound to the current service being presented.	155
String	**COMPONENTS_IDENTIFIED** An event indicating that list of components in the current service is either first obtained or updated.	154
String	**COMPONENTS_SET** An event indicating that the list of components to be presented is determined.	154
String	**MANAGER_DISPOSED** An event indicating that `SerivceManager` has been disposed.	153
String	**MANAGER_STOPPED** An event indicating that the presentation of the previous service has been stopped.	155
String	**PLAYERS_REALIZED** An event indicating that all the `Player`s are prepared for the presentation of the current set of components in the current service.	154
String	**PRESENTATION_STARTED** An event indicating that the presentation of the selected components just began.	155
String	**REASON_ALTERNATIVE_CONTENT** The current service is to be replaced with an alternative content for some reason (a reason code).	156

String	**REASON_APPLICATION_REQUESTED** Application has requested and resulted in the corresponding event (a reason code).	*155*
String	**REASON_EQUIVALENT_SERVICE** Transition to an equivalent service has begun (a reason code).	*156*
String	**REASON_NO_RIGHT** No proper right is obtained for the presentation of the current service (a reason code).	*157*
String	**REASON_NORMAL_CONTENT** Selection of the original content has begun (a reason code).	*156*
String	**REASON_OTHER** Unknown reason (a reason code).	*157*
String	**REASON_PREVIOUS_SELECTION_FAILED** Selection of the previous service has been failed (a reason code).	*155*
String	**REASON_RESOURCE_UNAVAILABLE** Some of resources required for the presentation of the current service are no longer available (a reason code).	*157*
String	**REASON_SERVICE_UNAVAILABLE** The current service is no longer available for some reasons (a reason code).	*157*
String	**REASON_SWITCH_FORCED** For some reasons such as emergency broadcasting, automatic transition to another service has begun (a reason code).	*156*
String	**SELECTION_INITIATED** An event indicating that a selection of a new service has begun.	*153*

Method Summary		*Page*
void	**managerUpdate**(ServiceManager s, String eventType, Object data) Called when there is a change in ServiceManager to which this listener has been added.	*158*

Field Detail

MANAGER_DISPOSED

```
public static final String MANAGER_DISPOSED = "managerDisposed"
```

An event indicating that SerivceManager has been disposed. The corresponding event data is always null.

SELECTION_INITIATED

```
public static final String SELECTION_INITIATED = "selectionInitiated"
```

An event indicating that a selection of a new service has begun. This event is generated, in most cases, as a result of a call to `ServiceManager.select(String,AsyncRequestor)`. But in some cases, this may be delivered automatically. For instance, when a preview period ends, the receiver may select a promotional service.

When this event is delivered to a listener, `ServiceMananger.getService()` returns the service being selected. This event entails an appropriate reason code relevant to the event. So the last argument to `managerUpdate(ServiceManager,String,Object)` may be compared to reason codes defined in this interface.

COMPONENTS_IDENTIFIED

```
public static final String COMPONENTS_IDENTIFIED = "componentsIdentified"
```

An event indicating that list of components in the current service is either first obtained or updated. This event is only delivered to an application owning the corresponding `ServiceManager`, and may be generated at any time, more than once after a delivery of `SELECTION_INITIATED` event. `ServiceManager.getAllComponents()` returns all the components in the current service upon delivery of this event.

If this event is generated while selecting a service, the presentation of the previous service shall be stopped. And if a service was being presented, then the receiver tries its best to maintain the presentation of the service. It is guaranteed that there is no change in the presentation of the current service util all the listener methods return. So within listener methods, setComponents may be called to set the list of components to be presented. If listeners do not set components to present, the default selection will be used, and the default selection is passed to listeners as the third argument to `managerUpdate(ServiceManager,String,Object)`. It is an array of `ServiceComponent`s. If no listener calls setComponents, then the default selection of components will be set for presentation by the underlying receiver implementation.

COMPONENTS_SET

```
public static final String COMPONENTS_SET = "componentsSet"
```

An event indicating that the list of components to be presented is determined. The third argument to `managerUpdate(ServiceManager,String,Object)` carries a relevant reason code, and the final list of components can be obtained with `ServiceManager.getSelectedComponents()`.

PLAYERS_REALIZED

```
public static final String PLAYERS_REALIZED = "playersRealized"
```

An event indicating that all the `Player`s are prepared for the presentation of the current set of components in the current service. This event is delivered only to an application owning the corresponding `ServiceManager` only when there is at least one `Player` newly created. An array of the `Player`s are passed as the third argument to `managerUpdate(ServiceManager,String,Object)`, and they are all in REALIZED state. Listeners may obtain controls such as `BackgroundVideoControl` to initialize and set up the `Player`s. The `Player`s are started automatically when all the listener methods are called and returned. If an application are to set bounds of video presentation, then it should respond to this method.

PRESENTATION_STARTED

```
public static final String PRESENTATION_STARTED = "presentationStarted"
```

An event indicating that the presentation of the selected components just began. There is no data associated with this event, so the third argument to managerUpdate(ServiceManager,String,Object) is always null.

APP_LIST_CHANGED

```
public static final String APP_LIST_CHANGED = "appListChanged"
```

An event indicating that the list of service- or event-bound applications bound to the current service being presented. The corresponding event data is always null. If this event received, ServiceManager.getServiceBoundApps() and ServiceManager.getEventBoundApps() have to be called again to get new list of applications, whether the list was previously retrieved or not.

MANAGER_STOPPED

```
public static final String MANAGER_STOPPED = "managerStopped"
```

An event indicating that the presentation of the previous service has been stopped. This event entails an appropriate reason code describing the cause of this event. The third argument to managerUpdate(ServiceManager,String,Object) may be compared with appropriate reason codes defined in this interface.

REASON_PREVIOUS_SELECTION_FAILED

```
public static final String REASON_PREVIOUS_SELECTION_FAILED = "previousSelectionFailed"
```

Selection of the previous service has been failed (a reason code).

See Also:

MANAGER_STOPPED

REASON_APPLICATION_REQUESTED

```
public static final String REASON_APPLICATION_REQUESTED = "applicationRequested"
```

Application has requested and resulted in the corresponding event (a reason code).

See Also:

SELECTION_INITIATED, COMPONENTS_SET, MANAGER_STOPPED

REASON_ALTERNATIVE_CONTENT

```
public static final String REASON_ALTERNATIVE_CONTENT = "alternativeContent"
```

The current service is to be replaced with an alternative content for some reason (a reason code). This reason code is specified in the following conditions:

- The previous selection incurred selection of an alternative content, where the alternative content may be another service, components that is not being presented, a purchase dialogue, or a promotional material.
- The current components being presented have been replaced with other alternative content, for example, when a preview period ends.

See Also:

SELECTION_INITIATED, COMPONENTS_SET

REASON_NORMAL_CONTENT

```
public static final String REASON_NORMAL_CONTENT = "normalContent"
```

Selection of the original content has begun (a reason code). This reason code may be specified in the following condition.

While presenting an alternative content, transition to the original content has been triggered. For example, as an alternative content, a promotional material and a purchase dialogue may have been presented. But when the user completes the purchase, transition to the original content should be started automatically.

See Also:

SELECTION_INITIATED, COMPONENTS_SET

REASON_EQUIVALENT_SERVICE

```
public static final String REASON_EQUIVALENT_SERVICE = "equivalentService"
```

Transition to an equivalent service has begun (a reason code). As a example, the current service may not keep presented because of weak signal. In that case, the receiver may trigger selection of another service the signal of which is strong enough and carries the same programs.

See Also:

SELECTION_INITIATED

REASON_SWITCH_FORCED

```
public static final String REASON_SWITCH_FORCED = "switchForced"
```

For some reasons such as emergency broadcasting, automatic transition to another service has begun (a reason code).

See Also:

SELECTION_INITIATED

REASON_SERVICE_UNAVAILABLE

```
public static final String REASON_SERVICE_UNAVAILABLE = "serviceUnavailable"
```

The current service is no longer available for some reasons (a reason code). It may be no longer transmitted, or the signal is lost.

See Also:

MANAGER_STOPPED

REASON_RESOURCE_UNAVAILABLE

```
public static final String REASON_RESOURCE_UNAVAILABLE = "resourcesUnavailable"
```

Some of resources required for the presentation of the current service are no longer available (a reason code). For example, a tuner used for the presentation of the current service may be tuned away by a cause other than the service selection.

See Also:

MANAGER_STOPPED

REASON_NO_RIGHT

```
public static final String REASON_NO_RIGHT = "noRight"
```

No proper right is obtained for the presentation of the current service (a reason code).

See Also:

MANAGER_STOPPED

REASON_OTHER

```
public static final String REASON_OTHER = "other"
```

Unknown reason (a reason code).

Method Detail

managerUpdate

```
public void managerUpdate(ServiceManager s,
                          String eventType,
                          Object data)
```

Called when there is a change in `ServiceManager` to which this listener has been added.

Parameters:

`s` - `ServiceManager` from which the event is originated

`eventType` - the event type. One of the events defined in this interface

`data` - data associated with each event. It is different from event to event. Refer to the description on each event

Package dmb.si

Provides APIs for giving access to service information managed by the underlying receiver implementation.

See:

Description

Interface Summary		Page
SIAttribute	Represents an attribute belonging to a specific type of `SIObject`.	*160*
SIAttributeSet	Represents the set of attributes for a type of `SIObject`.	*168*
SIChangeListener	A listener interface to be implemented by listeners that need to get notified of changes in `SIView`s.	*169*
SIObject	Represents an object in the SI database.	*175*
SIQuery	Represents a query to be submitted against `SIView`s.	*177*
SIView	Represents a set of objects satisfying a specific condition.	*178*

Class Summary		Page
SIDatabase	Represents a database of service information.	*171*

Exception Summary		Page
InvalidQueryException	An exception thrown when a query can not be created or executed.	*159*

Package dmb.si Description

Provides APIs for giving access to service information managed by the underlying receiver implementation. The APIs defined in this package provide a database consisting of `SIObject`s, which extends `AttributedObject`. Applications such as EPG may submit queries to retrieve a set of `SIObject`s of their interest. This API is largely independent of the underlying schema supported by each system, and aiming to provide a generic framework for representing and easily retrieving service informations obeying various schema. New additional objects or totally new schema may be used within the framework without changing the API design.

Class InvalidQueryException

dmb.si

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ dmb.si.InvalidQueryException
```

All Implemented Interfaces:

Serializable

public class **InvalidQueryException**

extends Exception

An exception thrown when a query can not be created or executed.

Constructor Summary	Page
`InvalidQueryException()` Creates an instance of this exeption without the reason for this exception.	160
`InvalidQueryException(String reason)` Creates an instance of this exception with the reason that caused this exception to be thrown.	160

Constructor Detail

InvalidQueryException

```
public InvalidQueryException()
```

Creates an instance of this exeption without the reason for this exception.

InvalidQueryException

```
public InvalidQueryException(String reason)
```

Creates an instance of this exception with the reason that caused this exception to be thrown.

Interface SIAttribute

dmb.si

public interface **SIAttribute**

Represents an attribute belonging to a specific type of `SIObject`. This is used in `SIQuery`s for representing the value of an attribute.

Field Summary		Page
String	**TYPE** An attribute representing the type of an `SIObject`.	162

Method Summary		Page
SIQuery	**after**(Date d) Creates an SIQuery that is satisfied when the value of this Date type attribute represents a Date after the given Date.	166
SIQuery	**before**(Date d) Creates an SIQuery that is satisfied when the value of this Date type attribute represents a Date before the given Date.	166
SIQuery	**contains**(Object o) Creates an SIQuery that is satisfied when an element of this attribute contains the given object.	168
SIQuery	**equalTo**(int v) Creates an SIQuery that is satisfied when the value of this int type attribute is equal to the specified value.	164
SIQuery	**equalTo**(Object o) Creates an SIQuery that is satisfied when the value of this attribute is equal to the given object.	166
SIQuery	**equalTo**(long v) Creates an SIQuery that is satisfied when the value of this long type attribute is equal to the specified value.	165
SIQuery	**greaterThan**(int v) Creates an SIQuery that is satisfied when the value of this int type attribute is greater than the specified value.	162
SIQuery	**greaterThan**(long v) Creates an SIQuery that is satisfied when the value of this long type attribute is greater than the specified value.	164
SIQuery	**greaterThanOrEqualTo**(int v) Creates an SIQuery that is satisfied when the value of this int type attribute is greater than or equal to the specified value.	163
SIQuery	**greaterThanOrEqualTo**(long v) Creates an SIQuery that is satisfied when the value of this long type attribute is greater than or equal to the specified value.	165
SIQuery	**isFalse**() Creates an SIQuery that is satisfied when the value of this boolean attribute is false.	167
SIQuery	**isTrue**() Creates an SIQuery that is satisfied when the value of this boolean attribute is true.	167
SIQuery	**lessThan**(int v) Creates an SIQuery that is satisfied when the value of this int type attribute is less than the specified value.	162
SIQuery	**lessThan**(long v) Creates an SIQuery that is satisfied when the value of this long type attribute is less than the specified value.	164

SIQuery	**lessThanOrEqualTo**(int v) Creates an SIQuery that is satisfied when the value of this int type attribute is less than or equal to the specified value.	*163*
SIQuery	**lessThanOrEqualTo**(long v) Creates an SIQuery that is satisfied when the value of this long type attribute is less than or equal to the specified value.	*165*
SIQuery	**notEqualTo**(Object o) Creates an SIQuery that is satisfied when the value of this attribute is not equal to the given object.	*167*
SIQuery	**startsWith**(String o) Creates an SIQuery that is satisfied when the value of this String attribute starts with the given string.	*168*

Field Detail

TYPE

```
public static final String TYPE = "type"
```

An attribute representing the type of an SIObject. The type of values of this attribute is Object, each of which represents a specific type.

Method Detail

lessThan

```
public SIQuery lessThan(int v)
                 throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this int type attribute is less than the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an int one

---

greaterThan

```
public SIQuery greaterThan(int v)
                    throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this int type attribute is greater than the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an int one

---

lessThanOrEqualTo

```
public SIQuery lessThanOrEqualTo(int v)
                          throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this int type attribute is less than or equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an int one

---

greaterThanOrEqualTo

```
public SIQuery greaterThanOrEqualTo(int v)
                             throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this int type attribute is greater than or equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an int one

---

equalTo

```
public SIQuery equalTo(int v)
                throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this int type attribute is equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an int one

---

lessThan

```
public SIQuery lessThan(long v)
                 throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this long type attribute is less than the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an long one

---

greaterThan

```
public SIQuery greaterThan(long v)
                    throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this long type attribute is greater than the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an long one

lessThanOrEqualTo

```
public SIQuery lessThanOrEqualTo(long v)
                          throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this long type attribute is less than or equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an long one

---

greaterThanOrEqualTo

```
public SIQuery greaterThanOrEqualTo(long v)
                             throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this long type attribute is greater than or equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an long one

---

equalTo

```
public SIQuery equalTo(long v)
                throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this long type attribute is equal to the specified value.

Parameters:

v - the value to compare with that of this attribute

Returns:

the created SIQuery object

Throws:

InvalidQueryException - thrown when this attribute is not an long one

---

before

```
public SIQuery before(Date d)
              throws InvalidQueryException,
                     NullPointerException
```

Creates an SIQuery that is satisfied when the value of this Date type attribute represents a Date before the given Date.

Parameters:

d - the Date to compare

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not Date

NullPointerException - thrown when the given d parameter is null

---

after

```
public SIQuery after(Date d)
              throws InvalidQueryException,
                     NullPointerException
```

Creates an SIQuery that is satisfied when the value of this Date type attribute represents a Date after the given Date.

Parameters:

d - the Date to compare

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not Date

NullPointerException - thrown when the given d parameter is null

---

equalTo

```
public SIQuery equalTo(Object o)
              throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this attribute is equal to the given object.

Parameters:

o - an object to compare

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of values of this attribute is not an Object

---

notEqualTo

```
public SIQuery notEqualTo(Object o)
                   throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this attribute is not equal to the given object.

Parameters:

o - an object to compare

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of values of this attribute is not an Object

---

isTrue

```
public SIQuery isTrue()
              throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this boolean attribute is true.

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not boolean

---

isFalse

```
public SIQuery isFalse()
               throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this boolean attribute is false.

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not boolean

---

startsWith

```
public SIQuery startsWith(String o)
                   throws InvalidQueryException
```

Creates an SIQuery that is satisfied when the value of this String attribute starts with the given string.

Parameters:

o - the prefix string

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not String

---

contains

```
public SIQuery contains(Object o)
                 throws InvalidQueryException
```

Creates an SIQuery that is satisfied when an element of this attribute contains the given object.

Parameters:

o - the object to check in the array that is a value of this attribute

Returns:

the created SIQuery

Throws:

InvalidQueryException - thrown when the type of this attribute is not an array of Object

Interface SIAttributeSet

dmb.si

---

public interface **SIAttributeSet**

Represents the set of attributes for a type of SIObject. An instance of this interface is used to create SIAttribute objects, and they are used when constructing SIQuerys. There is a specific SIAttributeSet object per each type of SIObjects. For a SIQuery, all the SIAttribute used in it must be from the same SIAttributeSet.

Method Summary		Page
SIAttribute	**get**(String a) Returns an SIAttribute representing an attribute of the given name.	169

Method Detail

get

```
public SIAttribute get(String a)
                throws InvalidAttributeException
```

Returns an SIAttribute representing an attribute of the given name.

Parameters:

a - name of the attribute

Returns:

an SIAttribute object corresponding to the given attribute

Throws:

InvalidAttributeException - thrown when there is no such attribute

NullPointerException - thrown when the given a parameter is null

Interface SIChangeListener

dmb.si

public interface **SIChangeListener**

A listener interface to be implemented by listeners that need to get notified of changes in SIViews. Receiver implementations are supposed to detect changes in SIViews, but may not find out the exact kinds of changes (addition, removal, and change). In this case, an implementation shall generate a BULK_UPDATE event.

Field Summary		Page
String	**BULK_UPDATE** An event generated when there was a radical change in an SIView.	170

String	**OBJECTS_ADDED** An event generated when SI objects are added to an SIView.	*170*
String	**OBJECTS_CHANGED** An event generated when there is any change in the objects within an SIView.	*171*
String	**OBJECTS_REMOVED** An event generated when SI objects are removed from an SIView.	*170*

Method Summary		Page
void	**siUpdate**(SIView source, String type, SIObject[] data) Called when there is any change in an SIView.	*171*

Field Detail

BULK_UPDATE

```
public static final String BULK_UPDATE = "bulkUpdate"
```

An event generated when there was a radical change in an SIView. Upon receiving this event, the application must re-execute queries to retrieve SI objects used for displaying SI. Once an event of this type is generated, no separate OBJECTS_ADDED, OBJECTS_REMOVED, and/or OBJECTS_CHANGED events are generated for the change reported by the event. The last argument to siUpdate(SIView,String,SIObject[]) is null for this type of events.

OBJECTS_ADDED

```
public static final String OBJECTS_ADDED = "objectsAdded"
```

An event generated when SI objects are added to an SIView. The added objects are passed to siUpdate(SIView,String,SIObject[]) as the last argument.

OBJECTS_REMOVED

```
public static final String OBJECTS_REMOVED = "objectsRemoved"
```

An event generated when SI objects are removed from an SIView. When programs end, rather than having a change in the underlying SI database, then this event is not generated. The last argument to siUpdate(SIView,String,SIObject[]) is list of the removed objects. Note that the returned objects are already removed from the underlying data when they are passed to registered listeners. Therefore it may not possible to get the values of attributes the type of which is SIView.

OBJECTS_CHANGED

```
public static final String OBJECTS_CHANGED = "objectsChanged"
```

An event generated when there is any change in the objects within an SIView. List of the changed objects is passed as the last argument to siUpdate(SIView,String,SIObject[]).

Method Detail

siUpdate

```
public void siUpdate(SIView source,
                     String type,
                     SIObject[] data)
```

Called when there is any change in an SIView. But a call to this method does not guarantee that there really is any change in an SIView. That is, this method is guaranteed to be called, when there is any change in an SIView, but there may be false alarms. Depending on implementations, they may try to reduce or eliminate such false alarms.

Parameters:

source - the SIView containing possible changes

type - the type of the event. One of BULK_UPDATE, OBJECTS_ADDED, OBJECTS_REMOVED, and OBJECTS_CHANGED

data - list of SIObjects related to the event

Class SIDatabase

dmb.si

java.lang.Object

└ **dmb.si.SIDatabase**

All Implemented Interfaces:
AttributedObject

---

abstract public class **SIDatabase**

extends Object

implements AttributedObject

Represents a database of service information. Each database may be constructed from single source or merged from multiple sources. As such, each database may describe completely different set of services and events, or in some cases, there may be more than one databases describing the same set of services and events.

SIDatabase implements AttributedObject. This is for retrieving database-wide attributes such as list of genres.

Constructor Summary		Page
protected	**SIDatabase**() Creates a SIDatabase object.	172

Method Summary		Page
abstract AsyncResult	**forceUpdate**(AsyncRequestor r) Invalidates all the previously-cached service information and starts re-caching the information from the beginning.	175
abstract SIAttributeSet	**getAttributeSet**(Object type) Returns an SIAttributeSet object representing all the attributes associated with SIObjects of the given type.	175
static SIDatabase[]	**getDatabases**() Returns all the SIDatabase available in the device.	173
static SIDatabase	**getDefault**() Returns the system-default SIDatabase instance.	173
abstract SIObject	**getLocatedObject**(String locator) Returns an SIObject represented by the given locator.	174
abstract String	**getPreferredLanguage**() Returns the preferred language for the information retrieved from this SIDatabase.	174
abstract String	**getProvider**() Returns the ID of the supplier of this SIDatabase.	173
abstract SIView	**getView**(Object type) Returns an SIView representing all the SIObjects of the given type in this SIDatabase.	174
abstract void	**setPreferredLanguage**(String lang) Sets the preferred language used for values retrieved from this SIDatabase.	173

Methods inherited from interface dmb.util.AttributedObject
getAttributes, getBoolean, getBooleanList, getBytes, getDate, getDateList, getInt, getIntList, getLong, getLongList, getObject, getObjectList, getString, getStringList, isValid

Constructor Detail

SIDatabase

protected **SIDatabase**()

Creates a SIDatabase object. This constructor is added for implementation convenience and evalution of the specification. Therefore, applications are not supposed to use this constructor.

Method Detail

getDatabases

`public static SIDatabase[] getDatabases()`

Returns all the `SIDatabase` available in the device.

Returns:

an array containing all the database instances

---

getDefault

`public static SIDatabase getDefault()`

Returns the system-default `SIDatabase` instance.

Returns:

the default database

---

getProvider

`public abstract String getProvider()`

Returns the ID of the supplier of this `SIDatabase`.

Returns:

the supplier ID

---

setPreferredLanguage

`public abstract void setPreferredLanguage(String lang)`

Sets the preferred language used for values retrieved from this `SIDatabase`. A language code defined in RFC 3066 [15] must be specified. If this method is called, even `SIObject`s created before the call are affected.

Parameters:

`lang` - the language code

Throws:

`NullPointerException` - thrown when the given `lang` parameter is `null`

`IllegalArgumentException` - thrown when the given `lang` parameter is not a valid langauge code or the corresponding language is not supported by the device

getPreferredLanguage

```
public abstract String getPreferredLanguage()
```

Returns the preferred language for the information retrieved from this `SIDatabase`. Returns one of the language codes defined in RFC 3066 [15].

---

getView

```
public abstract SIView getView(Object type)
```

Returns an `SIView` representing all the `SIObject`s of the given type in this `SIDatabase`. Since the returned object is an `SIView`, any change in the underlying database will be reflected to it automatically. Additional queries may be given to the `SIView` to narrow down the set of `SIObject`s. And when done, the current snapshot of the set of `SIObject`s represented by the `SIView` may be retrieved.

Parameters:

`type` - the type of the `SIView`

Returns:

an `SIView` containing all the `SIObject`s of the specified type

Throws:

`IllegalArgumentException` - thrown when there is no such type

`NullPointerException` - thrown when the `type` parameter is `null`

---

getLocatedObject

```
public abstract SIObject getLocatedObject(String locator)
```

Returns an `SIObject` represented by the given locator.

Parameters:

`locator` - the locator for the `SIObject` to retrieve

Returns:

an `SIObject` pointed to by the locator

Throws:

`NullPointerException` - thrown when the `locator` parameter is `null`

`IllegalArgumentException` - thrown when the `locator` parameter is not a valid locator, or there is no `SIObject` corresponding to the given locator

---

getAttributeSet

`public abstract SIAttributeSet getAttributeSet(Object type)`

Returns an `SIAttributeSet` object representing all the attributes associated with `SIObject`s of the given type. The returned `SIAttributeSet` is used to construct queries.

Parameters:

`type` - the type of the `SIObject` for which to retrieve an `SIAttributeSet`

Returns:

the requested `SIAttributeSet`

Throws:

`NullPointerException` - thrown when the given `type` parameter is `null`

`IllegalArgumentException` - thrown when the given `type` parameter does not represent a valid type known to this `SIDatabase`

---

forceUpdate

`public abstract AsyncResult forceUpdate(AsyncRequestor r)`

Invalidates all the previously-cached service information and starts re-caching the information from the beginning.

Parameters:

`r` - `AsyncRequestor` to get notified of the progress of the caching. `null` may be given when no notification is required.

Returns:

an `AsyncResult` to monitor the progress of the caching. `AsyncResult.get()` returns `null` if the service information was successfully cached. Otherwise, it will throw an exception representing the cause of the failure.

Interface SIObject

dmb.si

All Superinterfaces:
AttributedObject

---

public interface **SIObject**

extends AttributedObject

Represents an object in the SI database. All types of SI objects are represented by this object. The type of each object may be identified with value of the SIAttribute.TYPE attribute. Each object has a set of attributes defined by the underlying platform depending on its type. Those attributes may be retrieved with get methods defined in this interface.

An SIObject represents a specific version of SI data. Thus, once obtained, it will return any attribute value cached in the object, regardless of changes in the underlying SI database. And if some attributes are not cached and there was any change in the object, then any attempt to retrieve their values shall result in an IllegalStateException. This behaviour is a safeguard in order not to return different versions of data from a single SIObject causing inconsistencies.

When r297SIObjects representing the same object are retrieved, they may be distinct instances.

Method Summary		Page
SIView	**getView**(String a) Returns value of the given attribute the type of which is SIView.	176

Methods inherited from interface dmb.util.AttributedObject
getAttributes, getBoolean, getBooleanList, getBytes, getDate, getDateList, getInt, getIntList, getLong, getLongList, getObject, getObjectList, getString, getStringList, isValid

Method Detail

getView

```
public SIView getView(String a)
               throws InvalidAttributeException,
                      NullPointerException,
                      IllegalStateException
```

Returns value of the given attribute the type of which is SIView.

Parameters:

a - name of the attribute

Returns:

an SIView that is the value of the attribute

Throws:

InvalidAttributeException - thrown when there is no such attribute in this object, or its type is not SIView

NullPointerException - thrown when the given name is null

IllegalStateException - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

Interface SIQuery

dmb.si

---

public interface **SIQuery**

Represents a query to be submitted against SIViews. A query may specify conditions to meet only on one type of SIObjects. A query can be constructed first by obtaining an SIAttributeSet from an SIDatabase.

```
SIAttributeSet s = SIDatabase.getDefault()
                          .getAttributeSet(Program.TYPE);
// Genre: OriginationCS and L1 are 5 and 7, respectively (Cinema)
SIQuery q1 = s.get(Program.GENRE).startsWith("005.007");
SIQuery q2 = s.get(Program.FREE).isTrue();
SIQuery q = q1.and(q2);

SIView programs = SIDatabase.getDefault().getView(Program.TYPE);
SIObject[] result = programs.createView(q).getSnapshot();
```

Method Summary		Page
SIQuery	**and**(SIQuery q) Creates a query that is satisfied if and only if both this and the given queries are satisfied at the same time.	*177*
SIQuery	**negate**() Creates a query that is satisfied if and only if this query is not satisified.	*178*
SIQuery	**or**(SIQuery q) Creates a query that is satisfied if and only if this or the given query is satisfied.	*178*

Method Detail

and

```
public SIQuery and(SIQuery q)
            throws InvalidQueryException,
                   NullPointerException
```

Creates a query that is satisfied if and only if both this and the given queries are satisfied at the same time.

Parameters:

`q` - a query to combine

Returns:

created query

Throws:

`InvalidQueryException` - thrown when the type of objects for the given query is different from that of this `SIQuery` object

`NullPointerException` - thrown when the `q` parameter is `null`

or

```
public SIQuery or(SIQuery q)
           throws InvalidQueryException,
                  NullPointerException
```

Creates a query that is satisfied if and only if this or the given query is satisfied.

Parameters:

`q` - a query to combine

Returns:

created query

Throws:

`InvalidQueryException` - thrown when the type of objects for the given query is different from that of this `SIQuery` object

`NullPointerException` - thrown when the `q` parameter is `null`

negate

```
public SIQuery negate()
```

Creates a query that is satisfied if and only if this query is not satisified.

Returns:

created query

Interface SIView

dmb.si

public interface **SIView**

Represents a set of objects satisfying a specific condition. A view may be created from another view by calling `createView(SIQuery)` or `createAttributeView(String)`. Also one may be obtained from `SIDatabase.getView(Object)` or `SIObject.getView(String)` methods. Once created, a view reflects the current state of the underlying SI database regardless of updates occurred after its creation.

Method Summary		Page
void	**addSIChangeListener**(SIChangeListener l) Adds the given listener to monitor changes in this view such as additions, removals, and updates to the view contents.	182
SIView	**createAttributeView**(String a) Creates a view representing objects that are values of the given attribute of the objects in this view.	180
SIView	**createView**(SIQuery query) Creates a view representing a set of objects satisfying the given query.	179
SIObject	**getObject**() Returns an SIObject contained in this view.	181
int	**getSize**() Returns the number of SIObjects contained in this view.	180
SIObject[]	**getSnapshot**() Returns all the SIObjects represented by this view at the time of calling this method.	180
SIObject[]	**getSnapshot**(int startIndex, int count) Returns a subset of SIObjects contained in this view by specifying the start index in the list of SIObjects and the number of SIObjects to retrieve.	180
void	**removeSIChangeListener**(SIChangeListener l) Removes the given listener from this view.	182
void	**sort**(String[] sortBy, boolean[] isDescending) Sorts a list of objects represented by this view by the specified order.	181

Method Detail

createView

```
public SIView createView(SIQuery query)
                  throws InvalidQueryException,
                         NullPointerException
```

Creates a view representing a set of objects satisfying the given query.

Parameters:

`query` - the query. This must be one created for objects of the type of this view

Returns:

the created view

Throws:

`InvalidQueryException` - thrown when the type the given query is created for and the type of this view are different

`NullPointerException` - thrown when the given `query` parameter is `null`

createAttributeView

```
public SIView createAttributeView(String a)
                           throws InvalidAttributeException,
                                  NullPointerException
```

Creates a view representing objects that are values of the given attribute of the objects in this view.

Parameters:

`a` - name of the attribute

Returns:

the created view

Throws:

`InvalidAttributeException` - thrown when type of the given attribute is not `SIObject` or r311SIView

`NullPointerException` - thrown when the given `a` parameter is `null`

---

getSize

```
public int getSize()
```

Returns the number of `SIObject`s contained in this view.

Returns:

the number of `SIObject`s. If it is impossible to get a snapshot of this view, returns `-1`

---

getSnapshot

```
public SIObject[] getSnapshot()
```

Returns all the `SIObject`s represented by this view at the time of calling this method.

Returns:

list of `SIObject`s this view contains. In case where the list ca not be retrieved for some reason, returns `null`. This should be distinguished from the case where this view does not contain any object, thus returning 0

---

getSnapshot

```
public SIObject[] getSnapshot(int startIndex,
                              int count)
```

Returns a subset of SIObjects contained in this view by specifying the start index in the list of SIObjects and the number of SIObjects to retrieve.

Parameters:

`startIndex` - the start index in the list of SIObjects in this view, from which SIObjects will be retrieved

`count` - the number of SIObjects to retrieve. If the number of SIObjects is less than the number of SIObjects from the given start index to the end of the list, then all of the available SIObjects are returned

Returns:

An array containing retrieved SIObjects. If there is no object to return, then returns an array of zero-length. If it is impossible to retrieve the designated SIObjects, then returns `null`.

---

getObject

```
public SIObject getObject()
```

Returns an SIObject contained in this view. This method is a convenience method used when it is obvious that this view contains only one object. When there is more than one object within this view, it depends on the implementation what to return from this method.

Returns:

an SIObject this view contains. If there is no object within this view or it is impossible to retrieve objects within this view, returns `null`. Note that this method cannot distinguish between the above mentioned two cases. If such distinction is required, `getSnapshot()` must be used instead

---

sort

```
public void sort(String[] sortBy,
                 boolean[] isDescending)
```

Sorts a list of objects represented by this view by the specified order. The relative ordering of attribute values is defined for the following types: `boolean` (`false` is followed by `true` in ascending order), `int`, `long`, `String` (the order defined by `String.compareTo(String)`), `Date`(the order of `Date.getTime()`s). In other cases, an exception is thrown since there is no defined order between values. And if values are references (that is, objects), and `null` values are retrieved, `null` is considered to come later in ascending order.

Parameters:

`sortBy` - list of attributes in the objects in this view. The first one in the list is considered first when sorting out the result, ties are broken by the next attribute, and the same repeated

`isDescending` - each entry in this array corresponds to an entry in the `sortBy` array at the same index. If an entry in this array is `false` it means the corresponding attribute is consulted and the sort is done in ascending order with respect to the attribute. If `true`, the atttribute is considered in descending order. This array may be shorter than `sortBy`. In that case, missing entries are considered to have `false` meaning those attributes are consulted for sorting in ascending order

Returns:

the list of SIObjects this view contains after sorting them as specified by `sortBy` and `isDescending` parameters. If it is impossible to retrieve objects, the returns `null` rather than 0

Throws:

`InvalidAttributeException` - thrown when the `sortBy` parameter contains non-existing attribute, or there is no order defined for values of any attribute

`NullPointerException` - thrown when the `sortBy` parameter or its elements are `null`, and also when the `isDescending` paraemeter is `null`

---

addSIChangeListener

`public void` **`addSIChangeListener`**`(SIChangeListener l)`

Adds the given listener to monitor changes in this view such as additions, removals, and updates to the view contents. If the given listener is `null`, it is ignored silently without throwing an exception.

Parameters:

`l` - the listener to add

---

removeSIChangeListener

`public void` **`removeSIChangeListener`**`(SIChangeListener l)`

Removes the given listener from this view. If the listener was not added to this view before, or is `null`, then it is simply ignored without incurring any exception.

Parameters:

`l` - the listener to remove

Package dmb.tuning

Defines a set of APIs for controlling tuners available in the receiver.

See:

Description

Interface Summary		Page
TunerListener	An interface to be implemented by an object that needs to get notified of changes in a Tuner.	*187*

Class Summary		Page
Tuner	Represents a tuner in the receiver.	*184*
TunerLock	Represents a state where a tuner is locked on an ensemble.	*188*

Exception Summary		Page
ScanningFailedException	An exception thrown when frequency scanning has been failed for some reason.	*183*
TuningFailedException	An exception thrown when tuning has been failed for some reason.	*190*

Package dmb.tuning Description

Defines a set of APIs for controlling tuners available in the receiver. Tuning is performed via creating a TunerLock, and acquiring it with ResourceManager. And a separate abstraction called Tuner is provided to monitor the state of a tuner, and to do tasks such as service scanning. The reason why there is another abstraction TunerLock in addition to Tuner is that it enables sharing of a tuner among multiple applications if they are required to tune to a single ensemble.

Class ScanningFailedException

dmb.tuning

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ dmb.tuning.ScanningFailedException
```

All Implemented Interfaces:
Serializable

---

public class **ScanningFailedException**

extends Exception

An exception thrown when frequency scanning has been failed for some reason.

Constructor Summary	Page
ScanningFailedException() Creates an instance of this exception without the detailed message.	*184*
ScanningFailedException(String reason) Creates an instance of this exception with the given detailed message.	*184*

Constructor Detail

ScanningFailedException

public **ScanningFailedException**()

Creates an instance of this exception without the detailed message.

ScanningFailedException

public **ScanningFailedException**(String reason)

Creates an instance of this exception with the given detailed message.

Class Tuner

dmb.tuning

java.lang.Object

└ **dmb.tuning.Tuner**

All Implemented Interfaces:

Resource

public class **Tuner**

extends Object

implements Resource

Represents a tuner in the receiver. The tuner may be temporarily deactivated if there has been no TunerLock owned by applications for some time that is implementation dependent, or for some other reasons such as battery shortage. Once deactivated, the tuner may be activated again by a call to scan(int[], boolean, AsyncRequestor) or owning a TunerLock.

Constructor Summary		Page
protected	**Tuner**() Creates an instance of Tuner.	185

Method Summary		Page
void	**addTunerListener**(TunerListener l) Adds a listener to monitor changes in the state of this tuner.	187
static Tuner	**getDefault**() Returns the default tuner in the receiver.	186
String	**getLocator**() Returns the locator representing the broadcast channel this tuner is locked to.	186
int	**getSignalQuality**() Returns the quality of the signal recevied by this tuner.	187
static Tuner[]	**list**() Returns all the tuners available in the receiver.	185
void	**removeTunerListener**(TunerListener l) Removes a listener that was previously added to this tuner for monitoring changes in the state of this tuner.	187
AsyncResult	**scan**(int[] freqs, boolean resetSI, AsyncRequestor r) Scans the specified frequencies for services.	186

Constructor Detail

Tuner

protected **Tuner**()

Creates an instance of Tuner. This constructor is for implementation convenience and evolution of the specification. Therefore, applications must not make use of this constructor.

Method Detail

list

public static Tuner[] **list**()

Returns all the tuners available in the receiver.

Returns:

list of tuners

getDefault

```
public static Tuner getDefault()
```

Returns the default tuner in the receiver.

Returns:

the default tuner

---

scan

```
public AsyncResult scan(int[] freqs,
                        boolean resetSI,
                        AsyncRequestor r)
                 throws ResourceNotOwnedException,
                        SecurityException
```

Scans the specified frequencies for services. If this tuner is not active, it is automatically activated. Prior to calling this method, this tuner must be owned by the calling application, and the application must have appropriate permissions. The relevant permission to be owned for calling this method is `dmb.tuning.scan`.

Parameters:

`freqs` - list of frequencies in Hz

`resetSI` - if `true`, clears SI data including the list of services. If `false`, only information obtained by scaning is updated, and other information is retained as it is

`r` - `AsyncRequestor` to get notified of progresses in the scanning. If no notification is anticipated, then `null` may be specified

Returns:

`AsyncResult` object to query and control the scanning process. Once completed, `AsyncResult.get()` returns `null`. If the scanning fails, a `ScanningFailedException` is thrown

Throws:

`ResourceNotOwnedException` - thrown when this method is called without owning this tuner via `ResourceManager`

`SecurityException` - thrown when the calling application does not have the required permission

---

getLocator

```
public String getLocator()
```

Returns the locator representing the broadcast channel this tuner is locked to. If this tuner is not locked to a channel, this returns `null`.

Returns:

the locator for the channel this tuner is locked to. If this tuner is turned off or not in the locked state, then returns `null`

getSignalQuality

```
public int getSignalQuality()
```

Returns the quality of the signal recevied by this tuner. The quality is an integer in the range of 0 to 100 inclusive, where 0 means no signal at all, and 100 means the signal is strongest possible. Note that there is no guarantee that all the values between 0 and 100 inclusive are returned from this method. Depending on the underlying hardware, only a subset of the values may be returned. This returns 0 when this tuner is currently turned off, or not in the locked state.

Returns:

the signal quality

---

addTunerListener

```
public void addTunerListener(TunerListener l)
```

Adds a listener to monitor changes in the state of this tuner. If the given `l` parameter is `null`, it is silently ignored without causing an exception thrown.

Parameters:

`l` - the listener to add

---

removeTunerListener

```
public void removeTunerListener(TunerListener l)
```

Removes a listener that was previously added to this tuner for monitoring changes in the state of this tuner. If the given listener was not added to this tuner, or the `l` parameter is `null`, it is silently ignored without throwing an exception.

Parameters:

`l` - the listener to remove

Interface TunerListener

dmb.tuning

---

public interface **TunerListener**

An interface to be implemented by an object that needs to get notified of changes in a `Tuner`. Such change includes that of signal quality received by a `Tuner`.

Method Summary		Page
void	**signalQualityChanged**(Tuner tuner, int quality) Called when there is change in the quality of signal received from the given Tuner.	188
void	**tunedTo**(Tuner tuner, String locator) Called when the given Tuner is tuned to an ensemble represented by the given locator.	188

Method Detail

tunedTo

```
public void tunedTo(Tuner tuner,
                    String locator)
```

Called when the given Tuner is tuned to an ensemble represented by the given locator.

Parameters:

tuner - the Tuner tuned to other ensemble

locator - the locator pointing to the ensemble. null is specified when the Tuner is turned off

---

signalQualityChanged

```
public void signalQualityChanged(Tuner tuner,
                                 int quality)
```

Called when there is change in the quality of signal received from the given Tuner. Note that this is not called when a Tuner is turned off.

Parameters:

tuner - the Tuner

quality - new quality indicator (from 0 to 100 inclusive).

Class TunerLock

dmb.tuning

java.lang.Object

└ **dmb.tuning.TunerLock**

All Implemented Interfaces:

Resource

final public class **TunerLock**

extends Object

implements Resource

Represents a state where a tuner is locked on an ensemble. If acquisition of a `TunerLock` is requested by an application via `ResourceManager`, then a `Tuner` is allocated, and tuning begins. If the tuning succeeds, the `TunerLock` is successfully acquired. If not, `AsyncResult.get()` throws a `TuningFailedException`. Once acquired, the `Tuner` is guaranteed to be locked on the current ensemble as long as the ownership of `TunerLock` is retained. The `Tuner` used for a `TunerLock` is not specified explicitly. Instead, the receiver implmentation picks appropriate one. If there is no `Tuner` that can satisfy the requirement set by a `TunerLock`, then a `Tuner` associated with `TunerLock`(s), all of which are held by applications with lower priorities than that of the `TunerLock` being acquired. If none has a lower priority, then acquisition of the `TunerLock` fails.

For tuning, applications must have appropriate permissions. The permissions relevant to tuning are as follows:

- `dmb.tuning.tune.<locator>`: Here <locator> represents a set of ensembles an application can tune to. Within <locator> string, wildcards specified in the main body of the present document can be used to designate a set of ensembles

If more than one application requests tuning to a single ensemble, then regardless of the number of `TunerLock`s, they may share the same `Tuner`.

Constructor Summary	Page
`TunerLock``(String locator)` Creates a `TunerLock` that requests tuning to an ensemble represented by the given locator.	*189*

Method Summary		Page
`String`	**`getLocator`**`()` Returns the locator representing the ensemble to tune to.	*190*
`Tuner`	**`getTuner`**`()` Returns the `Tuner` that is allocated to meet the requirement set by this `TunerLock`.	*190*

Constructor Detail

TunerLock

```
public TunerLock(String locator)
          throws IllegalArgumentException,
                 SecurityException,
                 NullPointerException
```

Creates a `TunerLock` that requests tuning to an ensemble represented by the given locator. If it is acquired by the application via `ResourceManager`, a `Tuner` is allocated, and the tuning completes. If tuning fails while a `TunerLock` is being acquired, a `TuningFailedException` is thrown from `AsyncResult.get()` or `AsyncResult.complete()`.

Throws:

`IllegalArgumentException`

`SecurityException`

`NullPointerException`

Method Detail

getTuner

`public Tuner **getTuner**()`

Returns the `Tuner` that is allocated to meet the requirement set by this `TunerLock`.

Returns:

the `Tuner`. If this `TunerLock` is not in the acquired state, the returns `null`

---

getLocator

`public String **getLocator**()`

Returns the locator representing the ensemble to tune to.

Returns:

the locator pointing to an ensemble

Class TuningFailedException

dmb.tuning

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ dmb.tuning.TuningFailedException
```

All Implemented Interfaces:
Serializable

---

public class **TuningFailedException**

extends Exception

An exception thrown when tuning has been failed for some reason.

Constructor Summary	Page
TuningFailedException() Creates an instance of this exception without the detailed message.	191
TuningFailedException(String reason) Creates an instance of this exception with the given detailed message.	191

Constructor Detail

TuningFailedException

```
public TuningFailedException()
```

Creates an instance of this exception without the detailed message.

TuningFailedException

```
public TuningFailedException(String reason)
```

Creates an instance of this exception with the given detailed message.

Package dmb.ui

This package provides a UI extension to `javax.microedition.lcdui` to handle the peculiarities in DMB environment such as support for transparent graphics plane.

See:

Description

Class Summary		Page
AlphaAttribute	Represents the attributes of the graphic context, which are relevant to the alpha component and the composite mode of an associated `Graphics` object.	192
DisplayControl	Controls the z-order of the `Display` and the key focus associated with an application.	196
DMBCanvas	Represents the basic drawing surface which is laid over the video presentation.	200
DMBItem	The base class for UI components that can be added to a `DMBCanvas`.	214
FontLoader	Loads new fonts from an `InputStream` or application resource.	217
KeyLock	Represents the right to exclusively receive events generated by the associated set of keys.	218
TextItem	A `DMBItem` subclass providing text editing capability.	221
UserItem	A `DMBItem` subclass that may be subclassed and customized.	228

Package dmb.ui Description

This package provides a UI extension to `javax.microedition.lcdui` to handle the peculiarities in DMB environment such as support for transparent graphics plane. In DMB environment, more than one application may be running simultaneously. The UI system designates one application as the foreground application, and others are treated as background ones. The foreground application obtains the key focus.

In DMB environment, more than one application represent themselves on the display at the same time, and may need to receive key events associated with the keys of their interest even if it does not have the key focus, or in some cases, it does not have a UI component at all. The extension defined in this package deals with these peculiarities.

Class AlphaAttribute

dmb.ui

`java.lang.Object`

`└ dmb.ui.AlphaAttribute`

---

public class **AlphaAttribute**

extends Object

Represents the attributes of the graphic context, which are relevant to the alpha component and the composite mode of an associated `Graphics` object.

The drawing surface represented by a Graphics object does not have an alpha component. The purpose of this object is to provide the control over the alpha component of the drawing surface used in DMB environment. With this object, the current alpha value and the current composite mode may be specified to affect the rendering methods in the associated Graphics object.

Alpha Value

AlphaAttribute allows for the specification of an alpha value ranging from 0 to 255 inclusive, in addition to RGB values that may be specified with Graphics object. The alpha value specified with setAlphaValue(int) is applied to most of drawing methods of the corresponding Graphics object. But, methods such as drawImage, drawRegion, and drawRGB, which may entail separate alpha values in the source, are not affected by the current alpha value set with setAlphaValue(int). In case the source of the methoes does not have an alpha channel, then they are assumed to be fully opaque, that is, an alpha value of 255.

Alpha Composite Modes

AlphaAttribute object allows for setting of the alpha composite mode. The mode is applied to most of operations performed on the associated Graphics object. Among the Porter-Duff rules, CLEAR, SRC, and SRCOVER rules are supported. CLEAR sets all of the components in the destination pixels to zero, thus making the destination fully transparent, and SRC copies all the components of the source pixels including the alpha component to the destination. SRCOVER is the default rule, and works exactly as the way Graphics does in LCD UI. But it considers the alpha channel of the destination surface different from the case of usual MIDP environment where the destination does not have an alpha channel. For the details on the alpha composite modes specified here, refer to the documentation for java.awt.AlphaComposite, which is a part of Java Standard Edition.

Lifecycle

An AlphaAttribute instance may be obtained by calling getAlphaAttribute(Graphics) method with a Graphics object. Once created, it will be valid exactly while the corresponding Graphics object is valid. If getAlphaAttribute(Graphics) is invoked more than once with the same Graphics object, the returned AlphaAttribute object will be identical between the calls. And if it is created from a Graphics object passed into various paint(Graphics) methods, then the AlphaAttribute object will last for the duration of executing paint(Graphics). If a rendering surface associated with a Graphics object is an opaque surface without an alpha channel, a new alpha channel is created when the associated AlphaAttribute is created, and the surface is initialized to be fully transparent. The default alpha value for the newly obtained AlphaAttribute is 255, meaning fully opaque, and the default composite mode is SRCOVER.

Field Summary		Page
static final int	**CLEAR** A composite rule setting all of the compomponents in the destination to zero (Porter-Duff Clear rule).	194
static final int	**SRC** A composite rule copying the components in the source pixels to the destination pixels (Porter-Duff Source rule).	194
static final int	**SRCOVER** A composite rule determining the values of the destination pixels based on both the values of the source and the destination pixels (Porter-Duff Source Over Destination rule).	194

Method Summary		Page
static AlphaAttribute	**getAlphaAttribute**(Graphics g) Returns an AlphaAttribute object associated with the given Graphics object.	195
int	**getAlphaValue**() Returns the current alpha value for the associated Graphics object.	196
int	**getComposite**() Returns the current alpha composite mode.	195
void	**setAlphaValue**(int alpha) Sets the alpha value that will affect graphics operations performed with the associated Graphics object.	196
void	**setComposite**(int compRule) Sets the alpha composite mode for the Graphics object associated with this AlphaAttribute object.	195

Field Detail

CLEAR

```
public static final int CLEAR = 1
```

A composite rule setting all of the compomponents in the destination to zero (Porter-Duff Clear rule). This composite mode does not use both source and destination values.

SRC

```
public static final int SRC = 2
```

A composite rule copying the components in the source pixels to the destination pixels (Porter-Duff Source rule). The desitination value is not used when drawing is performed.

SRCOVER

```
public static final int SRCOVER = 0
```

A composite rule determining the values of the destination pixels based on both the values of the source and the destination pixels (Porter-Duff Source Over Destination rule). Refer to java.awt.AlphaComposite in Java Standard Edition for details on this composite mode. The implementation of this mode is permitted not be exact, since the exact implementation requires significant computing power.

Method Detail

getAlphaAttribute

```
public static AlphaAttribute getAlphaAttribute(Graphics g)
```

Returns an `AlphaAttribute` object associated with the given `Graphics` object. There is only one `AlphaAttribute` object per `Graphics` object, All the settings done to an `AlphaAttribute` object are valid while the corresponding `Graphics` object is valid.

If there is no alpha channel in the drawing surface associated with the given `Graphics` object, then an alpha channel is created for the surface upon the creation of an `AlphaAttribute`, and the alpha values for the surface are set to zero, thus making it fully transparent.

When first created, the default alpha value is set to 255, and the default composite mode to `SRCOVER`.

Parameters:

`g` - the `Graphics` object

Throws:

`NullPointerException` - if the `g` parameter is `null`

---

setComposite

```
public void setComposite(int compRule)
```

Sets the alpha composite mode for the `Graphics` object associated with this `AlphaAttribute` object. The valid set of modes includes `SRC`, `SRCOVER`, and `CLEAR`. Once set, the mode affects all the following operations performed on the associated `Graphics` object with a few exceptions such as `drawImage` as specified in the class documentation for this class.

Parameters:

`compRule` - the alpha composite mode to set

Throws:

`IllegalArgumentException` - thrown when the `compRule` is invalid

---

getComposite

```
public int getComposite()
```

Returns the current alpha composite mode.

Returns:

the current composite mode. The default value is `SRCOVER`. One of `SRC`, `SRCOVER`, and `CLEAR` may be specified

---

setAlphaValue

`public void` **`setAlphaValue`**`(int alpha)`

Sets the alpha value that will affect graphics operations performed with the associated `Graphics` object. This value affects most of methods called on the associated `Graphics` object. The default value is 255.

Parameters:

`alpha` - an alpha value ranging from 0 to 255 inclusive

Throws:

`IllegalArgumentException` - thrown when the given `alpha` is out of the valid range

---

getAlphaValue

`public int` **`getAlphaValue`**`()`

Returns the current alpha value for the associated `Graphics` object.

Returns:

the current alpha value

Class DisplayControl

dmb.ui

`java.lang.Object`

`└` **`dmb.ui.DisplayControl`**

---

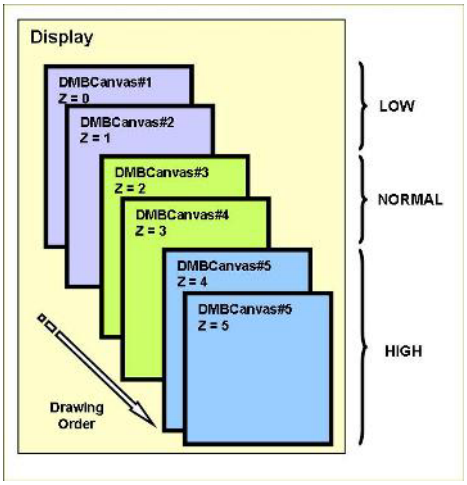
public class **DisplayControl**

extends Object

Controls the z-order of the `Display` and the key focus associated with an application.

Z-Order Control

When more than one applications share the display, this class provides a means to set the z-order of the display of each application. In addition to the z-order, this class classifies applications into three classes of `HIGH`, `NORMAL`, and `LOW`.



As shown in the above picture, each application is assigned a priority that is one of `HIGH`, `NORMAL`, and `LOW`. An application with a higher priority than others is always displayed closer to the viewer. To set the priority of an application, `setPriority(int)` may be invoked. Among applications with identical priority, their relative z-order may be changed by `toFront()` and `toBack()` methods. For instance, if an application should be displayed on the top of other applications, first its priority should be set to `HIGH`, and `toFront()` should be called to bring it to the front among those assigned `HIGH` priority.

Key Focus Management

In some cases, an application may not want to receive keys even when it is topmost. By calling `setFocusable(boolean)`, an application may be set if it wishes to receive key events or not.

Field Summary		Page
static final int	**HIGH** Means the corresponding application has the high priority in terms of the display device.	198
static final int	**LOW** Means the corresponding application has the low priority in terms of the display device.	198
static final int	**NORMAL** Means the corresponding application has the normal priority in terms of the display device.	198

Method Summary		Page
static DisplayControl	**getDisplayControl**(MIDlet m) Returns a `DisplayControl` object for the given `MIDlet`.	198
int	**getPriority**() Returns the current priority of the application associated with this `DisplayControl`.	200
boolean	**isFocusable**() Returns if the associated application will receive the key focus or not.	200
void	**removeDisplayable**() Removes the current `Displayable` (including `DMBCanvas`) set to the `Display` associated with the application, in turn, associated with this `DisplayControl`.	199

void	**setFocusable**(boolean focusable) Sets whether the associated application will receive key events.	*200*
void	**setPriority**(int priority) Sets the display priority for the application associated with this DisplayControl.	*199*
void	**toBack**() Moves the Displayable of the corresponding application to the bottom among the applications with the same priority.	*199*
void	**toFront**() Brings the Displayable of the corresponding application to the front among the applications with the same priority.	*199*

Field Detail

HIGH

```
public static final int HIGH = 268435456
```

Means the corresponding application has the high priority in terms of the display device. Applications with this priority are displayed closest to the viewer, and guaranteed atop other applications with a lower priority.

NORMAL

```
public static final int NORMAL = 536870912
```

Means the corresponding application has the normal priority in terms of the display device. Applications with this priority are displayed under those with HIGH priority, and above those with LOW priority.

LOW

```
public static final int LOW = 805306368
```

Means the corresponding application has the low priority in terms of the display device. Applications with this priority are displayed under applications with a higher priority.

Method Detail

getDisplayControl

```
public static DisplayControl getDisplayControl(MIDlet m)
```

Returns a DisplayControl object for the given MIDlet.

Parameters:

m - the MIDlet instance

Throws:

`NullPointerException` - thrown when the `m` parameter is `null`

---

removeDisplayable

`public void **removeDisplayable**()`

Removes the current `Displayable` (including `DMBCanvas`) set to the `Display` associated with the application, in turn, associated with this `DisplayControl`. After a call to this method, `Display.getCurrent()` returns `null`. Note that calling `Display.setCurrent(Displayable)` with `null` does not remove the current `Displayable`.

---

toFront

`public void **toFront**()`

Brings the `Displayable` of the corresponding application to the front among the applications with the same priority.

---

toBack

`public void **toBack**()`

Moves the `Displayable` of the corresponding application to the bottom among the applications with the same priority.

---

setPriority

`public void **setPriority**(int priority)`

Sets the display priority for the application associated with this `DisplayControl`. The priority must be one of `HIGH`, `NORMAL`, and `LOW`. And the higher the priority is, the closer the display is to the viewer.

Parameters:

`priority` - the priority

Throws:

`IllegalArgumentException` - thrown when the `priority` is not one of `HIGH`, `NORMAL`, and `LOW`

---

getPriority

```
public int getPriority()
```

Returns the current priority of the application associated with this `DisplayControl`. The default value is `NORMAL`.

Returns:

the current priority

---

setFocusable

```
public void setFocusable(boolean focusable)
```

Sets whether the associated application will receive key events. If `true`, the application gets the key focus when it is topmost in the display device. Otherwise, it yields the key focus to the application next to itself in the z-order. And the same rule is applied to the next one. This process is repeated until an application to get the key focus is found. If no application is eligible for owning the key focus, the key focus is set to no application, and key events are just discarded. The default value for this property is `true`.

Parameters:

`focusable` - whether the associated application hopes to receive the key focus or not

---

isFocusable

```
public boolean isFocusable()
```

Returns if the associated application will receive the key focus or not.

Returns:

`true` if it will get the key focus, `false`, otherwise

Class DMBCanvas

dmb.ui

```
java.lang.Object
  └ javax.microedition.lcdui.Displayable
      └ javax.microedition.lcdui.Canvas
          └ dmb.ui.DMBCanvas
```

abstract public class **DMBCanvas**

extends Canvas

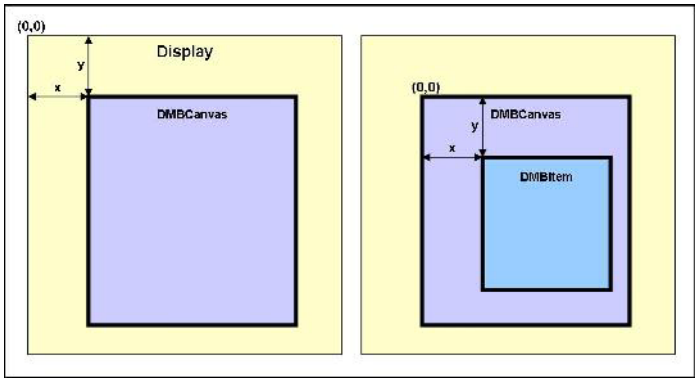
Represents the basic drawing surface which is laid over the video presentation. This is a subclass of `Canvas` and provides additional key mappings, which are unique to DMB environment. Besides the above mentioned facilities, `DMBCanvas` provides a simple framework for implementing custom UI components by allowing `DMBItem`s to be added to it.

Screen Management

When a `Canvas` in MIDP LCD UI is set to the `Display`, the current `Canvas` of the foreground application occupies the whole display area, and consumes all the input events generated by the user. Different from its superclass `Canvas`, `DMBCanvas` shares the display area with other `DMBCanvas`es set to `Display` by other applications. Therefore, the bounds of `DMBCanvas` must be specified. Graphics generated by a `DMBCanvas` will not be drawn outside its bounds, and pointer events are delivered to a `DMBCanvas` only when the pointer is inside the bounds of the `DMBCanvas`.:

As shown in the above picture, a display is shared among more than one `DMBCanvas`, and they are placed along the line of sight depending on the z-order value assigned for them. When a display is updated, a `DMBCanvas` with the smallest z-order value is drawn first, one with the next smallest value drawn next, and so on. So a `DMBCavnas` with the largest z-order value is drawn closest to the viewer (in the picture, the one with Z=2), and it is the foreground application.

Local Coordinate System



`DMBCanvas` and `DMBItem` have a local coordinate system of their own. In the left picture above, the display coordinate system is used when specifying the bounds of a `DMBCanvas` via its constructor or `setBounds(int,int,int,int)`. But the methods on `Graphics`, and other methods like `pointerPressed(int,int)`, `pointerDragged(int,int)`, and `pointerReleased(int,int)` use the local coordinate system of the corresponding `DMBCanvas`, where the origin in the coordinate system is the top-left corner of the `DMBCanvas`. As such, `DMBItem` uses the local coordinate system of its parent `DMBCanvas` when specifying its bounds within the parent, but when specifying coordinates for generating graphics and processing events, all the coordinates are in the local coordinate of the `DMBItem`.

DMB Action

To map actions unique in DMB environment, `DMBCanvas` provides the concept of DMB action. It is much like "game action" in `Canvas`, but defines a different set of actions which are unique to DMB. Instead of mapping arbitrary key codes to some actions, key codes should be checked if they correspond to any of DMB actions defined in this class, such as `VOLUME_UP`, `VOLUME_DOWN`, `MUTE`, `CHANNEL_UP`, `CHANNEL_DOWN`, `RECORD`, `GUIDE`, `INFO`, and so on. Each key code may be mapped to only one DMB action.

Key and Focus Events

Key events are delivered to a `DMBCavnas` that has the key focus. Among `DMBCanvas`es set to `Display` by focusable applications (refer to `DisplayControl.setFocusable(boolean)`), the one that is closest to the viewer receives the

key focus, and so does key events. If a MIDlet placed at the top of Display does not want to have the key focus, then it can call DisplayControl.setFocusable(boolean) with false.

In some cases, an application needs to exclusively receive events generated by acting on a set of keys, even when the application does not have a DMBCanvas. It may use KeyLock and reserve it with ResourceManager. Then the events for the designated set of keys will be delivered to the application owning the KeyLock whether it has UI and the key focus or not.

For some reasons, the key focus may be transferred among applications. In that case, the one losing the focus will be notified via a call to focusLost(), and another gaining the focus will be notified via a call to focusGained().

Pointer Events

When the pointer is pressed on the display, the corresponding event is delivered to a DMBCanvas which is closest to the viewer among those containing the point where the pointer is pressed. Drag and release events following a press event are delivered to the DMBCanvas to which the press event was delivered. If the pointer is pressed at a point not belonging to a DMBCanvas, then all the pointer events generated until another press event is generated will be delivered to no DMBCanvas, and silently ignored.

Paint Events

Different from Canvas to which paint events are delivered only when the Canvas is set by the foreground application, DMBCanvas may receive paint events. All the graphic operations performed in paint(Graphics) take effect on the display if they are not obscured by graphics generated by another DMBCanvas.

When more than one DMBCanvas display graphics on the display, it is drawn in the order imposed by their z-order values. But depending on implementations, if an area within a DMBCanvas is updated, other DMBCanvases covering a portion of the updated area need to recover the portion. In that case, the update performance may degrade significantly. This needs to be considered when planning applications coexisting at the same time sharing the display.

DMBItem

In a much similar way to Form, DMBCanvas may contain a set of DMBItems, provides methods for adding/removing and controlling DMBItems within it, and manages the contained DMBItems.

When a key event is generated, it is delivered to a DMBItem which has the key focus if the containing DMBCanvas has the key focus. Key events are first delivered to DMBCanvas, and then to a focused DMBItem. Note that the dispatch is done by the default implementation of the relevant event handling methods defined in DMBCanvas such as keyPressed(int).

In the case of pointer events, once they are to be delivered to a DMBCanvas according to the rule described above, they are delivered to a DMBItem within the DMBCanvas that contains the point asssociated with pointer events, and is closest to the viewer. Pointer events are also delivered to a DMBItem first through DMBCanvas. The dispatch mechanism is implemented within the relevant event handling methods defined in DMBCanvas. Once a pressed event is delivered to a DMBItem, then the corresponding dragged and released events are delivered to the same DMBItem whether the events are generated within the DMBItem or not. On the other hand, the pointer is pressed on no DMBItem, then any other pointer events generated till another pressed event is generated shall be ignored silently. Thus they are delivered to no DMBItem.

DMBItems are assigned an index. One with smaller index is drawn below one with larger index. The drawing is performed in paintItems(Graphics) method.

Field Summary		Page
static final int	**BACK** The action of going back in the user interface.	206

static final int	**CHANNEL_DOWN** The channel down action.	*205*
static final int	**CHANNEL_UP** The channel up action.	*205*
static final int	**EXIT** The action of exiting a context.	*206*
static final int	**GUIDE** The action of activating the guide program.	*206*
static final int	**INFO** The action of requesting more information.	*206*
static final int	**MUTE** The audio muting action.	*205*
static final int	**RECORD** The action of initiating recording of media.	*205*
static final int	**VOLUME_DOWN** The volume down action.	*205*
static final int	**VOLUME_UP** The volume up action.	*205*

Constructor Summary		***Page***
protected	**DMBCanvas**(int x, int y, int width, int height) Creates a new DMBCanvas to occupy the given area in the display.	*206*

Method Summary		***Page***
int	**appendItem**(DMBItem item) Adds the given DMBItem to this DMBCanvas.	*208*
protected void	**focusGained**() Called when this DMBCanvas gained the key focus.	*208*
protected void	**focusLost**() Called when this DMBCanvas lost the key focus.	*208*
int	**getDMBAction**(int keyCode) Returns the DMB action corresponding to the given key code in the device.	*207*
int	**getDMBKeyCode**(int dmbAction) Returns a key code corresponding to the given DMB action in the device.	*207*
DMBItem	**getFocusedItem**() Returns the DMBItem that has the key focus within this DMBCanvas.	*211*

DMBItem	**getItem**(int itemNum) Returns a DMBItem at the given index in the list of DMBItems contained in this DMBCanvas.	209
int	**getItemNum**() Returns the number of DMBItems contained in this DMBCanvas.	210
int	**getX**() Returns the x cooridnate of the top-left corner of this DMBCanvas in the display's coordinate system.	211
int	**getY**() Returns the y cooridnate of the top-left corner of this DMBCanvas in the display's coordinate system.	211
final boolean	**hasFocus**() Returns whether this DMBCanvas has the key focus.	208
void	**insertItem**(DMBItem item, int itemNum) Inserts the given DMBItem at the given index in the list of DMBItems in this DMBCanvas.	209
protected void	**keyPressed**(int keyCode) Called when this DMBCanvas has the key focus and a key is pressed.	212
protected void	**keyReleased**(int keyCode) Called when a key that was previously pressed is released.	212
protected void	**keyRepeated**(int keyCode) Called when a key is repeated (held down).	213
protected void	**paint**(Graphics g) Paints the content of this DMBCanvas.	211
void	**paintItems**(Graphics g) Paints all the DMBItems in this DMBCanvas from the one with the smallest index.	211
protected void	**pointerDragged**(int x, int y) Called when the pointer is dragged after the corresponding pressed event is delivered to this DMBCanvas.	214
protected void	**pointerPressed**(int x, int y) Called when the pointer is pressed on a point this DMBCanvas contains.	213
protected void	**pointerReleased**(int x, int y) Called when the pointer was pressed in this DMBCanvas and is then released.	213
void	**removeAllItems**() Removes all the DMBItems contained in this DMBCanvas.	210
void	**removeItem**(DMBItem item) Removes the given DMBItem from the list of DMBItems contained in this DMBCanvas.	210
void	**removeItem**(int itemNum) Removes a DMBItem at the given index in the list of DMBItems contained in this DMBCanvas.	210

void	**setBounds**(int x, int y, int width, int height) Sets the bounds of this DMBCanvas.	*207*

Field Detail

VOLUME_UP

```
public static final int VOLUME_UP = 1
```

The volume up action.

VOLUME_DOWN

```
public static final int VOLUME_DOWN = 2
```

The volume down action.

MUTE

```
public static final int MUTE = 3
```

The audio muting action.

CHANNEL_UP

```
public static final int CHANNEL_UP = 4
```

The channel up action.

CHANNEL_DOWN

```
public static final int CHANNEL_DOWN = 5
```

The channel down action.

RECORD

```
public static final int RECORD = 6
```

The action of initiating recording of media.

GUIDE

```
public static final int GUIDE = 7
```

The action of activating the guide program.

INFO

```
public static final int INFO = 8
```

The action of requesting more information.

BACK

```
public static final int BACK = 9
```

The action of going back in the user interface.

EXIT

```
public static final int EXIT = 10
```

The action of exiting a context.

Constructor Detail

DMBCanvas

```
protected DMBCanvas(int x,
                    int y,
                    int width,
                    int height)
```

Creates a new `DMBCanvas` to occupy the given area in the display. The coordinates are specified in the coordinate system of the display.

Method Detail

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Sets the bounds of this `DMBCanvas`. The coordinates are in the display coordinate system.

Parameters:

`x` - the x coordinate of the top-left corner of this `DMBCanvas`

`y` - the y coordinate of the top-left corner of this `DMBCanvas`

`width` - the width of this `DMBCanvas` in pixels

`height` - the height of this `DMBCanvas` in pixels

---

getDMBAction

```
public int getDMBAction(int keyCode)
```

Returns the DMB action corresponding to the given key code in the device.

Parameters:

`keyCode` - a key code

Returns:

the DMB action associated with the given key code. If there is no corresponding action, then returns 0

---

getDMBKeyCode

```
public int getDMBKeyCode(int dmbAction)
                  throws IllegalArgumentException
```

Returns a key code corresponding to the given DMB action in the device.

Parameters:

`dmbAction` - a DMB action

Returns:

a key code corresponding to the given DMB action

Throws:

`IllegalArgumentException` - thrown when the given action is invalid

---

focusGained

```
protected void focusGained()
```

Called when this DMBCanvas gained the key focus. More specifically, this is called in the following cases.

- When this DMBCanvas gained the key focus, and it did not have it before
- When this DMBCanvas set to Display, and the application has already fulfilled the condition to get the key focus. In this case, the key focus is immediately transferred to the newly set DMBCanvas.

The default implementation of this method dispatches this event by invoking UserItem.focusGained() to the UserItem having the key focus within the focused DMBCanvas.

Note that when a DMBItem that is not a UserItem has the key focus, the default implementation of this method may have the responsibility of notifying the DMBItem of its gaining the key focus. Thus the default implementation of this method must be called at an appropriate timing when overriding this method in any of subclasses.

---

focusLost

```
protected void focusLost()
```

Called when this DMBCanvas lost the key focus. But this method is not called if this DMBCanvas had once the key focus, and is removed from Display by calling DisplayControl.removeDisplayable().

The default implementation of this method dispatches this event to the UserItem owning the key focus within this DMBCanvas by calling UserItem.focusLost().

Note that even when the focused DMBItem is not a UserItem, the underlyiing implementation may rely on the default implementation of this method to get notified of the fact that the key focus has been moved to the DMBItem. Thus the default implementation of this method must be called in any of descendants of this class when overriding the default implementation.

---

hasFocus

```
public final boolean hasFocus()
```

Returns whether this DMBCanvas has the key focus. When a DMBCanvas has the key focus, it is eligible for receiving forthcoming key events.

Returns:

if owning the key focus, true. Otherwise false

---

appendItem

```
public int appendItem(DMBItem item)
```

Adds the given DMBItem to this DMBCanvas. It is placed at the end of the list of DMBItems managed by this DMBCanvas. It means that the DMBItem placed closest to the viewer.

Parameters:

item - DMBItem to add

Returns:

the index of the added DMBItem in the list DMBItem

Throws:

NullPointerException - thrown when the argument, item is null

IllegalStateException - thrown when the given item is already added to DMBCanvas

---

insertItem

```
public void insertItem(DMBItem item,
                       int itemNum)
```

Inserts the given DMBItem at the given index in the list of DMBItems in this DMBCanvas. itemNum must be in a range from 0 to getItemNum() including 0 and the getItemNum(), and if itemNum is equal to getItemNum(), then the DMBItem is added to the end of this list.

Parameters:

item - the DMBItem to insert

itemNum - the index at which the item will be inserted

Throws:

NullPointerException - thrown if item is null

IllegalStateException - thrown when the given item is already contained in this DMBCanvas

IndexOutOfBoundsException - thrown when the given itemNum is out of the valid range

---

getItem

```
public DMBItem getItem(int itemNum)
```

Returns a DMBItem at the given index in the list of DMBItems contained in this DMBCanvas. The given itemNum should be in the range from 0 to getItemNum()-1 inclusive.

Parameters:

itemNum - the index of the DMBItem to retrieve

Returns:

the designated DMBItem

Throws:

IndexOutOfBoundsException - thrown when the given itemNum is not in the valid range

removeItem

`public void **removeItem**(int itemNum)`

Removes a DMBItem at the given index in the list of DMBItems contained in this DMBCanvas. The given index, itemNum, should be in the range from 0 to getItemNum()-1 inclusive. If the DMBItem that is removed, had the key focus, then after removal, getFocusedItem() returns null. But in this case, even though the removed DMBItem is a UserItem, UserItem.focusLost() will not be called.

Parameters:

itemNum - the index of a DMBItem to remove

Throws:

IndexOutOfBoundsException - thrown when the given itemNum is outside the valid range

---

removeItem

`public void **removeItem**(DMBItem item)`

Removes the given DMBItem from the list of DMBItems contained in this DMBCanvas. If there is no such DMBItem in this DMBCanvas, it is silently ignored without throwing an exception. If the removed DMBItem had the key focus within this DMBCanvas, after removal, getFocusedItem() will return null.

Parameters:

item - the DMBItem to remove

Throws:

NullPointerException - thrown if the given item is null

---

removeAllItems

`public void **removeAllItems**()`

Removes all the DMBItems contained in this DMBCanvas.

---

getItemNum

`public int **getItemNum**()`

Returns the number of DMBItems contained in this DMBCanvas.

Returns:

the number of DMBItems in this DMBCanvas

paintItems

`public void **paintItems**(Graphics g)`

Paints all the DMBItems in this DMBCanvas from the one with the smallest index.

Parameters:

g - a Graphics object to be used to paint DMBItems

---

getFocusedItem

`public DMBItem **getFocusedItem**()`

Returns the DMBItem that has the key focus within this DMBCanvas. A newly created r394DMBCanvas does not have a DMBItem. If the first DMBItem is inserted, and DMBItem.setFocus() is not invoked on it, then getFocusedItem() shall return null meaning there is no DMBItem owning the key focus. An explicit call to DMBItem.setFocus() is required to set a DMBItem focused.

Returns:

a DMBItem that owns the key focus. If there is no DMBItem owning the key focus, then returns null

---

getX

`public int **getX**()`

Returns the x cooridnate of the top-left corner of this DMBCanvas in the display's coordinate system.

Returns:

the x coordinate of this DMBCanvas

---

getY

`public int **getY**()`

Returns the y cooridnate of the top-left corner of this DMBCanvas in the display's coordinate system.

Returns:

the y coordinate of this DMBCanvas

---

paint

`protected void **paint**(Graphics g)`

Paints the content of this DMBCanvas. The default implementation of this method calls paintItems(Graphics) to paint DMBItems contained in this DMBCanvas. If any graphics need to be painted in the background of the DMBItems, then this method should be overriden. The overriden method may paint something first in the background, and then invoke the default implementation of this method or paintItems(Graphics). Conversely if this method is overriden and the default implementation is not called, then no DMBItems will be painted automatically.

Overrides:

paint in class Canvas

Parameters:

g - the Graphics object to be used for painting the content of this DMBCanvas

---

keyPressed

```
protected void keyPressed(int keyCode)
```

Called when this DMBCanvas has the key focus and a key is pressed. The implementation of this method in r394DMBCanvas tries to dispatch the event to the DMBItem currently owning the key focus within this DMBCanvas. If the DMBItem owning the key focus is a UserItemUserItem.keyPressed(int) is called on the UserItem.

Note that a DMBItem other than UserItem may rely on the implementation of this method to obtain key events. Thus if this method is overriden, and all the DMBItems need to work properly, then the original implementation of this method must be called at an appropriate location within the overriden method.

Overrides:

keyPressed in class Canvas

Parameters:

keyCode - code for the key that is pressed

---

keyReleased

```
protected void keyReleased(int keyCode)
```

Called when a key that was previously pressed is released. The implementation of this method in DMBCanvas tries to dispatch the event to a DMBItem owning the key focus. If the focused DMBItem is a UserItem, then UserItem.keyReleased(int) is called.

Note that DMBItems other than a UserItem may rely on the implementation of this method to get key events. So if this method is overriden, and all the DMBItems within this DMBCanvas need to respond to key events properly, the original implementation must be called appropriately.

Overrides:

keyReleased in class Canvas

Parameters:

keyCode - code for the key that is released

keyRepeated

```
protected void keyRepeated(int keyCode)
```

Called when a key is repeated (held down). The implementation of this method in DMBCanvas tries to dispatch the event to a DMBItem in this DMBCanvas, which has the key focus. If the DMBItem is a UserItem, then UserItem.keyRepeated(int) is called on it.

Even in the case of other DMBItems than UserItems, they may rely on the implementation of this method to get key events of their interest. So if this method is overriden, and such DMBItems need to work properly, then this original implementation must be called at an appropriate time.

Overrides:

keyRepeated in class Canvas

Parameters:

keyCode - code for the key that is repeated

---

pointerPressed

```
protected void pointerPressed(int x,
                              int y)
```

Called when the pointer is pressed on a point this DMBCanvas contains. The implementation of this method in DMBCanvas tries to dispatch the event to a DMBItem which is closest to the viewer among those containing the point. If it is a UserItem, UserItem.pointerPressed(int,int) shall be called, where x and y coodinates are in the local coordinate system of the UserItem.

Note that DMBItems other than UserItems may rely on the implementation of this method to get pointer events of their interest. So if this method is overriden, and such DMBItems need to work properly, the original implementation of this method must be called at an appropriate time.

Overrides:

pointerPressed in class Canvas

Parameters:

x - the x coordinate of the point where the pointer is pressed in the local coordinate system of this DMBCanvas

y - the y coordinate of the point where the pointer is pressed in the local coordinate system of this DMBCanvas

---

pointerReleased

```
protected void pointerReleased(int x,
                               int y)
```

Called when the pointer was pressed in this DMBCanvas and is then released. The implementation of this method in DMBCanvas tries to dispatch the event to a DMBItem the corresponding pressed event was delivered

to. If it is a UserItem, UserItem.pointerReleased(int,int) shall be called on it. The x and y coordinates are in the local coordinate system of the UserItem.

Note that DMBItems other than UserItems may rely on the implementation of this method to get pointer events of their interest. If this method is overriden and DMBItems need to work properly, then the original implementation must be called at an appropriate time.

Overrides:

pointerReleased in class Canvas

Parameters:

x - the x coordinate of the point where the pointer is released

y - the y coordinate of the point where the pointer is released

---

pointerDragged

```
protected void pointerDragged(int x,
                              int y)
```

Called when the pointer is dragged after the corresponding pressed event is delivered to this DMBCanvas. The implementation of this method in this class tries to dispatch the event to an appropriate DMBItem within this DMBCanvas. The event is delivered to a DMBItem where the corresponding pressed event was delivered. If the DMBItem is a UserItem, then UserItem.pointerDragged(int,int) is invoked, where x and y coordinates are in the local cooridnate system of the UserItem.

Note that even if the DMBItem is not a UserItem, the underlying implementation of the DMBItem may rely on the implementation of this method to get the relevant events. Thus if this method is overriden in a subclass of this class, then the original implementation must be invoked, or DMBItems within instances of the subclass may not work properly.

Overrides:

pointerDragged in class Canvas

Parameters:

x - the x coordinate of the pointer when this event is generated in the local coordinate system of this DMBCanvas

y - the y coordinate of the pointer when this event is generated in the local coordinate system of this DMBCanvas

Class DMBItem

dmb.ui

```
java.lang.Object
  └ dmb.ui.DMBItem
```

Direct Known Subclasses:

TextItem, UserItem

abstract public class **DMBItem**

extends Object

The base class for UI components that can be added to a DMBCanvas. When specifying the bounds of this DMBItem, the local coordinate system of the parent DMBCanvas is used. But in the case of painting and pointer events, relevant coordinates are in the local coordinate system of this DMBItem.

Method Summary		Page
int	**getHeight**() Returns the height of this DMBItem in pixels.	216
int	**getWidth**() Returns the width of this DMBItem in pixels.	216
int	**getX**() Returns the x coordinate of the top-left corner of this DMBItem in the local coordinate system of the enclosing DMBCanvas.	216
int	**getY**() Returns the y coordinate of the top-left corner of this DMBItem in the local coordinate system of the enclosing DMBCanvas.	216
final boolean	**hasFocus**() Returns whether this DMBItem has the key focus.	216
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of this DMBItem in the local coordinate system of the enclosing DMBCanvas.	215
void	**setFocus**() Sets this DMBItem to be the key focus owner within its parent DMBCanvas.	217

Method Detail

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Sets the bounds of this DMBItem in the local coordinate system of the enclosing DMBCanvas.

Parameters:

x - the x cooridnate of this DMBItem

y - the y cooridnate of this DMBItem

width - the width of this DMBItem in pixels

height - the height of this DMBItem in pixels

getX

```
public int getX()
```

Returns the x coordinate of the top-left corner of this `DMBItem` in the local coordinate system of the enclosing `DMBCanvas`.

Returns:

the x coordinate

---

getY

```
public int getY()
```

Returns the y coordinate of the top-left corner of this `DMBItem` in the local coordinate system of the enclosing `DMBCanvas`.

Returns:

the y coordinate

---

getWidth

```
public int getWidth()
```

Returns the width of this `DMBItem` in pixels.

Returns:

the width

---

getHeight

```
public int getHeight()
```

Returns the height of this `DMBItem` in pixels.

Returns:

the height

---

hasFocus

```
public final boolean hasFocus()
```

Returns whether this DMBItem has the key focus. More specifically, for this method to return true, the parent DMBCanvas of this DMBItem must have the key focus, and this must be designated as a DMBItem owning the key focus within the DMBCanvas.

Returns:

true if this DMBItem has the key focus. Otherwise, false

---

setFocus

public void **setFocus**()

Sets this DMBItem to be the key focus owner within its parent DMBCanvas. If this DMBItem is not added to a DMBCanvas, this request is silently ignored.

If this method is invoked, the DMBItem owning the key focus is changed, but this does not mean that its parent DMBCanvas will be the key focus owner. Thus hasFocus() must return true, and actual key events will be delivered to the focused DMBItem.

Class FontLoader

dmb.ui

java.lang.Object

└ **dmb.ui.FontLoader**

---

public class **FontLoader**

extends Object

Loads new fonts from an InputStream or application resource. This class is for introducing new fonts that were not installed in the device. Font data may be transferred as a part of an application, or obtained in any other means. The created font will be valid only during the lifespan of the application loading the font. Thus, when a MIDlet is shutdown, all the fonts loaded by it shall be unloaded accordingly.

Method Summary		Page
static Font	**loadFont**(InputStream i) Creates a Font fron the given InputStream.	218
static Font	**loadFont**(String resourceName) Creates a Font from the specified application resource.	218

Method Detail

loadFont

```
public static Font loadFont(InputStream i)
```

Creates a `Font` fron the given `InputStream`.

Parameters:

`i` - an `InputStream` containing the raw font data

Returns:

the created `Font`

Throws:

`IllegalArgumentException` - thrown when the given `InputStream` does not convey a valid font data

---

loadFont

```
public static Font loadFont(String resourceName)
```

Creates a `Font` from the specified application resource.

Parameters:

`resourceName` - name of the resource containing the font data

Returns:

the created `Font`

Throws:

`IllegalArgumentException` - thrown when a `Font` can not be created from the specified resource

Class KeyLock

dmb.ui

```
java.lang.Object
  └ dmb.ui.KeyLock
```

All Implemented Interfaces:

Resource

public class **KeyLock**

extends Object

implements Resource

Represents the right to exclusively receive events generated by the associated set of keys. To exclusively receive events for a set of keys no matter whether an application has UI or not, the application should acquire an instance of this class that is created in association with the keys of interest. Then the key events will exclusively be delivered to the owning application.

Constructor Summary	Page
KeyLock(int[] keyCodes) Creates a KeyLock instance to get the exclusive right to receive key events generated by the specified keys.	219

	Method Summary	Page
int[]	**getKeyCodes**() Returns the key codes associated with this KeyLock object.	219
protected boolean	**keyPressed**(int keyCode) Called when one of the associated keys is pressed.	220
protected boolean	**keyReleased**(int keyCode) Called when one of the associated keys is released.	220
protected boolean	**keyRepeated**(int keyCode) Called when one of the associated keys is repeated.	220

Constructor Detail

KeyLock

public **KeyLock**(int[] keyCodes)

Creates a KeyLock instance to get the exclusive right to receive key events generated by the specified keys.

Method Detail

getKeyCodes

public int[] **getKeyCodes**()

Returns the key codes associated with this KeyLock object. The array returned is a copy of the array given to the constructor of KeyLock.

Returns:

the list of key codes associated with this object

---

keyPressed

```
protected boolean keyPressed(int keyCode)
```

Called when one of the associated keys is pressed. Depending on the return value of this method, the event is further dispatched to the `Displayable`. If `true` is returned, it means that the key event is fully processed within this method, so the event is not propagated further. But if `false` is returned, it signifies that the event should be processed further by a UI component that has the key focus within the context of the application receiving the key event. The default implementation of this method returns `false`. Thus in case an application wants to process key events it exclusively reserved, `KeyLock` need not be subclassed.

Parameters:

`keyCode` - the code of the pressed key

Returns:

`true`, if the key event should not be dispatched to a UI component. `false` otherwise

---

keyReleased

```
protected boolean keyReleased(int keyCode)
```

Called when one of the associated keys is released. Depending on the return value of this method, the event is further dispatched to the `Displayable`. If `true` is returned, it means that the key event is fully processed within this method, so the event is not propagated further. But if `false` is returned, it signifies that the event should be processed further by a UI component that has the key focus within the context of the application receiving the key event. The default implementation of this method returns `false`. Thus in case an application wants to process key events it exclusively reserved, `KeyLock` need not be subclassed.

Parameters:

`keyCode` - the code of the released key

Returns:

`true`, if the key event should not be dispatched to a UI component. `false` otherwise

---

keyRepeated

```
protected boolean keyRepeated(int keyCode)
```

Called when one of the associated keys is repeated. Depending on the return value of this method, the event is further dispatched to the `Displayable`. If `true` is returned, it means that the key event is fully processed within this method, so the event is not propagated further. But if `false` is returned, it signifies that the event should be processed further by a UI component that has the key focus within the context of the application receiving the key event. The default implementation of this method returns `false`. Thus in case an application wants to process key events it exclusively reserved, `KeyLock` need not be subclassed.

Parameters:

`keyCode` - the code of the repeated key

Returns:

`true`, if the key event should not be dispatched to a UI component. `false` otherwise

Class TextItem

dmb.ui

`java.lang.Object`

`dmb.ui.DMBItem`

`dmb.ui.TextItem`

---

public class **TextItem**

extends DMBItem

A `DMBItem` subclass providing text editing capability. This class has almost the same set of methods as `TextField`, and input constraints and mode setting are identical to it. Instead, this `DMBItem` does not have any decoration around it, thus may be used in the context of custom user interfaces without any inconsistency with the look of the surrounding user interfaces.

Constructor Summary	*Page*
`TextItem``(String text, int maxSize, int constraints)` Creates a new `TextItem` with the given initial text, the maximum number of characters, and a constraints.	*222*

Method Summary		*Page*
`void`	**`delete`**`(int offset, int length)` Deletes characters from this `TextItem`.	*227*
`int`	**`getCaretPosition`**`()` Returns the current position of the caret where the next character will be inserted.	*223*
`void`	**`getChars`**`(char[] data)` Copies the current contents of this `TextItem` into the given character array starting at index zero.	*226*
`int`	**`getConstraints`**`()` Returns the current input constraints of this `TextItem`.	*224*

int	**getMaxSize**() Returns the maximum size (number of characters) that can be stored in this TextItem.	*224*
String	**getString**() Returns the contents of this TextItem as a string value.	*225*
void	**insert**(char[] data, int offset, int length, int position) Inserts a subrange of the given array of characters into the contents of this TextItem.	*226*
void	**insert**(String str, int position) Inserts a string into the contents of this TextItem.	*227*
void	**setChars**(char[] data, int offset, int length) Sets the contents of this TextItem from a character array, replacing the previous contents.	*225*
void	**setConstraints**(int constraints) Sets the input constraints of this TextItem.	*224*
void	**setInitialInputMode**(String characterSubset) Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this TextItem.	*223*
void	**setMaxSize**(int maxSize) Sets the maximum size (number of characters) that can be contained in this Textitem.	*223*
void	**setString**(String text) Sets the contents of this TextItem as a string value, replacing the previous contents.	*224*
int	**size**() Returns the number of characters that are current stored in this TextItem.	*223*

Methods inherited from class dmb.ui.DMBItem
getHeight, getWidth, getX, getY, hasFocus, setBounds, setFocus

Constructor Detail

TextItem

```
public TextItem(String text,
                int maxSize,
                int constraints)
```

Creates a new TextItem with the given initial text, the maximum number of characters, and a constraints.

Throws:

IllegalArgumentException

Method Detail

getCaretPosition

```
public int getCaretPosition()
```

Returns the current position of the caret where the next character will be inserted.

Returns:

The current input position, where 0 means the beginning of the text

---

setInitialInputMode

```
public void setInitialInputMode(String characterSubset)
```

Sets a hint to the implementation as to the input mode that should be used when the user initiates editing of this `TextItem`. The `characterSubset` parameter names a subset of Unicode characters that is used by the implementation to choose an initial input mode. If `null` is passed, the implementation should choose a default input modes. The concept of input modes is same as that of `TextField`. Please refer to the description on the class for further details on input modes.

Parameters:

`characterSubset` - a string naming a Unicode character subset, or `null`

---

size

```
public int size()
```

Returns the number of characters that are current stored in this `TextItem`.

Returns:

number of characters in this `TextItem`

---

setMaxSize

```
public void setMaxSize(int maxSize)
```

Sets the maximum size (number of characters) that can be contained in this `Textitem`. If the current contents of the `TextItem` are larger than `maxSize`, the contents are truncated to fit.

Parameters:

`maxSize` - the new maximum size

Throws:

`IllegalArgumentException` - thrown when any of the followings is true.

- when `maxSize` is non-positive
- when the contents after truncation would be illegal for the current input constraints

---

getMaxSize

```
public int getMaxSize()
```

Returns the maximum size (number of characters) that can be stored in this `TextItem`.

Returns:

the maximum size in characters

---

setConstraints

```
public void setConstraints(int constraints)
```

Sets the input constraints of this `TextItem`. If the curent contents of this `TextItem` do not match the new `constraints`, the contents are set to empty.

Parameters:

`constraints` - the input constraints. See `TextField` for the details of input constraints

Throws:

`IllegalArgumentException` - thrown when the constraints is not one of those specified in `TextField`

---

getConstraints

```
public int getConstraints()
```

Returns the current input constraints of this `TextItem`.

Returns:

the current input constraints value

See Also:

`setConstraints(int)`

---

setString

```
public void setString(String text)
```

Sets the contents of this `TextItem` as a string value, replacing the previous contents.

Parameters:

`text` - the new value of this `TextItem`, or `null` if this `TextItem` is to be made empty

Throws:

`IllegalArgumentException` - thrown in the following cases.

- when the given `text` is illegal for the current input constraints(see `setConstraints(int)`
- when the given `text` would exceed the current maximum capacity

---

setChars

```
public void setChars(char[] data,
                     int offset,
                     int length)
```

Sets the contents of this `TextItem` from a character array, replacing the previous contents. Characters are copied from the region of the given `data` array starting at the given index, `offset`, and running for the given `length` characters. If the `data` is `null`, this `TextItem` is set to be empty and the other parameters are ignored.

The `offset` and the `length` parameters must specify a valid range of characters within the character array, `data`. The `offset` parameter must be within the range [0..(data.length)], inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the source of the character data

`offset` - the beginning index of the region to copy in `data`

`length` - the number of characters to copy

Throws:

`ArrayIndexOutOfBoundsException` - thrown when `offset` and `length` do not specify a valid range within the `data` array

`IllegalArgumentException` - thrown when either the given `data` array is illegal for the current input constraints, or the specified text would exceed the current maximum capacity

---

getString

```
public String getString()
```

Returns the contents of this `TextItem` as a string value.

Returns:

the current contents as a string

---

getChars

```
public void getChars(char[] data)
```

Copies the current contents of this `TextItem` into the given character array starting at index zero. Array elements beyond the characters copied are left unchanged.

Parameters:

`data` - the character array into which the contents will be copied

Returns:

the number of characters to be copied

Throws:

`ArrayIndexOutOfBoundsException` - thrown when the length of the given array is smaller than the number of characters to copy

`NullPointerException` - thrown when the given `data` is `null`

---

insert

```
public void insert(char[] data,
                   int offset,
                   int length,
                   int position)
```

Inserts a subrange of the given array of characters into the contents of this `TextItem`. The given `offset` and the `length` parameters indicate the subrange of the data array to be used for insertion. Behaviour is otherwise identical to that of `insert(String, int)`.

The `offset` and the `length` parameters must specify a valid range of characters within the given character array, `data`. The `offset` parameter must be within the range [0..(data.length)], inclusive. The `length` parameter must be a non-negative integer such that `(offset + length) <= data.length`.

Parameters:

`data` - the source of the character data

`offset` - the beginning index of start of the region to copy

`length` - the number of characters to copy

`position` - the position at which the given data will be inserted

Throws:

`ArrayIndexOutOfBoundsException` - thrown when the given `offset` and the `length` do not specify a valid range within the given `data` array

`IllegalArgumentException` - thrown when the resulting contents would be illegal for the current input constraints, or exceed the current maximum capacity

`NullPointerException` - thrown when the given `data` is `null`

---

insert

```
public void insert(String str,
                   int position)
```

Inserts a string into the contents of this `TextItem`. The string is inserted just before the character indicated by the `position` parameter, where zero specifies the first character in the contents of this `TextItem`. If `position` is less than or equal to zero, the insertion occurs at the beginning of the contents, thus effecting a prepend operation. If `position` is greater than or equal to the current size of the contents, the insertion occurs immediately after the end of the contents, thus effecting an append operation. For example, `text.insert(s, text.size())` always appends the string `s` to the current contents.

The current size of the contents is increased by the number of inserted characters. The resulting string must fit within the current maximum capacity.

If the application needs to simulate typing of characters, it can determine the location of the current insertion point("caret") using `getCaretPosition()`. For example, `text.insert(s, text.getCaretPosition())` inserts the string `s` at the current caret position.

Parameters:

`str` - the string to be inserted

`position` - the position at which the `str` will be inserted

Throws:

`IllegalArgumentException` - thrown when the resulting contents would be illegal for the current input constraints, or exceed the current maximum capacity

`NullPointerException` - thrown when the given `str` is `null`

---

delete

```
public void delete(int offset,
                   int length)
```

Deletes characters from this `TextItem`.

The given `offset` and `length` parameters must specify a valid range of characters within the contents of this `TextItem`. The `offset` parameter must be in the range of [0..(size())], inclusive. And the `length` parameter must be a non-negative integer such that `(offset + length) <= size`.

Parameters:

`offset` - the beginningof the region to delete

`length` - the number of characters to delete

Throws:

`IllegalArgumentException` - thrown when the resulting contents would be illegal for the current input constraints

`StringIndexOutOfBoundsException` - thrown when the given `offset` and the `length` do not specify a valid range within the contents of this `TextItem`

Class UserItem

dmb.ui

java.lang.Object

  dmb.ui.DMBItem

    dmb.ui.UserItem

---

abstract public class **UserItem**

extends DMBItem

A DMBItem subclass that may be subclassed and customized. In this class, methods for setting and getting various properties affecting the behaviour and the look are provided. In addition to them, this class defines a set of methods to get notified of UI events delivered to UserItem.

Constructor Summary		Page
protected	**UserItem**() Creates a new UserItem.	229

Method Summary		Page
protected void	**added**(DMBCanvas canvas) Called when this UserItem is added to a DMBCanvas.	229
protected void	**focusGained**() Called when this UserItem gained the key focus.	231
protected void	**focusLost**() Called when the key focus is lost.	232
protected void	**keyPressed**(int keyCode) Called when a key is pressed while this UserItem owns the key focus (that is, hasFocus() returns true).	230
protected void	**keyReleased**(int keyCode) Called when a key is released while this UserItem owns the key focus (that is, hasFocus() returns true).	230
protected void	**keyRepeated**(int keyCode) Called when a key is repeated while this UserItem owns the key focus (that is, hasFocus() returns true).	230
protected abstract void	**paint**(Graphics g) Paints the contents of this UserItem using the given Graphics object.	229

protected void	**pointerDragged**(int x, int y) Called when the pointer is first pressed within this UserItem and dragged.	*231*
protected void	**pointerPressed**(int x, int y) Called when the pointer is pressed within this UserItem.	*231*
protected void	**pointerReleased**(int x, int y) Called when the pointer is pressed within this UserItem and released.	*231*
protected void	**removed**(DMBCanvas canvas) Called when this UserItem is removed from the parent DMBCanvas.	*230*

Methods inherited from class dmb.ui.DMBItem
getHeight, getWidth, getX, getY, hasFocus, setBounds, setFocus

Constructor Detail

UserItem

protected **UserItem**()

Creates a new UserItem.

Method Detail

paint

protected abstract void **paint**(Graphics g)

Paints the contents of this UserItem using the given Graphics object. The Graphics object uses the local coordinate system of this UserItem.

Parameters:

g - a Graphics to paint the contents with

---

added

protected void **added**(DMBCanvas canvas)

Called when this UserItem is added to a DMBCanvas.

As with methods described in *Event Handling* section of the package description of javax.microedition.lcdui, this method is not called at the same time as those methods.

Parameters:

canvas - DMBCanvas where this UserItem is added

removed

```
protected void removed(DMBCanvas canvas)
```

Called when this `UserItem` is removed from the parent `DMBCanvas`.

As with methods described in *Event Handling* section of the package description of `javax.microedition.lcdui`, this method is not called at the same time as those methods.

Parameters:

`canvas` - `DMBCanvas` from which this `UserItem` is removed

---

keyPressed

```
protected void keyPressed(int keyCode)
```

Called when a key is pressed while this `UserItem` owns the key focus (that is, `hasFocus()` returns `true`).

Parameters:

`keyCode` - the key code

---

keyReleased

```
protected void keyReleased(int keyCode)
```

Called when a key is released while this `UserItem` owns the key focus (that is, `hasFocus()` returns `true`).

Parameters:

`keyCode` - the key code

---

keyRepeated

```
protected void keyRepeated(int keyCode)
```

Called when a key is repeated while this `UserItem` owns the key focus (that is, `hasFocus()` returns `true`).

Parameters:

`keyCode` - the key code

---

pointerDragged

```
protected void pointerDragged(int x,
                              int y)
```

Called when the pointer is first pressed within this `UserItem` and dragged.

Parameters:

`x` - the x coordinate of the pointer when it is dragged

`y` - the y coordinate of the pointer when it is dragged

---

pointerPressed

```
protected void pointerPressed(int x,
                              int y)
```

Called when the pointer is pressed within this `UserItem`.

Parameters:

`x` - the x coordinate of the pointer when it is pressed

`y` - the y coordinate of the pointer when it is pressed

---

pointerReleased

```
protected void pointerReleased(int x,
                               int y)
```

Called when the pointer is pressed within this `UserItem` and released.

Parameters:

`x` - the x coordinate of the pointer when it is released

`y` - the y coordinate of the pointer when it is released

---

focusGained

```
protected void focusGained()
```

Called when this `UserItem` gained the key focus. To gain the key focus, this `UserItem` must be added to a `DMBCanvas`, set to get the key focus within the `DMBCanvas`, and the `DMBCanvas` must gain the key focus.

As with methods described in *Event Handling* section of the package description of `javax.microedition.lcdui`, this method is not called at the same time as those methods.

---

focusLost

```
protected void focusLost()
```

Called when the key focus is lost. More specifically, another DMBItem may gain the key focus, or DMBCanvas may lost the key focus.

As with methods described in *Event Handling* section of the package description of `javax.microedition.lcdui`, this method is not called at the same time as those methods.

Package dmb.util

Defines common interfaces and classes used in other packages.

See:

Description

Interface Summary		Page
AsyncRequestor	An interface to be implemented by an object that needs to get notified of progresses of an asynchronous operation.	*233*
AsyncResult	Represents an asynchronous operation that will be completed in the future.	*234*
AttributedObject	Represents an object having an associated set of attributes.	*238*

Exception Summary		Page
InvalidAttributeException	An exception thrown when a non-existing attribute is specified for an `AttributedObject`.	*247*

Package dmb.util Description

Defines common interfaces and classes used in other packages.

Interface AsyncRequestor

dmb.util

---

public interface **AsyncRequestor**

An interface to be implemented by an object that needs to get notified of progresses of an asynchronous operation. A method supporting asynchronous operation usually takes an instance of this interface as a parameter.

See Also:

`AsyncResult`

Method Summary		Page
void	**resultUpdated**(`AsyncResult` result) Called when an associated asynchronous operation has progressed.	*234*

Method Detail

resultUpdated

`public void **resultUpdated**(AsyncResult result)`

Called when an associated asynchronous operation has progressed. If the operation reports progresses besides its completion, then `AsyncResult.getProgress()` may be consulted. In that case, for an `AsyncResult`, this method may be called more than once. So `AsyncResult.isDone()` must be consulted to know if the operation has been completed.

Parameters:

`result` - the `AsyncResult` associated with the operation that has been progressed

Interface AsyncResult

dmb.util

---

public interface **AsyncResult**

Represents an asynchronous operation that will be completed in the future. APIs supporting asynchronous operation return an instance of this interface. With this interface, the operation may be tracked and controlled.

`AsyncResult` and `AsyncRequestor` are elements of an API pattern supporting monitoring and control of asynchronous operations. A method initiating an asynchronous operation should return an instance of `AsyncResult`. And if it is to provide notification of progresses of the operation, `AsyncRequestor` should also be taken as a parameter to the method. The following code snippets are examples of using APIs supporting the pattern consisting of `AsyncResult` and `AsyncRequestor`.

```
public class Loader {
    ...
    AsyncResult loadAsynchronously(String url, AsyncRequestor requestor);
    ...
}
...

    Loader l = new Loader();

    // Getting notified when the operation is completed
    l.loadAsynchronously("http://image.location/icon.png",
        new AsyncRequestor() {
            public void resultUpdated(AsyncResult result) {
                if (!result.isDone()) {
                    System.out.println("Progress (%): "
                        + result.getProgress());
                    return;
                }
                try {
                    ui.setIcon((Image) result.get());
                    ui.repaint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    };
 ...

    // After performing some other tasks, and then waiting for the result
    // of an asynchronous operation
    AsyncResult r = l.loadAsynchronously("http://image.location/icon.png",
```

```
                                        null);
    ... Do something else ...

    try {
        ui.setIcon((Image) r.get());
        ui.repaint();
    } catch (Exception e) {
        e.printStackTrace();
    }

...
    // Polling the result in-between performing other tasks
    AsyncResult r = l.loadAsynchronously("http://image.location/icon.png",
                                        null);
    try {
        while (!r.isDone()) {
            Thread.sleep(1000);
        }
        ui.setIcon((Image) r.get());
        ui.repaint();
    } catch (Exception e) {
        e.printStackTrace();
    }

...
```

Field Summary		Page
int	**PROGRESS_UNKNOWN** Returned from getProgress() when there is no information on the current progress, or the operation is canceled or failed.	236

Method Summary		Page
boolean	**cancel**() Cancels the operation.	238
void	**complete**() Waits for the completion of the operation.	237
void	**complete**(int timeout) Waits for the operation completes or the given timeout is expired.	237
Object	**get**() Returns the result of the operation.	236
Object	**get**(int timeout) Returns the result of this operation if it is completed before the given timeout.	237
int	**getProgress**() Returns the current progress of the operation in percents. 0 means there is no progress at all, and 100 means it is completed.	238
boolean	**isCanceled**() Reports whether the operation has been canceled or not.	236
boolean	**isDone**() Reports whether the operation is done.	236

Field Detail

PROGRESS_UNKNOWN

```
public static final int PROGRESS_UNKNOWN = -1
```

Returned from `getProgress()` when there is no information on the current progress, or the operation is canceled or failed.

Method Detail

isDone

```
public boolean isDone()
```

Reports whether the operation is done.

Returns:

if the operation is done, returns `true`. Otherwise, `false`. Note that this method returns `true`, whether the operation has been completed normally, canceled by `cancel()`, or failed for some reason

---

isCanceled

```
public boolean isCanceled()
```

Reports whether the operation has been canceled or not.

Returns:

if canceled, `true`. Otherwise, `false`

---

get

```
public Object get()
           throws InterruptedException,
                  Exception
```

Returns the result of the operation. If it is not complete yet, this method blocks until it is completed.

Throws:

`InterruptedException` - thrown when the calling thread is interrupted by `Thread.interrupt()`, or the operation is canceled

`Exception` - thrown when the operation caused any exception other than `InterruptedException` to be thrown

---

complete

```
public void complete()
              throws InterruptedException,
                     Exception
```

Waits for the completion of the operation. Except that this method does not return a result, this is identical to `get()`.

Throws:

`InterruptedException` - thrown when the calling thread is interrupted by `Thread.interrupt()`, or the operation is canceled

`Exception` - thrown when the operation caused any exception other than `InterruptedException` to be thrown

---

get

```
public Object get(int timeout)
           throws InterruptedException,
                  Exception
```

Returns the result of this operation if it is completed before the given timeout. This method blocks until the operation completes, or the given timeout is expired. When this method returns because of the timeout, returns `null`. But `null` may be returned when the result value itself is `null`. Therefore `isDone()` must be consulted to check whether the operation actually completed.

Parameters:

`timeout` - the timeout in milliseconds. `0` means there is no timeout

Returns:

the result of the operation. If this method returns because of the timeout, returns `null`

Throws:

`InterruptedException` - thrown when the calling thread is interrupted by `Thread.interrupt()`, or the operation is canceled

`Exception` - thrown when the operation caused any exception other than `InterruptedException` to be thrown

---

complete

```
public void complete(int timeout)
              throws InterruptedException,
                     Exception
```

Waits for the operation completes or the given timeout is expired. Except that this method does not return the result, this is identical to `get(int)`.

Parameters:

`timeout` - the timeout in milliseconds. `0` means there is no timeout

Throws:

`InterruptedException` - thrown when the calling thread is interrupted by `Thread.interrupt()`, or the operation is canceled

`Exception` - thrown when the operation caused any exception other than `InterruptedException` to be thrown

---

cancel

`public boolean` **`cancel`**`()`

Cancels the opeation.

Returns:

if the operation is successfully canceled, returns `true`. Otherwise `false`. Usually an asynchronous operation can not be canceled because it is already completed, or API implementation does not support cancelation

---

getProgress

`public int` **`getProgress`**`()`

Returns the current progress of the operation in percents. 0 means there is no progress at all, and 100 means it is completed. But 100 does not ensure that `isDone()` returns `true`. So to check if the operation is completed, `isDone()` must be consulted instead of this method.

Returns:

an integer representing the current progress. It is from 0 to 100, inclusive. For some reasons, the current progress is unknown, returns `PROGRESS_UNKNOWN`

Interface AttributedObject

dmb.util

All Known Subinterfaces:
BroadcastFileConnection, ServiceComponent, SIObject

All Known Implementing Classes:
SIDatabase

---

public interface **AttributedObject**

Represents an object having an associated set of attributes.

Method Summary		Page
String[]	**getAttributes**() Returns a list of the attributes associated with this object.	239
boolean	**getBoolean**(String a) Returns a boolean value that is the value of the given attribute.	241
boolean[]	**getBooleanList**(String a) Returns an array of booleans that is the value of the given attribute.	242
byte[]	**getBytes**(String a) Returns a byte array that is the value of the given attribute.	246
Date	**getDate**(String a) Returns a Date value that is the value of the given attribute.	245
Date[]	**getDateList**(String a) Returns an array of Dates that is the value of the given attribute.	246
int	**getInt**(String a) Returns an int value that is the value of the given attribute.	242
int[]	**getIntList**(String a) Returns an array of ints that is the value of the given attribute.	243
long	**getLong**(String a) Returns a long value that is the value of the given attribute.	243
long[]	**getLongList**(String a) Returns an array of longs that is the value of the given attribute.	244
Object	**getObject**(String a) Returns an Object that is the value of the given attribute.	240
Object[]	**getObjectList**(String a) Returns the value of the given attribute that is an array of objects.	241
String	**getString**(String a) Returns a String value that is the value of the given attribute.	244
String[]	**getStringList**(String a) Returns an array of Strings that is the value of the given attribute.	245
boolean	**isValid**(String a) Checks if there is an attribute of the given name in this object.	240

Method Detail

getAttributes

```
public String[] getAttributes()
```

Returns a list of the attributes associated with this object.

Returns:

an array containing names of all the attributes associated with this object

---

isValid

```
public boolean isValid(String a)
                throws NullPointerException
```

Checks if there is an attribute of the given name in this object.

Parameters:

`a` - name of the attribute to check

Returns:

`true` if there is an attribute of the given name. Otherwise, `false`

Throws:

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getObject

```
public Object getObject(String a)
                 throws InvalidAttributeException,
                        NullPointerException,
                        IllegalStateException
```

Returns an `Object` that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the object value

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getObjectList

```
public Object[] getObjectList(String a)
                        throws InvalidAttributeException,
                               NullPointerException,
                               IllegalStateException
```

Returns the value of the given attribute that is an array of objects.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute or its type is not an array of object

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getBoolean

```
public boolean getBoolean(String a)
                    throws InvalidAttributeException,
                           NullPointerException,
                           IllegalStateException
```

Returns a `boolean` value that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not `boolean`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getBooleanList

```
public boolean[] getBooleanList(String a)
                         throws InvalidAttributeException,
                                NullPointerException,
                                IllegalStateException
```

Returns an array of `boolean`s that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `boolean`s

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getInt

```
public int getInt(String a)
           throws InvalidAttributeException,
                  NullPointerException,
                  IllegalStateException
```

Returns an `int` value that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not `int`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getIntList

```
public int[] getIntList(String a)
                 throws InvalidAttributeException,
                        NullPointerException,
                        IllegalStateException
```

Returns an array of `int`s that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `int`s

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getLong

```
public long getLong(String a)
             throws InvalidAttributeException,
                    NullPointerException,
                    IllegalStateException
```

Returns a `long` value that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not `long`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getLongList

```
public long[] getLongList(String a)
                   throws InvalidAttributeException,
                          NullPointerException,
                          IllegalStateException
```

Returns an array of `long`s that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `long`s

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getString

```
public String getString(String a)
                  throws InvalidAttributeException,
                         NullPointerException,
                         IllegalStateException
```

Returns a `String` value that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not `String`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getStringList

```
public String[] getStringList(String a)
                        throws InvalidAttributeException,
                               NullPointerException,
                               IllegalStateException
```

Returns an array of `String`s that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `String`s

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getDate

```
public Date getDate(String a)
             throws InvalidAttributeException,
                    NullPointerException,
                    IllegalStateException
```

Returns a `Date` value that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not `Date`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getDateList

```
public Date[] getDateList(String a)
                   throws InvalidAttributeException,
                          NullPointerException,
                          IllegalStateException
```

Returns an array of `Date`s that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `Date`s

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

---

getBytes

```
public byte[] getBytes(String a)
                 throws InvalidAttributeException,
                        NullPointerException,
                        IllegalStateException
```

Returns a byte array that is the value of the given attribute.

Parameters:

`a` - name of the attribute

Returns:

the value of the attribute

Throws:

`InvalidAttributeException` - thrown when there is no such attribute in this object, or its type is not an array of `bytes`

`NullPointerException` - thrown when the given `a` parameter is `null`

`IllegalStateException` - thrown when it is impossible to read value of the given attribute because of version change or removal of the underlying data

Class InvalidAttributeException

dmb.util

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └dmb.util.InvalidAttributeException
```

All Implemented Interfaces:
Serializable

---

public class **InvalidAttributeException**

extends RuntimeException

An exception thrown when a non-existing attribute is specified for an `AttributedObject`.

Constructor Summary	*Page*
InvalidAttributeException() Creates an instance of this exception without specifying the reason.	*247*
InvalidAttributeException(String reason) Creates an instance of this exception with a string describing the reason of the exception.	*247*

Constructor Detail

InvalidAttributeException

```
public InvalidAttributeException()
```

Creates an instance of this exception without specifying the reason.

InvalidAttributeException

```
public InvalidAttributeException(String reason)
```

Creates an instance of this exception with a string describing the reason of the exception.

History

Document history		
V1.1.1	August 2009	Publication