

10<sup>th</sup> Meeting

Turin

99/10/13

**OPIMA**

**Open Platform Initiative for Multimedia Access**

Specification

Version 1.0

# **OPIMA Specification**

**Version 1.0**

## **Disclaimer**

OPIMA reserves the right to modify the contents of this document according to its procedures for technical work and without notice. OPIMA makes no claim as to operational integrity of any implementation derived from this document. This document is provided *as is* with no warranties whatsoever.

This OPIMA specification has been developed and published in the framework of the International Electrotechnical Commission's (IEC) Industry Technical Agreement (ITA) programme.

# Table of Contents

<b>1</b>	<b>Introduction (Informative)</b>	<b>5</b>
<b>2</b>	<b>The OPIMA Architecture (Informative)</b>	<b>6</b>
2.1	<i>Credential mechanisms in the OPIMA environment</i>	7
2.2	<i>Required Trusted Institutions</i>	7
2.2.1	Compartment ID issuance authority	8
2.2.2	Credential issuance authorities	8
2.2.3	IPMP Systems ID issuance authorities	8
2.2.4	OPIMA Peer ID issuance authorities	8
2.2.5	Encryption, Signature and Watermarking ID issuance authorities	8
2.3	<i>Back-end Infrastructure</i>	8
2.4	<i>Protocols</i>	8
<b>3</b>	<b>The OPIMA Architecture (Normative)</b>	<b>10</b>
3.1	<i>OPIMA Protocols</i>	10
3.1.1	First layer: Secure Authenticated Channel	10
3.1.2	Second layer: OPIMA Common Message Protocol	10
3.1.2.1	Open IPMP System download Message	10
3.1.2.2	IPMP System code Messages	10
3.1.2.3	Close of the OPIMA download protocol	11
3.1.2.4	Message IDs	11
3.2	<i>Credential Formats</i>	11
3.3	<i>OPIMA Peer</i>	11
3.3.1	OPIMA Virtual Machine	12
3.3.2	IPMPs	12
3.3.3	Application Services API	12
3.3.3.1	useContent	12
3.3.3.2	getIpmSystem	14
3.3.3.3	queryOVM	14
3.3.3.4	sendMessageToIPMP	15
3.3.3.5	notifyEvent	15
3.3.4	IPMP Services API	16
3.3.4.1	User Interface methods	16
3.3.4.1.1	sendMessageToUser	16
3.3.4.1.2	receiveMessageFromUser	17
3.3.4.2	Secure Storage Interface	17
3.3.4.2.1	secureStoreData	17
3.3.4.2.2	secureRetrieveData	17
3.3.4.2.3	secureDeleteData	18
3.3.4.3	Encryption and Decryption Engines	18
3.3.4.3.1	queryEncryptionAlgorithms	18
3.3.4.3.2	encrypt	19
3.3.4.3.3	initEncryption	19
3.3.4.3.4	updateEncryptionKeys	19
3.3.4.3.5	stopEncryption	20
3.3.4.3.6	decrypt	20
3.3.4.3.7	initDecryption	20
3.3.4.3.8	updateDecryptionKeys	21
3.3.4.3.9	stopDecryption	21

3.3.4.4	Signature Engines	21
3.3.4.4.1	querySignatureAlgorithms	21
3.3.4.4.2	verifySignature	21
3.3.4.4.3	verifyContentSignature	22
3.3.4.4.4	generateSignature	22
3.3.4.4.5	generateContentSignature	23
3.3.4.5	Watermarking Engines	23
3.3.4.5.1	queryWatermarkAlgorithms	24
3.3.4.5.2	extractWatermark	24
3.3.4.5.3	stopWatermarkExtraction	24
3.3.4.5.4	newWatermark	24
3.3.4.5.5	insertWatermark	25
3.3.4.5.6	stopWatermarkInsertion	25
3.3.4.6	Smart Cards	25
3.3.4.6.1	addCTListener	26
3.3.4.6.2	removeCTListener	26
3.3.4.6.3	getSlotId	26
3.3.4.6.4	isCardPresent	27
3.3.4.6.5	openSlotChannel	27
3.3.4.6.6	closeSlotChannel	27
3.3.4.6.7	getATR	28
3.3.4.6.8	reset	28
3.3.4.6.9	sendAPDU	28
3.3.4.6.10	cardInserted	29
3.3.4.6.11	cardRemoved	29
3.3.4.7	Abstract Access to Content	29
3.3.4.7.1	installCallbackContentAccess	29
3.3.4.7.2	abstractContentAccess	30
3.3.4.7.3	replyToContentAccess	30
3.3.4.8	Abstract Access to Rules	31
3.3.4.8.1	Obtain User Rules	31
3.3.4.8.2	Obtain Content Rules	31
3.3.4.8.3	newRules	32
3.3.4.8.4	updateContentRules	32
3.3.4.9	Abstract Access to OPIMA Peers	33
3.3.4.9.1	openConnection	33
3.3.4.9.2	closeConnection	33
3.3.4.9.3	addConnectionListener	33
3.3.4.9.4	sendMessage	34
3.3.4.9.5	newConnection	34
3.3.4.9.6	receiveMessageFromPeer	35
3.3.4.10	Abstract Access to Applications	35
3.3.4.10.1	installCallbackApplication	35
3.3.4.10.2	replyMessage	35
3.3.4.10.3	receiveMessageFromApplication	36
3.3.4.11	Life-cycle Control	36
3.3.4.11.1	initialize	36
3.3.4.11.2	terminate	36
3.3.4.11.3	remove	36
3.3.4.11.4	update	37
3.3.4.12	Locale interface	37
3.3.4.12.1	getTime	37
3.3.4.12.2	getCountry	37
3.3.4.12.3	getLanguage	37
<b>4</b>	<b>IDL definition of the OPIMA APIs (Normative)</b>	<b>38</b>
<b>5</b>	<b>Annexes (Informative)</b>	<b>42</b>

<i>5.1</i>	<i>Acronyms</i>	<i>42</i>
<i>5.2</i>	<i>Definitions</i>	<i>42</i>

## 1 Introduction (Informative)

This specification has been produced by OPIMA (Open Platform Initiative for Multimedia Access), an initiative in the Industry Technical Agreement (ITA) program of the International Electrotechnical Commission (IEC).

OPIMA has been established for the purpose of enabling a framework where content and service providers have the ability to extend the reach of their prospective customers and consumers have the ability to access a wide variety of content and service providers in a context of multiple content protection systems.

This specification describes the elements of *The OPIMA platform*. This platform is targeted at providing value-chain participants with the ability to acquire, supply, process and consume multimedia services on a global basis in accordance with the rights associated with these services. This specification specifically addresses intellectual property management and protection.

This specification presents an architecture and a description of the functions required to implement an OPIMA-compliant system. Further it presents security protocols and a description of Application Programming Interfaces (APIs) and functional behaviours that enable interoperability.

This specification contains four chapters and two Annexes. Chapter 1 is this Introduction. Chapter 2 provides a description of the OPIMA architecture. Chapter 3 provides the normative OPIMA specification with definitions of the APIs and a description of their behaviours. Chapter 3 also provides the OPIMA reference model. Chapter 4 is also normative and contains the IDL definition of the OPIMA APIs. Annexes give acronyms and definitions.

This version 1.0 of OPIMA specification is released for use by interested parties. OPIMA makes no claim as to operational integrity of any implementation derived from this document. This document is provided *as is* with no warranties whatsoever.

OPIMA is keen on receiving comments from implementers and will consider any comment received. Implementers are advised to consult the OPIMA web site

<http://www.cselt.it/opima/>

and subscribe to

[Opima-ver@kim.cselt.it](mailto:Opima-ver@kim.cselt.it)

To subscribe: send a message to [majordomo@kim.cselt.it](mailto:majordomo@kim.cselt.it) with “subscribe opima-ver” in the body of the message.

At a later date OPIMA may specify conformance-testing methodologies. However, actual conformance tests will not be carried out by OPIMA. This specification may be updated in the future without notice.

## 2 The OPIMA Architecture (Informative)

This chapter contains an architectural overview of the OPIMA environment. It presents a framework for inter-operation between OPIMA compliant devices, called OPIMA Peers. The framework is designed so that code can be executed in the user environment without possibility to be tampered with by the environment. Therefore the code and associated content can only be used according to the rules associated with the content. The OPIMA Peer can also be used to interface with the back-end infrastructure.

The OPIMA specification is *device and content independent*. *Content* includes all multimedia types and executables. The specification is independent of all digital content processing devices and content types. The term *Rules* refers to information that stipulates how content may be used on a given device; specifically, *Rules* determine how business models are established. An IPMP (Intellectual Property Management and Protection) system controls access and use of the content by enforcing the rules associated with it. Conditional access systems are particular examples of IPMP systems.

*Protected Content* consists of:

- a *content set*, which may consist of multiple media types;
- an *IPMP system set*, which may consist of multiple IPMP systems;
- a *rules set* that applies under the given IPMP system.

OPIMA acknowledges that there is a world of proprietary domains that have their own governance structures that are defined by a set of IPMP systems. An example is provided by traditional Conditional Access systems. OPIMA may be used to enable interoperability among such system components. OPIMA may also be used as a bridge between proprietary domains.

OPIMA provides tools to exchange a set of authenticated identifiers, called OPIMA Credentials, to enable Protected Content to flow within and between compartments. When OPIMA peers have an on-line connection, the OPIMA Credentials can be exchanged prior to the delivery of the Protected Content. OPIMA Credentials and IPMP Systems can be combined with the Protected Content when the delivery infrastructure does not support on-line connections (e.g. storage media, broadcast media). The OPIMA Credentials contain necessary information that may be used to enable Protected Content flow between compartments. For a given instance of Protected Content that is intended for consumption in two compartments, a Broadcast conditional access compartment and an Internet music delivery compartment, OPIMA Credentials from both compartments are associated with the Protected Content.

OPIMA enables generic interoperability between different applications, devices and IPMP systems belonging to different *compartments*. A compartment is a class of OPIMA enabled devices that share some common elements in their IPMP interfaces and/or architectural components. For example, DVB can be considered as a compartment, which in turn contains other compartments defined by specific IPMP system. Content does not necessarily flow between all compartments, however, OPIMA provides a schema for facilitating content flow between compartments. Compartments can be hierarchical. That is, a compartment can contain sub-compartments. OPIMA may also be used to provide interoperability within compartments even though in this case interoperability may be achieved by different means.

The OPIMA architecture is peer-to-peer. The core OPIMA element is a peer called the *OPIMA*

*Virtual Machine* (OVM). OPIMA provides protocols and infrastructure components that enable secure (trusted) inter-operation amongst these elements. The peer-to-peer interaction based approach allows the efficient implementation of traditional client-server configurations.

OPIMA protocols are those protocols that enable the establishment of Secure Authenticated Channels and the downloading of IPMP Systems into the OVM. Proprietary protocols may be used for the same purpose between peers within a compartment. In this case OPIMA protocols may also be used. OPIMA protocols must be used to download IPMP Systems between different compartments. Communication between IPMP Systems installed on OPIMA peers is effected using proprietary protocols, possibly on top of OPIMA protocols.

The aggregate of secure (trusted) interactions between OVMs represents the OPIMA environment. However, OVMs need not be continuously connected to the rest of the OPIMA environment. An OVM is able to function in a connected (one and two-way) and/or disconnected manner.

## **2.1 Credential mechanisms in the OPIMA environment**

OPIMA uses credentials to enable protected content to flow within and between compartments. Among other things, these credentials certify:

- OVMs, Applications and IPMP systems.
- Compartment identification. This implies:
  - functional capabilities of the compartment, such as cryptographic support, watermarking, and system renewability, etc.
  - security policies of the holder of the credential including the functional capabilities required by the other peer with which the holder wishes to establish a trusted relationship

## **2.2 Required Trusted Institutions**

The OPIMA specification identifies the need for institutions that manage identifiers, renewable security and revocation, compliance, etc. These may be managed by third-party private and public neutral organisations to be determined. OPIMA will *not* manage these infrastructure components.

The structure of these institutions, their mandate, and the format of interaction between them need to be detailed but is outside of the scope of this specification. The trustworthiness of a given compartment and the trustworthiness of content flow between compartments are enabled by the flow of credentials and their secure and predictable interpretation by the OPIMA components. The process conducted by these organisations may include auditing implementations prior to issuing credentials.

The actual number of trusted institutions will depend on how many authority functionalities a specific institution will carry out.

### 2.2.1 Compartment ID issuance authority

Each compartment must be uniquely identified. An authority will be in charge of issuing compartment IDs.

### 2.2.2 Credential issuance authorities

The OPIMA Credentials are issued by neutral OPIMA credential authorities, and implicitly certify the nature of the implementation of the OVM and the device on which it resides and its capabilities.

### 2.2.3 IPMP Systems ID issuance authorities

IPMP Systems need to be uniquely identified in order for an IPMP System to be requested, downloaded and trusted. Companies or organisations may be given the authority to issue IPMP Systems IDs within an ID assigned by one of these authorities.

### 2.2.4 OPIMA Peer ID issuance authorities

OPIMA Peers need to be uniquely identified in order to enable secure interaction between them. Companies or organisations may be given the authority to issue OPIMA Peer IDs within an ID assigned by one of these authorities.

### 2.2.5 Encryption, Signature and Watermarking ID issuance authorities

Encryption, Signature and Watermarking algorithms need to be uniquely identified in order for IPMP Systems and OVMs to communicate among themselves. Only algorithms that are openly used need to be identified.

## 2.3 Back-end Infrastructure

In addition to functional components on OPIMA devices, there may be an interface to back-end processing systems. These include financial, rights and usage clearinghouses. The structure of these components is proprietary; however, they shall use OVM peers to interface to the OPIMA environment.

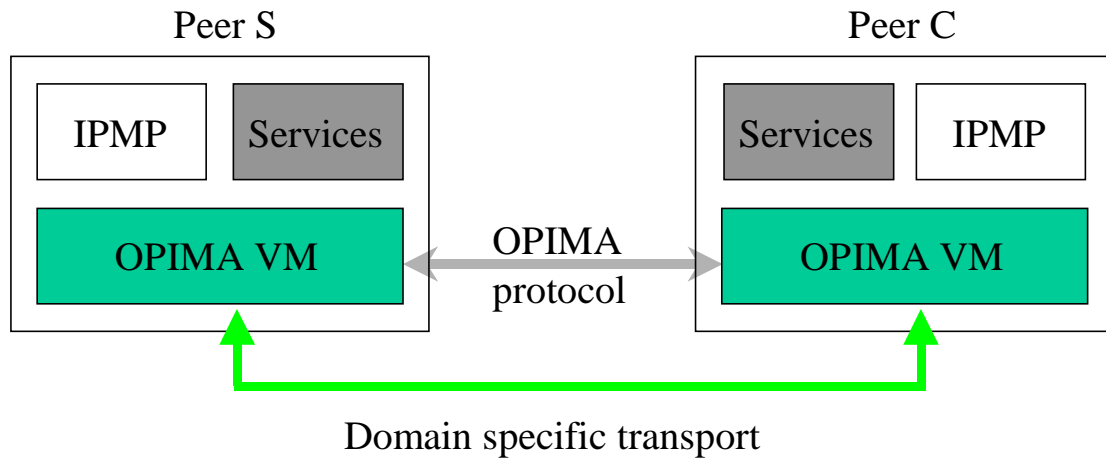
For example, a given compartment may have a network of clearinghouses for the aforementioned operations. A peer device in this compartment will issue audit trails for transmission to the clearinghouse components. The audit trails will result in credentials if successful. The OVM at a given clearinghouse will process the information accordingly. Clearinghouse components may service multiple compartments simultaneously. The back-end infrastructure may also host the renewable security and revocation mechanisms.

## 2.4 Protocols

An important part of the problem set addressed by OPIMA is the secure download of IPMP systems. In this relationship the peers play the roles of client (Peer C) and server (Peer S). An initial negotiation occurs, in which unauthenticated information is exchanged to facilitate the establishment of a Secure Authenticated Channel (SAC). This initial exchange takes place using



non-OPIMA protocols. Then, the OVM establishes the OPIMA peer relationship.



*Fig, 1 -OPIMA protocols*

For example OPIMA peers in a Client Peer - Server Peer set up can communicate in the following way:

1. An application requests the OVM to access protected content
2. The OVM requests the OS to establish initial network connections
3. The OPIMA Secure Authenticated Channel is established on top of this connection
4. The required IPMP System is requested and downloaded by the OVM

### 3 The OPIMA Architecture (Normative)

#### 3.1 OPIMA Protocols

OPIMA protocols are intended to provide a level of interoperability between OPIMA peers. The OPIMA protocols are divided into two layers:

- The first layer, called secure transport, negotiates and implements a SAC.
- The second layer is the OPIMA common message protocol. It uses the functionality provided by the SAC layer. The messages interchanged over this protocol allow it to provide the downloading of IPMP systems services.

##### 3.1.1 First layer: Secure Authenticated Channel

This specification identifies SSL as the reference SAC protocol for inter-compartment communication. Future versions of OPIMA may provide support for a SAC protocol also usable in a broadcast environment. The SAC establishment phase is intended to set up the encryption and decryption mechanisms used by the OVMs (e.g., select a cipher such as DES, select a chaining mode) to securely transmit IPMP Systems between the two correspondent communicating peers.

The SAC establishment phase shall also determine which keys for the aforementioned encryption and decryption mechanisms shall be used.

##### 3.1.2 Second layer: OPIMA Common Message Protocol

The OPIMA Common Message Protocol is used to establish communication between different compartments.

Such a message consists of a command and optional parameters. These parameters are listed in the tables below. The number of bits needed for these parameters are given in parentheses.

Each of the tables below includes the messages sent in sequence from “Sender” to “Recipient”.

The length field always refers to the number of bytes of the field immediately following the length field.

###### 3.1.2.1 Open IPMP System download Message

Sender	Reicipient	Content
Peer 1	Peer 2	OPEN, length(8), IPMPS_ID (var)

###### 3.1.2.2 IPMP System code Messages

Sender	Recipient	Content
Peer 2	Peer 1	MSGDTA (8), length(32), message (var)

In a push case Peer 1 will send the content and Peer 2 will receive it.

### 3.1.2.3 Close of the OPIMA download protocol

Sender	Reiceipient	Content
Peer 1	Peer 2	CLOSE
Peer 2	Peer 1	CLOSE

Both the party sending the IPMP System and the party receiving it are allowed to send this message.

### 3.1.2.4 Message IDs

Message Name	Value (binary)
MSGDTA	00000010
OPEN	00000001
CLOSE	00000011

## 3.2 Credential Formats

This OPIMA 1.0 specification mandates the use of X.509 certificates for credentials. OPIMA is open to reconsider this choice no later than 30<sup>th</sup> of June 2000 based on strong evidence from implementations that another solution, which must be openly accessible, is more suitable for this purpose.

Identification of compartment and identity of OPIMA peer will be included in the certificate.

An OPIMA credential must have the X.509 v.2 format where "subject" is formatted as a structure comprising two fields: CompartmentID and PeerID.

Therefore an OPIMA certificate is a X.509 certificate with the following relevant fields:

- Issuer: IssuerID
- Subject: SubjectID

The format of IssuerID is:

OCTET STRING(2). It contains the Certification authority ID.

The format of SubjectID is:

SEQUENCE{CompartmentID OCTET STRING(4), PeerID OCTET STRING(16)}.

## 3.3 OPIMA Peer

This section describes an OPIMA peer in terms of functional units. Figure 2 depicts the relationship of the pieces described below.

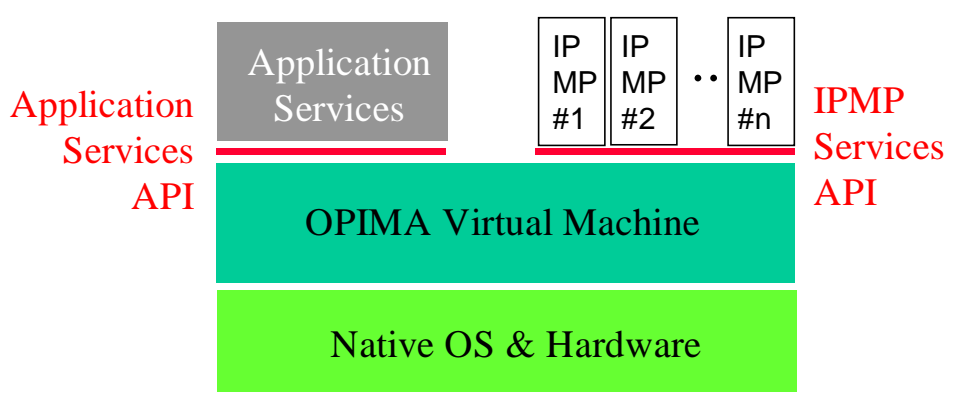


Figure 2: The OPIMA Peer

### 3.3.1 OPIMA Virtual Machine

The OPIMA peer contains a group of basic functional elements that implement the backbone of trust. This is called the OVM. The basic functionality of the OVM allows for application-specific extensions. The OVM is responsible for establishing authenticated, secure channels amongst OPIMA compliant devices.

### 3.3.2 IPMPs

IPMPs are processes that implement Intellectual Property Management & Protection Systems (IPMP-S) inside the OPIMA peer.

### 3.3.3 Application Services API

The OPIMA Application Services API allows services to communicate with the OVM and the IPMP system installed in the OVM. It supports the following functionalities:

- Requesting access to content
- Requesting the installation of an IPMP system component in the OVM
- Querying the installed IPMP system components
- Sending messages to the IPMP system components, among which a message to query content access rights.

The IDL definition of the Application Services API is given in 4.

#### 3.3.3.1 useContent

This method allows services to request that content be made available. The IPMP system decides whether the request is acceptable.

Input Parameters	Values
<i>Content</i> Identification of the content to be used.	URL
<i>Content sink</i> Identification of the sink to which content is handed off.	URL If the <i>content sink</i> is NULL, the sink is implied by the <i>content</i> . It is at the discretion of the IPMP system to decide

	whether this sink is acceptable.
<i>Purpose</i> Indication of the purpose for which the content is made available.	Enumeration for this identifier is given in Table 1 (see below). It is at the discretion of the IPMP system to decide whether this purpose is acceptable.
<i>IPMPsystemID</i> Identification of the IPMP system to be used to make the content available.	An array of bytes containing a unique ID assigned by a registration authority. If the indicated <i>IPMP system</i> is not available in the OVM, the OVM may try to obtain the corresponding IPMP system component, depending on OVM policy. If the <i>IPMP system</i> is NULL, the OVM will decide which IPMP system to use if any, depending on the OVM policy and possibly after user interaction.
<b>Return Variable</b>	<b>Values</b>
<i>Result</i>	32 bit integer, that can be either positive or negative. A positive value indicates session ID that can be used by the application to match subsequent asynchronous responses from the OVM. Negative values indicate that an error occurred and the reason of failure.
<b>Asynchronous Responses</b>	<b>Values</b>
<i>Success</i>	This asynchronous response is issued at the moment that the <i>content</i> is available. It can be used for synchronisation purposes.
<i>Failure</i>	This asynchronous response is issued at the moment that the <i>content</i> is not available or is no longer available. The reason for the failure is embedded: <ul style="list-style-type: none"> <li>• Content not found at indicated URL.</li> <li>• Content sink not available.</li> <li>• Indicated IPMP system not available.</li> <li>• Content access disallowed by IPMP system.</li> <li>• IPMP specific string to be interpreted by the application.</li> </ul> Table of values is given in the IDL definition.

Table 1 - Actions the OVM executes on content

IMPLICIT	The purpose is implied by the content source and the content destination
RENDER	Rendering the content on the OPIMA Peer
STOP	Stop ongoing action
COPY	Copy content under control of the IPMP System
MOVE	Move content under control of the IPMP System
EDIT	Edit content under control of the IPMP System
FORWARD	Forward content outside of the OPIMA Peer under the control of the IPMP System
RELEASE	Release content outside of the OPIMA Peer
OTHER	Anything else

### 3.3.3.2 getIpmpSystem

This method allows services to request that an IPMP system component is installed in the OVM. If the IPMP system is already installed in the OVM, the application is simply registered as a client of that IPMP system. Note that no compartment indication is passed by the application to the OVM, since the OVM knows its own compartment.

Input Parameters	Values
<i>IPMPsystemID</i> Identification of the IPMP system to be installed.	An array of bytes containing a unique ID assigned by a registration authority.
<i>Source</i> Identification of the source for the IPMP system component.	URL If <i>Source</i> is NULL, then the source is implied by <i>IPMP System</i> .
Return Variable	Values
<i>Result</i>	32 bit integer, that can be either positive or negative. A positive value indicates session ID that can be used by the application to match subsequent asynchronous responses from the OVM. Negative values indicate that an error occurred and the reason of failure.
Asynchronous Responses	Values
<i>Success</i>	This asynchronous response is issued at the moment that the IPMP system component has been successfully loaded.
<i>Failure</i>	This asynchronous response is issued at the moment that the IPMP system component is not available or is no longer available. The reason for the failure is embedded: <ul style="list-style-type: none"><li>• IPMP system not present.</li><li>• Source not available.</li><li>• Downloading refused by source.</li><li>• Not enough resources to install.</li></ul> Table of values is given in the IDL definition.

### 3.3.3.3 queryOVM

This method allows applications to query the OVM. Query OVM provides the list of IPMP systems that are applicable to a certain content and gives an indication of which IPMP systems are already available within the OVM.

Input Parameters	Values
<i>ContentId</i> Identification of the content to be used	URL
<i>Purpose</i> Identification of the purpose for which the content is	Same as in “useContent”

intended to be used	
<b>Output Parameters</b>	<b>Values</b>
<i>IpmpsID[][]</i> Output of the function	List of alternative sets of IPMP systems that are needed by the content in order for the OPIMA peer to perform the intended “purpose”, associated with the indication of their current status in the OVM (present/missing). IPMP systems are identified by IPMP system IDs, as defined above.
<b>Return Variable</b>	<b>Values</b>
<i>Result</i>	32 bit integer, that can be either positive or negative. A positive value indicates session ID that can be used by the application to match subsequent asynchronous responses from the OVM. Negative values indicate that an error occurred and the reason of failure.

#### 3.3.3.4 sendMessageToIPMP

This method allows services to send messages to the IPMP systems installed in the OVM and to receive answers.

<b>Input Parameters</b>	<b>Values</b>
<i>IPMPsystemID</i> Identification of the IPMP system to which the message is addressed.	An array of bytes containing a unique ID assigned by a registration authority.
<i>Message Type</i> Identification of the message type	<ul style="list-style-type: none"> <li>• Content Query</li> <li>• IPMP system proprietary</li> <li>• NULL message (to allow an application register itself as a receiver of messages without actually sending any message)</li> </ul> Table of values is given in the IDL definition.
<i>Message</i>	<ul style="list-style-type: none"> <li>• URL (in case of a content query message)</li> <li>• Data passed to the IPMP component.</li> </ul>
<b>Return Variables</b>	<b>Values</b>
<i>Result</i>	32 bit integer, that can be either positive or negative. A positive value indicates session ID that can be used by the application to match subsequent asynchronous responses from the OVM. Negative values indicate that an error occurred and the reason of failure.
<b>Asynchronous Responses</b>	<b>Values</b>
<i>Content query response</i>	<ul style="list-style-type: none"> <li>• Content not available.</li> <li>• String for display to end-user.</li> <li>• Data</li> </ul>

#### 3.3.3.5 notifyEvent

This asynchronous response is issued by the OVM to the application to notify that a certain event

has occurred; it can be used for synchronisation purposes.

Input Parameters	Values
<i>SessionID</i> An identifier provided by the OVM which refers to the action to which this is a response	Same value previously returned by either <i>UseContent</i> , <i>getIpmpSystem</i> , or <i>sendMessageToIPMP</i> .
<i>Status</i> Indicates success or failure, and reasons of failure	SUCCESS if status = 0 ErrorCode if status < 0
<i>Message</i> An IPMP specific string to be interpreted by the application	Either NULL or an IPMP specific string

### 3.3.4 IPMP Services API

This section contains the IPMP Services API, this being the interface between the IPMP and the OPIMA Virtual Machine (OVM). The IPMP Services API is the only way of communication between the IPMP Systems on the one side and the OVM and the external world on the other side. An IDL definition of this API is provided in 4.

#### 3.3.4.1 User Interface methods

The purpose of these methods is to obtain information directly from the user. This feature can be used for non-repudiation.

##### 3.3.4.1.1 sendMessageToUser

Input Parameters	Values
<i>MessageText</i> text to be displayed to the user.	A string of text
<i>Listener</i> callback function that delivers the user response to the IPMP system.	Function address
Return Variable	Values
<i>Status</i>	<i>Status</i> > 0: The returned value is a <i>SessionID</i> provided by the OVM. <i>Status</i> < 0: The returned value is an ErrorCode; can be one of: PERMISSION_DENIED NO_RESOURCES WRONG_PARAMETERS



#### 3.3.4.1.2 receiveMessageFromUser

Input Parameters	Values
<i>SessionID</i> Identifier of an action to which this is a reply	Any positive int value
<i>Response</i> Text returned by the user (e.g. a password).	A string of text

#### 3.3.4.2 Secure Storage Interface

This set of functions allows an IPMP system to use storage capabilities provided by the OVM. Information stored upon an IPMP system request will be made available to other instantiations of the same IPMP system. The OVM enforces that an IPMP system is not allowed to access information stored by other IPMP systems. Since memory can be a scarce resource in some OVM implementations, it is not guaranteed that stored information will persist indefinitely.

##### 3.3.4.2.1 secureStoreData

This function requests OVM to store information contained in an array of bytes. Some OVM implementations may decide to maintain storage of this information, even when the IPMP system is terminated.

Input Parameters	Values
<i>DataReference</i> An identifier that allows different instantiations of the same IPMP system to access stored data.	<ul style="list-style-type: none"><li>• Enumeration of these identifiers is proprietary to the IPMP system.</li><li>• This parameter is unique per IPMP.</li></ul>
<i>Data</i> A reference to data to be stored	<ul style="list-style-type: none"><li>• This parameter is treated as an unformatted array of bytes, which are not interpreted by the OVM.</li></ul>
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0; Otherwise one of the following error codes: METHOD_NOT_AVAILABLE (in this OVM) INSUFFICIENT_RESOURCES PERMISSION_DENIED

##### 3.3.4.2.2 secureRetrieveData

This function requests OVM to retrieve information that was previously stored under *DataReference* by an instantiation of the calling IPMP system.

Input Parameters	Values
<b>DataReference</b> An identifier that allows different instantiations of the same IPMP system to access stored data.	<ul style="list-style-type: none"> <li>Enumeration of these identifiers is proprietary to the IPMP system.</li> <li>This parameter is unique per IPMP.</li> </ul>
<b>Buffer</b> Memory location, where stored data is copied into.	<ul style="list-style-type: none"> <li>This reference is provided by the IPMP system.</li> </ul>
Return Variable	Values
<b>Status</b>	SUCCESS if <i>Status</i> = 0; Otherwise one of the following error codes: METHOD_NOT_AVAILABLE (in this OVM) INSUFFICIENT_RESOURCES (buffer is too small) NO_DATA PERMISSION_DENIED

#### 3.3.4.2.3 secureDeleteData

This function requests OVM to destroy information that was previously stored under *DataReference* by an instantiation of the calling IPMP system.

Input Parameters	Values
<b>DataReference</b> An identifier that allows different instantiations of the same IPMP system to access stored data.	<ul style="list-style-type: none"> <li>Enumeration of these identifiers is proprietary to the IPMP system.</li> <li>This parameter is unique per IPMP.</li> </ul>
Return Variable	Values
<b>Status</b>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3 Encryption and Decryption Engines

The IPMP Systems can access standard cryptographic algorithms (implemented in hardware or software) through the following functions.

##### 3.3.4.3.1 queryEncryptionAlgorithms

Return Variables	Values
<b>AlgorithmList</b>	Array of strings indicating the supported encryption and decryption algorithms.

#### 3.3.4.3.2 encrypt

This method encrypts a set of bytes. It is intended primarily to encrypt management messages.

Input Parameters	Values
<i>Algorithm</i> Identification of the algorithm used in the encryption.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>Key</i> The key used in the encryption.	An array of bytes
Input/Output Parameters	Values
<i>Data.</i> Input: the data to be encrypted Output: the encrypted data	An array of bytes
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3.3 initEncryption

This method initialises content encryption. The implementation of this method is able to access content on the basis of the *clearContentId* parameter. The return variable is a reference to the encrypted content or (if negative) an error code.

Input Parameters	Values
<i>Algorithm</i> Identification of the algorithm used in the encryption.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>ClearContentId</i> Reference to the content stream that must be encrypted	Integer
Return Variable	Values
<i>EncryptedContentId</i>	a reference to the encrypted content or (if negative) an error code.

#### 3.3.4.3.4 updateEncryptionKeys

This method starts or continues content encryption using new or modified keys. An array of keys is used to allow for key changes.

Input Parameters	Values
<i>Keys</i> The keys used in the encryption.	An array of keys (that is, an array of array of bytes).
<i>ClearContentId</i> Reference to the content stream	Integer

that must be encrypted	
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3.5 stopEncryption

<b>Input Parameters</b>	<b>Values</b>
<i>ClearContentId</i> Reference to the content stream	Integer
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3.6 decrypt

This method decrypts a set of bytes. It is intended primarily to decrypt management messages.

<b>Input Parameters</b>	<b>Values</b>
<i>Algorithm</i> Identification of the algorithm used in the decryption.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>Key</i> The key used in the decryption.	An array of bytes
<b>Input/Output Parameters</b>	<b>Values</b>
<i>Data.</i> Input: the data to be decrypted Output: the decrypted data	An array of bytes
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3.7 initDecryption

This method initialises content decryption. The implementation of this method is able to access content on the basis of the clearContentId parameter. The return variable is a reference to the decrypted content or (if negative) an error code.

<b>Input Parameters</b>	<b>Values</b>
<i>Algorithm</i> Identification of the algorithm used in the decryption.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>EncryptedContentId</i>	Integer

Reference to the content stream that must be decrypted	
<b>Return Variable</b>	<b>Values</b>
<i>DecryptedContentId</i>	a reference to the decrypted content or (if negative) an error code.

#### 3.3.4.3.8 updateDecryptionKeys

This method starts or continues content decryption using new or modified keys. An array of keys is used to allow for key changes.

<b>Input Parameters</b>	<b>Values</b>
<i>Keys</i> the keys used in the decryption.	An array of keys (that is, an array of array of bytes).
<i>EncryptedContentId</i> Reference to the content stream that must be decrypted	Integer
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.3.9 stopDecryption

<b>Input Parameters</b>	<b>Values</b>
<i>EncryptedContentId</i> Reference to the content stream	Integer
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

### 3.3.4.4 Signature Engines

This interface is used by IPMP Systems that wish to use public key cryptography in order to perform digital signature processing

#### 3.3.4.4.1 querySignatureAlgorithms

<b>Return Variables</b>	<b>Values</b>
<i>AlgorithmList</i>	Array of strings indicating the supported digital signature algorithms.

A Registration Authority will issue standard identifiers for established algorithms.

#### 3.3.4.4.2 verifySignature

This method verifies a digital signature. It is intended primarily to verify the authenticity of management messages.

<b>Input Parameters</b>	<b>Values</b>
<i>Algorithm</i> The algorithm used in the verification.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>PublicKey</i> The public key used in the verification.	An array of bytes
<i>Data</i> The data that has been signed.	An array of bytes
<i>Signature</i> The digital signature to be verified	An array of bytes
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	TRUE if <i>Status</i> = 1 FALSE if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.4.3 verifyContentSignature

This method verifies a digital signature. It is intended to verify the authenticity of content. The implementation of this method is able to access content on the basis of the *contentId* parameter.

<b>Input Parameters</b>	<b>Values</b>
<i>Algorithm</i> The algorithm used in the verification.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>PublicKey</i> The public key used in the verification.	An array of bytes
<i>ContentId</i> reference to the content that has been signed.	An integer value
<i>Signature</i> The digital signature to be verified	An array of bytes
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	TRUE if <i>Status</i> = 1 FALSE if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.4.4 generateSignature

This method generates a digital signature. It is intended primarily to sign management messages.

Input Parameters	Values
<i>Algorithm</i> The algorithm used in the verification.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>PrivateKey</i> The private key used in the generation.	An array of bytes
<i>Data</i> The data to be signed.	An array of bytes
Output Parameters	Values
<i>Signature</i> The resulting digital signature	An array of bytes
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.4.5 generateContentSignature

This method generates a digital signature. It is intended to sign content. The implementation of this method is able to access content on the basis of the *contentId* parameter.

Input Parameters	Values
<i>Algorithm</i> The algorithm used in the verification.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>PrivateKey</i> The private key used in the generation..	An array of bytes
<i>ContentId</i> reference to the content that has been signed.	An integer value
Output Parameters	Values
<i>Signature</i> The resulting digital signature	An array of bytes
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.5 Watermarking Engines

In case the OVM provides access to watermarking engines the following interfaces are used.

#### 3.3.4.5.1 queryWatermarkAlgorithms

Return Variables	Values
<i>AlgorithmList</i>	Array of strings indicating the supported watermarking algorithms.

A Registration Authority will issue standard identifiers for established algorithms.

#### 3.3.4.5.2 extractWatermark

This method starts watermark extraction from content identified by *contentId*. If the content is a stream, watermark extraction can be stopped by calling *stopWatermarkExtraction*. A call back interface is used to transfer the watermark.

Input Parameters	Values
<i>Algorithm</i> The watermarking algorithm.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>ContentId</i> Reference to the watermarked content	An integer value
<i>Listener</i> Callback function implemented by the IPMP System to accept the asynchronous responses that would be issued by the OVM whenever watermark is detected	An identifier of a function
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.5.3 stopWatermarkExtraction

This method stops watermark extraction from a content stream identified by *contentId*.

Input Parameters	Values
<i>ContentId</i> Reference to the content stream from which the a watermark is extracted.	An integer value
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.5.4 newWatermark



This is the call back interface for the watermark engine. It should be implemented by IPMP components that call the watermark engine.

Input Parameters	Values
<i>ContentId</i> Reference to the watermarked content	An integer value
<i>Watermark</i> Value of the extracted watermark	An array of bytes

#### 3.3.4.5.5 insertWatermark

This method starts watermark insertion into content identified by *sourceContentId*. The return variable is the *sinkContentId*. If the content is a stream, watermark insertion can be stopped by calling *stopWatermarkInsertion*.

Input Parameters	Values
<i>Algorithm</i> The watermarking algorithm.	A text string
<i>Params</i> The parameters of the algorithm	An array of bytes
<i>SourceContentId</i> Reference to the content to be watermarked.	An integer value
Return Variable	Values
<i>SinkContentId</i>	a reference to the watermarked content or (if negative) an error code.

#### 3.3.4.5.6 stopWatermarkInsertion

This method stops watermark extraction from a content stream identified by *contentId*.

Input Parameters	Values
<i>SourceContentId</i> Reference to the content to be watermarked.	An integer value
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6 Smart Cards

Smart cards accessible through local smart card readers can be accessed by means of this API.

The following two structures represent messages in ISO 7816 format:

```
struct CommandAPDU
{
    byte cla;
    byte ins;
    byte P1;
    byte P2;
    int lc;           // The length of the data field of the APDU.
    int le;           // The expected length of the ResponseAPDU.
    byte[] data;
}
```

```
struct ResponseAPDU
{
    byte sw1
    byte sw2
    byte[] data
}
```

#### 3.3.4.6.1 addCTListener

This method adds a Card Terminal listener to receive an indication of card insertion or removal from this card terminal.

<b>Input Parameters</b>	<b>Values</b>
<i>Listener</i> The listener to add	The function address of either <b>cardInserted</b> or <b>cardRemoved</b>
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.2 removeCTListener

Removes a CTListener so it no longer receives CardTerminalEvents from this card terminal

<b>Input Parameters</b>	<b>Values</b>
<i>Listener</i> The CTListener to remove.	The function address of either <b>cardInserted</b> or <b>cardRemoved</b>
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.3 getSlotId

Returns the list of slots belonging to this Card Terminal.

Output Parameters	Values
<i>SlotId</i> The array of slot identifiers.	An array of int
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.4 isCardPresent

Checks whether there is a smart card present in a particular slot

Input Parameters	Values
<i>SlotId</i> The slot to check for a card	A positive integer value
Return Variable	Values
<i>Status</i> Indicates if there is a smart card inserted in the slot.	TRUE if <i>Status</i> = 1 FALSE if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.5 openSlotChannel

Opens a SlotChannel on Slot number slotID.

Input Parameters	Values
<i>SlotId</i> The number of the slot for which a SlotChannel is requested.	A positive integer value
Return Variable	Values
<i>Status</i> Identification of the session to the smart card or error code.	SlotSessionId if <i>Status</i> > 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.6 closeSlotChannel

Closes a SlotChannel identified by slotSessionId.

Input Parameters	Values
<i>SlotSessionId</i> The smart card session to close.	A positive integer value
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.7 getATR

Returns the answer-to-reset (ATR) response of the card inserted in slot slotID.

Input Parameters	Values
<i>SlotId</i> The slot identifier	A positive integer value
<i>Ms</i> A timeout in milliseconds.	A positive integer value; zero is interpreted as “no timeout”
Output Parameters	Values
<i>ATR</i> The ATR response in form of a byte array.	An array of bytes
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.8 reset

Resets a smart card inserted in a slot.

Input Parameters	Values
<i>SlotSessionId</i> The open slot channel attached to the slot.	A positive integer value
<i>Timeout</i> A timeout in milliseconds.	A positive integer value; zero is interpreted as “no timeout”
Output Parameters	Values
<i>ATR</i> The ATR response in form of a byte array.	An array of bytes
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.6.9 sendAPDU

Sends a command to the smart card and waits for the response.

Input Parameters	Values
<i>SlotSessionId</i> The open slot channel attached to the slot.	A positive integer value
<i>CommandAPDU</i> The CommandAPDU to send.	See the “CommandAPDU” structure defined above.

<i>Timeout</i> A timeout in milliseconds.	A positive integer value; zero is interpreted as “no timeout”
<b>Output Parameters</b>	<b>Values</b>
<i>ResponseAPDU</i> The ResponseAPDU as received from the card.	See the “ResponseAPDU” structure defined above.
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

The following call backs are to be implemented by the IPMP system:

#### 3.3.4.6.10 cardInserted

Notifies that a smart card has been inserted in a slot identified by slotId.

<b>Input Parameters</b>	<b>Values</b>
<i>SlotId</i> The slot identifier.	A positive integer value

#### 3.3.4.6.11 cardRemoved

Notifies that a smart card has been removed from a slot identified by slotId.

<b>Input Parameters</b>	<b>Values</b>
<i>SlotId</i> The slot identifier.	A positive integer value

### 3.3.4.7 Abstract Access to Content

#### 3.3.4.7.1 installCallbackContentAccess

This function is a request from the IPMP system to the OVM, to install a callback function for this set of methods.

<b>Input Parameters</b>	<b>Values</b>
<i>Listener</i> Callback function implemented by the IPMP system to accept an asynchronous response issued by the OVM.	An identifier of a function.
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS (if <i>Status</i> = 0); Otherwise one of the following error codes: INSUFFICIENT_RESOURCES

	PERMISSION_DENIED
--	-------------------

#### 3.3.4.7.2 abstractContentAccess

This is a previously installed callback function, through which the OVM calls the IPMP system for a specific purpose. Usually this function should be called by the OVM upon request from the Application (see *useContent* function), thus implementing an indirect call of the IPMP system from an Application.

Input Parameters	Values
<i>ContentId</i> An identifier of the Content.	The result of the translation of the URL that has been passed from the Application (see <i>useContent</i> function).
<i>Content sink</i> Identification of the sink to which content is handed off.	URL If the <i>content sink</i> is NULL, the sink is implied by the <i>content</i> . It is at the discretion of the IPMP system to decide whether this sink is acceptable.
<i>Purpose</i> An identifier of the purpose to access content.	<ul style="list-style-type: none"> <li>Enumeration for this identifier is given in Table 1 (see 3.3.3.1).</li> <li>The OPIMA definition for this enumeration includes a set of <i>purposes</i>, as well as the possibility to accommodate proprietary values.</li> </ul>
<i>SessionId</i> An identifier of the session generated by the OVM	<ul style="list-style-type: none"> <li>This value is generated by the OVM and passed as input to the IPMP system.</li> <li>It will be used by <i>replyToContentAccess</i> to identify the session.</li> </ul>
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.7.3 replyToContentAccess

This function implements the response from the IPMP System to *abstractContentAccess*. This will be translated by the OVM into a call of the “*notifyEvent*” callback function, which is part of the Application Services API.

Input Parameters	Values
<i>SessionId</i> An identifier of the session.	This value has been received from the OVM through the call to <i>abstractContentAccess</i> .
<i>Status</i> A status indication for the application	SUCCESS ( <i>Status</i> = 0) ErrorCode ( <i>Status</i> < 0)
<i>Message</i> A reference containing the requested action.	An array of bytes

Return Variable	Values
<i>Status</i> A status indication for the IPMP system	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.8 Abstract Access to Rules

In the cases where the content is not directly accessible to an IPMP System in order to be processed, the rules may be accessed using this interface.

##### 3.3.4.8.1 obtainUserRules

This function retrieves the rules associated with a user.

Input Parameters	Values
<i>UserId</i> An identifier of the user.	This parameter is treated as an unformatted array of bytes.
<i>Listener</i> An identifier of a function.	Callback function implemented by the IPMP system to accept the asynchronous response that will be issued by the OVM.
Return Variable	Values
<i>Status</i>	<i>Status</i> > 0: The returned value is a <i>sessionId</i> provided by the OVM to be used by the IPMP system to manage the asynchronous response. This implies that the call has been successfully accepted by the OVM. <i>Status</i> < 0: METHOD_NOT_AVAILABLE (in this OVM implementation) INSUFFICIENT_RESOURCES PERMISSION_DENIED INVALID_PARAMETER

##### 3.3.4.8.2 obtainContentRules

This function retrieves the rules associated with a content.

Input Parameters	Values
<i>sourceContentId</i> An identifier of the content.	This parameter has been passed to the IPMP system in the <i>abstractContentAccess</i> function.
<i>listener</i> An identifier of a function.	Callback function implemented by the IPMP system to accept the asynchronous response that will be issued by the OVM.
Return Variable	Values

<i>Status</i>	<p><i>Status</i> &gt; 0: The returned value is a <i>sessionId</i> provided by the OVM to be used by the IPMP system to manage the asynchronous response. This implies that the call has been successfully accepted by the OVM.</p> <p><i>Status</i> &lt; 0: METHOD_NOT_AVAILABLE INSUFFICIENT_RESOURCES PERMISSION_DENIED INVALID_PARAMETER</p>
---------------	---

#### 3.3.4.8.3 newRules

This is a callback function implemented by the IPMP system. The intended use of this function is a listener for either *obtainContentRules* or *obtainUserRules*. The *SessionId* is used to associate a request to a response.

Input Parameters	Values
<p><i>SessionId</i> An identifier of the asynchronous session between the IPMP system and the OVM.</p>	This parameter has been received as a return value from <i>obtainUserRules</i> or <i>obtainContentRules</i> .
<p><i>Buffer</i> An identifier of the rules requested.</p>	<p>Requested rules. <i>Buffer</i> will be reused by the OVM.</p>

Note: To prevent memory leakage in the OVM, content in the *Buffer* parameter should be copied into the IPMP system memory space before returning from this callback function.

#### 3.3.4.8.4 updateContentRules

This function updates the rules associated with a *content*.

Input Parameters	Values
<p><i>SinkContentId</i> An identifier of the sink for the content.</p>	Any integer value
<p><i>Buffer</i> An identifier of a function.</p>	<i>Buffer</i> contains the update rule that must be forwarded to the Content sink.
Return Variable	Values
<i>Status</i>	<p>SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> &lt; 0</p>



#### 3.3.4.9 Abstract Access to OPIMA Peers

The purpose of this interface is to exchange IPMP system proprietary management and control messages between OPIMA peers. This interface can also be used for communication between local IPMP Systems.

##### 3.3.4.9.1 openConnection

This method opens a connection to an OPIMA peer.

Input Parameters	Values
<i>Peer</i> URL of the destined OPIMA peer.	An URL (a special case is “localhost”)
<i>IPMPsystemID</i> Identification of the destined IPMP system.	An array of bytes containing a unique ID assigned by a registration authority.
<i>ConnectionSpec</i> specification of the connection to be opened.	The connection specification ID specifies which type of connection to use <ul style="list-style-type: none"><li>• ID=0 cleartext</li><li>• ID=1 OPIMA SAC</li><li>• ID &gt; 1 compartment specific</li></ul>
<i>Listener</i> A function which listens to messages sent by the other peer	Function reference
Return Variable	Values
<i>Status</i> Either an identification of the connection or an error code	ConnectionId if <i>Status</i> > 0 ErrorCode if <i>Status</i> < 0

##### 3.3.4.9.2 closeConnection

This method closes a connection to an OPIMA peer.

Input Parameters	Values
<i>ConnectionId</i> Identification of the connection as returned earlier by the openConnection method.	A positive integer value.
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

##### 3.3.4.9.3 addConnectionListener

This method installs a listener for new connections that other OPIMA peers try to establish.

Input Parameters	Values
<i>Peer</i>	An URL

URL of the originating OPIMA peer. If NULL, all OPIMA peers can establish a connection.	
<i>IPMPsystemID</i> Identification of the originating IPMP system.	An array of bytes containing a unique ID assigned by a registration authority.
<i>Listener</i> A function which listens to connection requests from other OPIMA Peers	Function reference
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.9.4 sendMessage

This method sends a message on the designated connection.

<b>Input Parameters</b>	<b>Values</b>
<i>ConnectionId</i> identification of the connection as returned earlier by the openConnection method.	A positive integer value.
<i>Message</i> Message to send.	An array of bytes.
<b>Return Variable</b>	<b>Values</b>
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

The following call backs are to be implemented by the IPMP system.

#### 3.3.4.9.5 newConnection

This method signals the establishment of a new connection.

<b>Input Parameters</b>	<b>Values</b>
<i>Peer</i> URL of the originating OPIMA peer.	An URL
<i>IPMPsystemID</i> Identification of the destined IPMP system.	An array of bytes containing a unique ID assigned by a registration authority.
<i>ConnectionId</i> Identification of the connection.	A positive integer value.
<b>Return Variable</b>	<b>Values</b>
<i>Status</i> Status indication provided by the IPMP system to the OVM.	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.9.6 receiveMessageFromPeer

This method receives a message on the designated connection.

Input Parameters	Values
<i>ConnectionId</i> Identification of the connection as returned earlier by the openConnection method.	A positive integer value.
<i>Message</i> The received message.	An array of bytes.
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

#### 3.3.4.10 Abstract Access to Applications

Applications may access the local IPMP Systems if the IPMP Systems have previously installed the appropriate callback via this interface.

##### 3.3.4.10.1 installCallbackApplication

This is an IPMP System request to the OVM to install a callback function for the application.

Input Parameters	Values
<i>Listener</i> The listener of messages.	The function address of <i>receiveMessageFromApplication</i>
Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

##### 3.3.4.10.2 replyMessage

The IPMP system replies to a message received by means of a *receiveMessageFromApplication* callback with this method.

Input Parameters	Values
<i>SessionId</i> An identifier received in a <i>receiveMessageFromApplication</i> to which this is a response.	A positive integer value
<i>Status</i> Indicates success or failure, and reasons of failure.	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0
<i>Message</i> An IPMP specific string to be interpreted by the application.	Either NULL or an IPMP specific string
Return Variable	Values

<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0
---------------	--

The following callback is to be implemented by the IPMP system:

#### 3.3.4.10.3 receiveMessageFromApplication

Callback function by which the application can access the IPMP system (via the OVM). It is the OVM which calls the IPMP system by means of this method.

Input Parameters	Values
<i>SessionId</i> An identifier provided by the OVM which refers to this action.	A positive integer value.
<i>MessageType</i> An IPMP specific string to be interpreted by the IPMP.	An enumeration type with three values: CONTENT_QUERY, PROPRIETARY_MESSAGE, NULL_MESSAGE.
<i>Message</i> An IPMP specific string to be interpreted by the IPMP.	Either NULL or an IPMP specific string.

#### 3.3.4.11 Life-cycle Control

This interface is used to control the execution state of the IPMP System. The following functions are to be implemented by IPMP systems for the purpose of life cycle control:

##### 3.3.4.11.1 initialize

This method is called immediately after installation of the IPMP system. It allows the IPMP system to take care of any necessary installation procedures.

Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

##### 3.3.4.11.2 terminate

This method is called before de-installation of the IPMP system. It allows the IPMP system to take care of all necessary termination procedures. After this method has returned, the IPMP system will be removed from the OVM.

Return Variable	Values
<i>Status</i>	SUCCESS if <i>Status</i> = 0 ErrorCode if <i>Status</i> < 0

The following functions are to be implemented by the OVM for the purpose of life cycle control:

##### 3.3.4.11.3 remove

This method removes the IPMP system upon its request. It never returns.

#### 3.3.4.11.4 update

This method removes the current instance of the IPMP system and requests downloading of a new instance. It never returns.

Input Parameters	Values
<i>Source</i> Identification of the source for the new IPMP system.	A URL.

#### 3.3.4.12 Locale interface

This interface lets an IPMP system component running on top of an OVM know in a secure way where and when it is running.

##### 3.3.4.12.1 getTime

This method returns either an error number (if negative) or the number of seconds since "the Epoch" (the time 0 hours, 0 minutes, 0 seconds, January 1, 1970 Coordinated Universal Time - UTC). This introduces a discontinuity (a Y2K-like problem) in year 2038 that must be handled by the IPMPS component making use of this function.

Return variable	Values
<i>Status</i>	Time information if <i>Status</i> $\geq 0$ ErrorCode if <i>Status</i> $< 0$

##### 3.3.4.12.2 getCountry

This method returns either an empty string or an uppercase ISO 3166 2-letter code representing the country for which the OVM has been configured.

Return variable	Values
<i>Country</i>	NULL or an uppercase ISO 3166 2-letter country code

##### 3.3.4.12.3 getLanguage

This method returns either an empty string or the language code for this OVM, which will be a lowercase ISO 639 code.

Return variable	Values
<i>Language</i>	NULL or a lowercase ISO 639 language code

## 4 IDL definition of the OPIMA APIs (Normative)

```
/*
 * IDL definition of the OPIMA interfaces.
 * Note: the OPIMA peer is composed by Applications, IPMP systems, and the OPIMA VM.
 * The Applications implement: ApplicationServicesListener, ErrorCodes.
 * the IPMP systems implement: all the IPMP Listeners interfaces, ErrorCodes.
 * the OPIMA VM implements: ApplicationServices, all the IPMP Services interfaces, ErrorCodes.
 */
module OPIMA {
    /*--- return values (positive values); these can be either Boolean values or Session IDs
    ---*/

    interface BooleanResult {
        const long IS_TRUE = 1;
        const long IS_FALSE = 0;
    };

    interface SessionResult {
        const long SUCCESS = 0;
    };

    /*--- error codes (negative values). These can be extended for compartment specific
    errors ---*/

    interface ErrorCodes {
        const long CONTENT_NOT_FOUND = -1;           // Content not found at the indicated
URL.
        const long SINK_NOT_AVAILABLE = -2;         // Content sink not available.
        const long CONTENT_ACCESS_NOT_ALLOWED = -3;  // Content access prohibited by IPMP
System
        const long UNKNOWN_SOURCE = -4;             // Source could not be
reached / source unknown
        const long BUSY_SOURCE = -5;                // Source not available at this
moment / source busy
        const long INVALID_SOURCE = -6;             // Source was not
authenticated as a valid source for the requested IPMP system.
        const long IPMPS_NOT_AVAILABLE = -7;        // Requested IPMP system is not
available at the source.
        const long DOWNLOAD_NOT_ALLOWED = -8;       // Source refuses to download the
IPMP system to this OPIMA peer.
        const long INSUFFICIENT_RESOURCES = -9;      // Insufficient resources to
perform the requested action.
        const long TIMEOUT_EXPIRED = -10;           // A timeout expired before
completion of the task.
        const long PERMISSION_DENIED = -11;         // The requested action is not
allowed.
        const long WRONG_PARAMETER_1 = -12;         // Error in first parameter
        const long WRONG_PARAMETER_2 = -13;         // Error in second parameter
        const long WRONG_PARAMETER_3 = -14;         // Error in third parameter
        const long WRONG_PARAMETER_4 = -15;         // Error in fourth parameter
        const long WRONG_PARAMETER_5 = -16;         // Error in fifth parameter
        const long METHOD_NOT_AVAILABLE = -17;       // The requested function has
not been implemented on this OVM
        const long CARD_NOT_PRESENT = -18;          // The smart card has been removed
        const long COMMUNICATION_ERROR = -19;       // Transmission error between the
terminal and the smart card
        const long RULES_NOT_AVAILABLE = -20;       // The OVM was not able to extract
rules from content
        const long CANNOT_UPDATE_RULES = -21;       // The OVM was not able/allowed to
modify rules
        const long APPLICATION_TERMINATED = -22;    // The application was terminated
    };

    /*--- complex types used as method parameters in more than one interface ---*/

    typedef string URL;
    typedef sequence <octet, 256000> Buffer;
    typedef sequence <string, 1024> List;
    typedef sequence <octet, 255> IPMPSid;

    enum Purpose {IMPLICIT, RENDER, STOP, COPY, MOVE, EDIT, FORWARD, RELEASE, OTHER};
    enum MessageType {CONTENT_QUERY, NULL_MESSAGE, PROPRIETARY_FORMAT};

    /*----- listeners defined in Application Services -----*/

    interface ApplicationServicesListener {
        void notifyEvent(in long sessionId, in long status, in Buffer message);
    };
}
```

```

/*----- Application Services -----*/

interface ApplicationServices {
    struct IPMPSidContainer {
        IPMPSid id;
        boolean exists;
    } ;

    typedef sequence<sequence<IPMPSidContainer,256>, 256> IPMPSidContainerList;

    long useContent(in URL contentSource, in URL contentSink, in Purpose purpose, in
IPMPSid id, in ApplicationServicesListener listener);
    long QueryOVM(in URL contentId, in Purpose p, out IPMPSidContainerList l);
    long getIpmSystem(in IPMPSid id, in URL source, in ApplicationServicesListener
listener );
    long sendMessageToIPMP(in IPMPSid id, in MessageType mt, in Buffer message, in
ApplicationServicesListener listener);
};

/*----- listeners defined in IPMP Services -----*/

interface UI_Listener {
    void receiveMessageFromUser(in long sessionId, in string response); // user
interface
};

interface WE_Listener {
    void newWatermark(in long contentId, in Buffer watermark); // watermark engine
};

interface SC_Listener {
    void cardInserted(in long slotId); // smart card
    void cardRemoved(in long slotId); // smart card
};

interface AAC_Listener {
    long abstractContentAccess( in long contentSourceId, in URL contentSink, in
Purpose purpose, in long sessionId); // abstract access to content
};

interface AAR_Listener {
    void newRules(in long sessionId, in Buffer buffer); // abstract access to
rules
};

interface AAP_Listener {
    long newConnection(in URL peer, in IPMPSid id, in long connectionId);
// abstract access to Peers (SAC)
    long receiveMessageFromPeer(in long connectionId, in Buffer message);
// abstract access to Peers (SAC)
};

interface AAA_Listener {
    void receiveMessageFromApplication(in long sessionId, in MessageType mt, in Buffer
message); // abstract access to application
};

/* this is the only mandatory interface that must be implemented in an IPMPS component */

interface LCC_Listener {
    long initialize(); // life-cycle control
    long terminate(); // life-cycle control
};

/*----- IPMP Services -----*/

interface UserInterface {
    long sendMessageToUser(in string messageText, in UI_Listener listener);
};

interface SecureStorage {
    long secureStoreData (in long dataReference, in Buffer buffer);
    long secureRetrieveData(in long dataReference, out Buffer buffer);
    long secureDeleteData (in long dataReference);
};

interface EncryptionEngine {
    typedef sequence <Buffer, 1024> KeyList;
};

```

```

        List queryEncryptionAlgorithms();
        long encrypt(in string algorithm, in Buffer params, in Buffer key, inout Buffer
data);
        long initEncryption(in string algorithm, in Buffer params, in long
clearContentId);
        long updateEncryptionKeys(in KeyList keys, in long clearContentId);
        long stopEncryption(in long clearContentId);
        long decrypt(in string algorithm, in Buffer params, in Buffer key, inout Buffer
data);
        long initDecryption(in string algorithm, in Buffer params, in long
encryptedContentId);
        long updateDecryptionKeys(in KeyList keys, in long encryptedContentId);
        long stopDecryption(in long encryptedContentId);
    };

    interface SignatureEngine {
        List querySignatureAlgorithms();
        long verifySignature(in string algorithm, in Buffer params, in Buffer publicKey,
in Buffer data, in Buffer signature);
        long verifyContentSignature(in string algorithm, in Buffer params, in Buffer
publicKey, in long contentId, in Buffer signature);
        long generateSignature(in string algorithm, in Buffer params, in Buffer
privateKey, in Buffer data, out Buffer signature);
        long generateContentSignature(in string algorithm, in Buffer params, in Buffer
privateKey, in long contentId, out Buffer signature);
    };

    interface WatermarkEngine {
        List queryWatermarkAlgorithms();
        long extractWatermark(in string algorithm, in Buffer params, in long contentId, in
WE_Listener listener);
        long stopWatermarkExtraction(in long contentId);
        long insertWatermark(in string algorithm, in Buffer params, in long
sourceContentId);
        long stopWatermarkInsertion(in long sourceContentId);
    };

    interface SmartCard {
        typedef sequence <long, 1024> SlotIdList;

        /* these structures declare Application Protocol Data Units as defined in ISO 7816
*/

        struct CommandAPDU {
            octet cla;
            octet ins;
            octet pl;
            octet p2;
            short lc;
            short le;
            sequence <octet, 65535> data;
        };

        struct ResponseAPDU {
            octet sw1;
            octet sw2;
            sequence <octet, 65535> data;
        };

        long addCTListener(in SC_Listener listener);
        long removeCTListener(in SC_Listener listener);
        long getSlotId(out SlotIdList slotId);
        long isCardPresent(in long slotId);
        long openSlotChannel(in long slotId);
        long closeSlotChannel(in long slotSessionId);
        long getATR(in long slotId, in long ms, out Buffer ATR);
        long reset(in long slotSessionId, in long ms, out Buffer ATR);
        long sendAPDU(in long slotSessionId, in CommandAPDU capdu, in long ms, out
ResponseAPDU rapdu);
    };

    interface AbstractAccessToContent {
        long installCallBackContentAccess(in AAC_Listener listener);
        long replyToContentAccess(in long sessionId, in long status, in Buffer message);
    };

```



```

interface AbstractAccessToRules {
    long obtainUserRules(in Buffer userId, in AAR_Listener listener);
    long obtainContentRules(in long sourceContentId, in AAR_Listener listener);
    long updateContentRules(in long sinkContentID, in Buffer buffer);
};

interface AbstractAccessToPeers {
    long openConnection(in URL peer, in IPMPSid id, in long connectionSpecId, in
AAP_Listener listener);
    long closeConnection(in long connectionId);
    long addConnectionListener(in URL peer, in IPMPSid id, in AAP_Listener listener);
    long sendMessage(in long connectionId, in Buffer message);
};

interface AbstractAccessToApplications {
    long installCallBackApplication(in AAA_Listener listener);
    long replyMessage(in long sessionId, in long status, in Buffer message);
};

interface LifeCycleControl {
    void remove();
    void update(in URL source);
};

interface Locale {
    long getTime();
    string getCountry();
    string getLanguage();
};
};

```

## 5 Annexes (Informative)

### 5.1 Acronyms

API	Application Programming Interface
DVB	Digital Video Broadcasting
ID	Identity
IPMP	Intellectual Property Management & Protection
MPEG	Moving Picture Experts Group
OVM	OPIMA Virtual Machine
SAC	Secure Authenticated Channel
TS	Transport Stream (i.e. MPEG-2 Transport Stream)
URL	Uniform Resource Locator

### 5.2 Definitions

Compartment	A class of OPIMA enabled devices that share some common elements in their IPMP interfaces and/or architectural components.
Content	All digital data representing audio, graphics, text, video and associated metadata
IPMP System	The system which protects and manages intellectual property rights associated with content.
Credentials	A set of authenticated identifiers, certifying the compartment ID and the peer ID;
OVM	A group of basic functional elements that implement a secure execution environment for IPMP Systems
OPIMA Peer	An implementation of the OPIMA specification running on a device
Protected Content	Content protected by an IPMP System with associated Rules.
Rules	Statements that govern the way a specific piece of content protected by an IPMP System can be managed