External Functionality Interface Framework

Approved Version 1.1 – 15 Mar 2011

Open Mobile Alliance

OMA-WAP-EFI-V1_1-20110315-A

Continues the Technical Activities
Originated in the WAP Forum

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

Use of this document is subject to all of the terms and conditions of the Use Agreement located at http://www.openmobilealliance.org/UseAgreement.html.

Unless this document is clearly designated as an approved specification, this document is a work in process, is not an approved Open Mobile Alliance™ specification, and is subject to revision or removal without notice.

You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. Information contained in this document may be used, at your sole risk, for any purposes. You may not use this document in any other manner without the prior written permission of the Open Mobile Alliance. The Open Mobile Alliance authorizes you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services. The Open Mobile Alliance assumes no responsibility for errors or omissions in this document.

Each Open Mobile Alliance member has agreed to use reasonable endeavors to inform the Open Mobile Alliance in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. However, the members do not have an obligation to conduct IPR searches. The declared Essential IPR is publicly available to members and non-members of the Open Mobile Alliance and may be found on the "OMA IPR Declarations" list at http://www.openmobilealliance.org/ipr.html. The Open Mobile Alliance has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions. Defined terms above are set forth in the schedule to the Open Mobile Alliance Application Form.

NO REPRESENTATIONS OR WARRANTIES (WHETHER EXPRESS OR IMPLIED) ARE MADE BY THE OPEN MOBILE ALLIANCE OR ANY OPEN MOBILE ALLIANCE MEMBER OR ITS AFFILIATES REGARDING ANY OF THE IPR'S REPRESENTED ON THE "OMA IPR DECLARATIONS" LIST, INCLUDING, BUT NOT LIMITED TO THE ACCURACY, COMPLETENESS, VALIDITY OR RELEVANCE OF THE INFORMATION OR WHETHER OR NOT SUCH RIGHTS ARE ESSENTIAL OR NON-ESSENTIAL.

THE OPEN MOBILE ALLIANCE IS NOT LIABLE FOR AND HEREBY DISCLAIMS ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF DOCUMENTS AND THE INFORMATION CONTAINED IN THE DOCUMENTS.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms set forth above.

Contents

1. SCOPE 5
2. REFERENCES 6
 2.1 NORMATIVE REFERENCES 6
 2.2 INFORMATIVE REFERENCES 6
3. TERMINOLOGY AND CONVENTIONS 7
 3.1 CONVENTIONS 7
 3.2 DEFINITIONS 7
 3.3 ABBREVIATIONS 8
4. INTRODUCTION 9
 4.1 EFI WITHIN THE MOBILE TERMINAL 9
 4.2 EFI REFERENCE ARCHITECTURE 9
  4.2.1 Components 10
  4.2.2 Servers and services 12
  4.2.3 Interfaces 12
  4.2.4 Services 12
  4.2.5 API Considerations 13
 4.3 EXAMPLE 13
  4.3.1 Components of the Example 14
  4.3.2 Walkthrough 14
5. NAMING CONVENTION 16
 5.1 NOTATION 16
  5.1.1 Scheme 16
  5.1.2 Server 16
  5.1.3 Service 17
  5.1.4 Parameters 17
 5.2 EXAMPLES 18
 5.3 NAMESPACES IN THE API 18
 5.4 VENDOR-SPECIFIC NAMESPACE 18
 5.5 RESERVED NAMES 19
6. VERSIONS (INFORMATIVE) 20
 6.1 VERSION HISTORY 21
7. SCRIPT API 22
 7.1 NAMESPACE USAGE 22
 7.2 SERVER ATTRIBUTES 22
 7.3 CLASS PROPERTIES 23
 7.4 SERVICE DISCOVERY 23
 7.5 SERVICE CONTROL 23
  7.5.1 Service control codes 28
  7.5.2 Error and status codes 28
 7.6 WMLSCRIPT API 29
  7.6.1 Containers 29
  7.6.2 Server Attributes 32
  7.6.3 Class Properties 33
  7.6.4 Service Discovery 34
  7.6.5 Service Control 35
 7.7 ECMASCRIPT API 39
  7.7.1 Name/value collections 39
8. MARKUP API 46
 8.1 BEHAVIOUR OF THE MOBILE CLIENT 46
 8.2 SERVERS 47
  8.2.1 EF Broker 47

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

8.2.2 EF Class Agent 47
8.2.3 EF Unit....47
8.3 DISCONTINUOUS MODE 48
8.3.1 Continuation document....48
8.3.2 Return variable....48
8.4 CONTEXT MANAGEMENT 48
8.5 STATUS CODES....49
8.6 UAPROF 50
8.7 CACHE 50
8.8 EXAMPLE....50
APPENDIX A. STATIC CONFORMANCE REQUIREMENTS....52
A.1 SCRIPT ENCODER OPTIONS 52
A.2 CLIENT OPTIONS....52
A.2.1 Broker 52
A.2.2 Scheme....53
A.2.3 APIs 54
A.2.4 WMLScript API....54
A.2.5 ECMAScript API....55
A.2.6 Attributes, Properties 55
A.2.7 Local Server....56
APPENDIX B. CHANGE HISTORY (INFORMATIVE)....57
B.1 APPROVED VERSION HISTORY 57

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

1. Scope

The Wireless Application Protocol (WAP) is a result of continuous work to define an industry-wide specification for developing applications that operate over wireless communication networks. The scope for the Open Mobile Alliance is to define a set of specifications to be used by services and applications. The wireless market is growing very quickly, and reaching new customers and services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation the Open Mobile Alliance defines a set of protocols for the transport, security, transaction, session and application layers. For additional information on the WAP architecture, please refer to "Wireless Application Protocol Architecture Specification" [WAPARCH].

Current trends in telecommunications enable new kinds of functionality in a wireless terminal; either through the integration of new features into the mobile terminal or by allowing new types of devices to be connected to the mobile terminal. Supporting this development in OMA standards will strengthen OMA's position as a platform for advanced wireless data services by providing access to new capabilities.

External Functionality (EF) is a general term for components or entities with embedded applications that execute outside of the Wireless Application Environment (WAE) or other user agent, and conform to the EF requirements. The External Functionality can be built-in or connected to a mobile terminal. This connection can be permanent or temporary.

An application environment of WAP is the place within the terminal where applications are executed, either in the form of markup pages or in the form of scripts or both. The most convenient way to facilitate the connection between the application and new functionality of the terminal is to specify new standard services that can be accessed by an application that is being executed in WAP application environment. The External Functionality Interface supports the notion of classes, conceptual groups of functions that pertain to the same application areas.

The External Functionality Interface (EFI) specifications in WAP provide methods enabling applications to access External Functionality in a uniform way through the EFI Application Interface (EFI AI). The EFI specifications consists of the Framework, the Process specification and a set of Class Specifications, each one specific to the given application area.

EFI Framework defines the general behaviour of EFI implementation in the WAP terminal while detailed requirements for the class are provided in individual Class Specification documents. The Process specification facilitates the development of Class Specifications by defining steps that should be taken in order to achieve the quality Class Specification.

The EFI Application Interface (EFI AI) is a high level interface that shall suit a number of different applications. Various external functions are grouped in classes that offer common functionality across different makes and versions of terminals and external functionality entities. The EFI Framework provides an extensible set of interfaces that can support services, including the ability to query for the particular service as well as the ability to capture the functionality that is specific to the given device or software installed. However, there is no functionality to dynamically add new services so there is no general service discovery mechanism.

This document defines the EFI Framework. The document starts with the requirements and principles that the EFI Framework is built upon. Next, the conceptual architecture of EFI is presented and the terminology is introduced. The definition of the Framework follows, addressing issues such as naming convention and version control. The following chapters show how the Framework can be accessed via a markup language (WML and XHTML Mobile Profile) and scripting language (WMLScript or ECMAScript). The Framework requires the mobile client to support both the scripting and markup languages specified by the Wireless Application Environment [WAE].

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

2. References

2.1 Normative References

[IOPProc] "OMA Interoperability Policy and Process". Open Mobile Alliance™. OMA-IOP-Process-v1_0. URL:http//www.openmobilealliance.org/

[RFC822] "Standard for the Format of ARPA Internet Text Messages". Crocker, D. August 1982. http://www.ietf.org/rfc/rfc822.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. November 1997. URL:http://www.ietf.org/rfc/rfc2234.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax". T. Berners-Lee, R. Fielding, U.C. Irvine, L. Masinter. August 1998.http://www.ietf.org/rfc/rfc2396.txt

[ESMP] "ECMAScript Mobile Profile", Open Mobile Alliance™. OMA-WAP-ESMP-V1_0. http://www.openmobilealliance.org/

[UAProf] "User Agent Profile Specification", WAP Forum™, WAP-248-UAProf. http://www.openmobilealliance.org/

[XHTMLMP] "XHTML Mobile Profile 1.1", Open Mobile Alliance™. OMA-WAP-XHTMLMP-V1_1.http://www.openmobilealliance.org/

[WAE] "Wireless Application Environment Version 2.1", Open Mobile Alliance™. OMA-WAP-WAESpec-V2_1http://www.openmobilealliance.org/

[WML1] "Wireless Markup Language, Version 1.3", WAP Forum™, WAP-191-WML. http://www.openmobilealliance.org/

[WML2] "Wireless Markup Language, Version 2.0", WAP Forum™, WAP-238-WML. http://www.openmobilealliance.org/

[WMLScript] "WMLScript Language Specification", WAP Forum™, WAP-193-WMLScript, http://www.openmobilealliance.org/

2.2 Informative References

[WAPARCH] "WAP Architecture". WAP Forum™. WAP-210-WAPArch. http://www.openmobilealliance.org/

[CACHE] "WAP Caching Model Specification", WAP Forum™, WAP-175-CacheOp. http://www.openmobilealliance.org/

[RFC2616] "RFC2616: Hypertext Transfer Protocol -- HTTP/1.1". R. Fielding, et al. June 1999. http://www.ietf.org/rfc/rfc2616.txt

[WMLLib] "WMLScript Standard Libraries Specification", WAP Forum™, WAP-194-WMLScriptLibs. http://www.openmobilealliance.org/

[WSP] "Wireless Session Protocol", WAP Forum™, WAP-230-WSP. http://www.openmobilealliance.org/

[WTA] "Wireless Telephony Application Specification", WAP Forum™, WAP-266-WTA. http://www.openmobilealliance.org/

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

Application	The executable or interpretable code that is running within the application environment (such as WAE); an application may use various APIs to access EFI services.
Broker	The conceptual entity that exists between the EF Units, EF Class Agents and the EFI AI. The EF Broker maintains the list of available functionality and routes requests to the correct EF Unit or EF Class Agent or handles them itself.
Class	The collection of all EF Units and EF Class Agents that share the same functionality according to the same Class Specification.
Class Agent	The conceptual active element that provides added functionality on the basis of EF Units of the same EF Class Realisation.
Class Realisation	The collection of EF Units and optionally the EF Class Agent that belong to the same EF Class and are available to a particular Terminal.
Class Specification	The definition of services that are provided by every EF Unit that belongs to the given class and services provided by the EF Class Agent.
Entity	The conceptual component that expresses the EFI view on a software or hardware component of the mobile terminal that exposes some of its function for the purpose of EFI.
Gateway	WAP gateway as specified in [WAPArch].
Implementation	The software and hardware that is used in the particular terminal to implement the functionality.
Mobile Terminal	The physical unit where the WAE executes.
Origin Server	The server on which a given resource resides or is to be created. Often referred to as a web server or an HTTP server.
Registry	The conceptual place where information about available EF Units and EF Class Agents is stored and then made accessible by the EF Broker.
Server	Any of the components of the EFI conceptual architecture that can be addressed to provide the service for an application; a collective name for the EF Broker, EF Units and EF Class Agents.
Service	The specified functionality provided by one of the servers: EF Broker, EF Class Agent or EF Unit.
Unit	The conceptual component that resides in or outside the mobile terminal and provides access to the EF Services on the EF Entities.
XHTML Mobile Profile	A language which extends the syntax of XHTML Basic as specified in [XHTMLMP].

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

3.3 Abbreviations

AI	Application Interface
API	Application Programming Interface
EF	External Functionality
EFI	External Functionality Interface
EFE	External Functionality Entity
ESMP	ECMAScript Mobile Profile [ESMP]
OMA	Open Mobile Alliance
SIM	Subscriber Identity Module
UAProf	User Agent Profiling
XHTML	eXtensible HyperText Markup Language
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WINA	WAP Interim Naming Authority
WML	Wireless Markup Language; refers collectively to WML Version 1.3 [WML1] and WML Version 2.0 [WML2]
WTA	Wireless Telephony Application [WTA]

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

4. Introduction

The conceptual architecture of EFI consists of several blocks that are collectively called 'EFI'. This chapter defines how the 'EFI' relates to other components that may reside in the mobile terminal from the perspective of application execution environment and how the EFI component can be further conceptually structured.

4.1 EFI within the mobile terminal

An application environment within the mobile terminal consists of several components of which EFI is one. The simplified relationship of those components is depicted below.

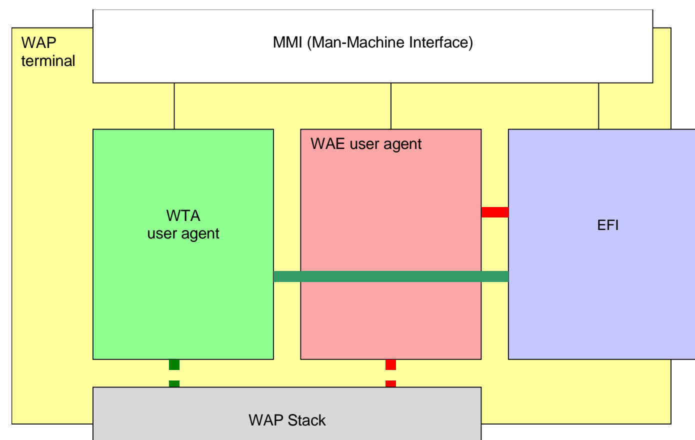

Figure 1. EFI Architectural Overview

EFI is positioned as the terminal component that interacts with WAE user agent [WAE], similar to WTA. The interface provided by EFI allows both WAE applications and WTA applications to call the external functionality through EFI.

The functionality that is accessible through EFI may require certain resources from the mobile terminal, like memory space, processor time or dedicated hardware. It is assumed that the implementation of EFI provides means that allow the proper resource sharing within the mobile terminal.

EFI has the same rights to use man-machine interface (MMI) and communication capabilities as WTA or WAE. However, the EFI Framework does not provide any formal description of how EFI is using those capabilities, leaving it to the implementation of particular components that constitute EFI.

The primary purpose of EFI is to provide access to external functionality. The functionality is considered 'external' when it is not the standard functionality of WAE or WTA. Whether such functionality resides inside or outside of the terminal or whether the functionality is permanently or only temporarily available is irrelevant.

4.2 EFI Reference Architecture

The framework defines the conceptual reference architecture, as depicted below. The intent of this architecture is to provide consistent terminology and understanding. The reference architecture does not imply or endorse any particular implementation but demonstrates the structure of EFI as perceived by the application developer.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

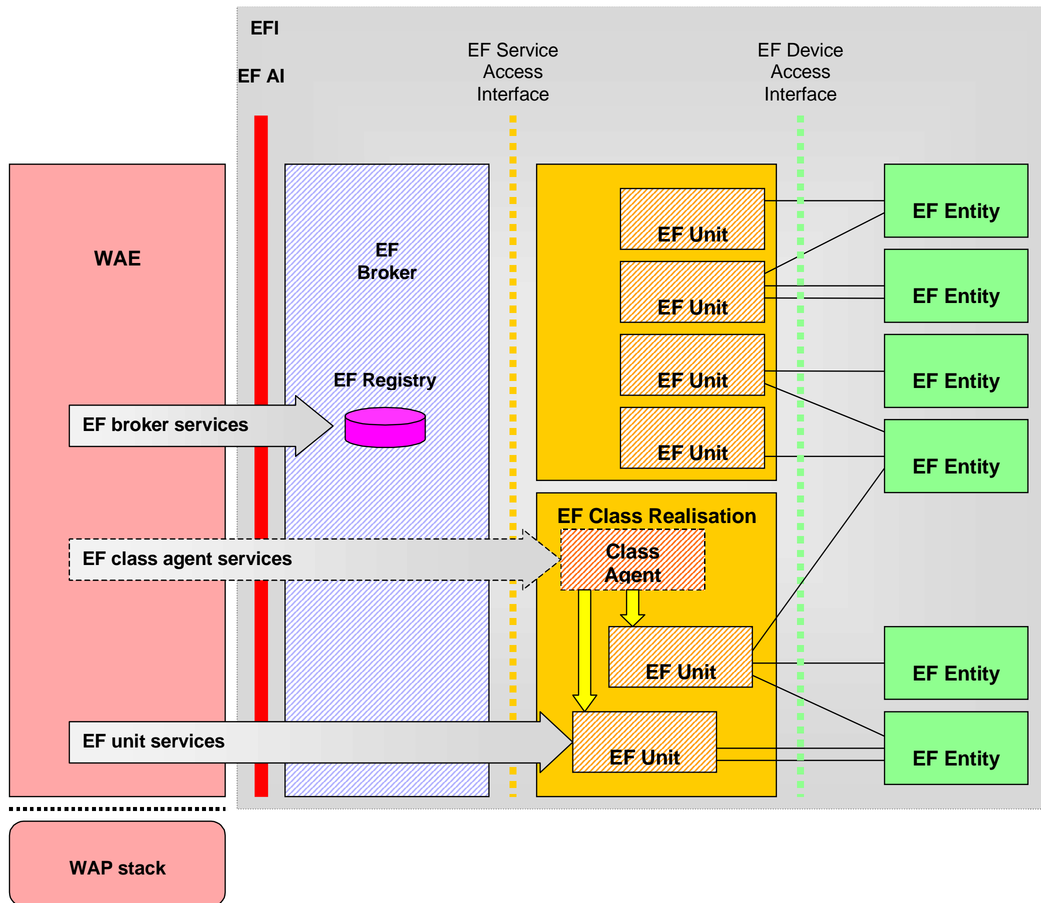

Figure 2. EFI conceptual architecture

4.2.1 Components

The conceptual architecture of EFI identifies several components. The following description starts from the rightmost (most detailed) components shown on the picture.

This architectural view of EFI highlights components and the static relationship between them. The dynamic architecture of servers and services is presented in the next chapter.

4.2.1.1 EF Entity

The EF Entity (EFE) is a component that implements a functionality. The EFE can be internal or external to the mobile terminal or even a set of software modules that provide a functionality that should be made available to applications in WAE. The EF Entity can be functionally larger than what is seen through the EFI Interfaces with only part of the overall EFE functionality exposed within EFI.

The term 'external' need not be interpreted in a physical sense. The EF Entity may physically reside on the mobile terminal. 'External' means that the EF Entity is not specified as part of the wireless application environment. Specifically, only a subset of the actual device functionality can be published through EFI. The EF Entities are allowed, and expected to be proprietary implementations.

The EF Entity that represents the functionality that spans several EF classes may be shared by EF Units from different Class Realisations. Within each Class Realisation the EFE exposes the part of its functionality that is specific to the class. For example, the Global Positioning System (GPS) clock can be used by the 'positioning' class as well as by the 'calendar' class.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

Examples of EF Entities include GPS units, thermometers, video recorders, cameras, measuring units, navigation systems, SIM Toolkit Applications, Smart Card Readers etc.

4.2.1.2 EF Unit

An EF Unit is an abstraction of EF Entity functionality expressed in a way that is compatible with EFI. The EF Unit is the component that actually provides services to the application, so that it acts as a server for application. The sets of services that are reachable through the EF Unit are mapped to functionality available from EF Entities in an implementation-specific way.

Note that one EF Unit may build its functionality on one or more EF Entity components. Also one EF Entity may deliver its functionality to more that one EF Unit, whether they are from the same class or from different classes.

The Class Specification defines services that the EF Unit must provide to the application. All EF Units that belong to the same Class must provide the mandatory class services. A simple example of a Class could be digital camera where the minimal set of EF Services could consist of takePictures.

Each EF Unit belongs to exactly one EF Class Realisation. Each EF Unit has a unique identifier within the Terminal. The unit identifier is defined as a valid name not duplicated by any other unit or EF Class Realisation. The EF Broker can dynamically create the identifier.

4.2.1.3 EF Class Realisation

The EF Class Realisation houses within the terminal EF Units that fulfils the same Class Specification regardless of the actual version and of the implementation. The EF Class Realisation may also contain the EF Class Agent. Within the terminal there is no more than one Class Realisation of each Class. The name of the Class Realisation is always identical with the name of the Class it realises.

An EF Unit always belongs to an EF Class Realisation. If the particular EF Unit does not fall into any specific Class as defined by the Open Mobile Alliance, it is defined as a part of a vendor-specific EFI Class Realisation.

The EF Class Realisation identifies one of its EF Units as the default EF Unit that is used when an application does not specify any unit. For a WAP application to use class services at least one EF Unit of the Class Realisation must exist in the mobile terminal.

EF Units may be added, without any standardisation, whether they provide standardised services or add vendor-specific capabilities. Addition of new classes will always require some standardisation effort. Specifically, the name of the Class and its required services are defined by the specification.

4.2.1.4 EF Class Agent

The EF Class Agent is the active component of the EF Class Realisation that aggregates functions of particular units within the class or manages units in a class specific manner. The EF Class Agent is an EF Server as it provides its services to the application. The EF Class Agent provides services that allow applications to access functionality that is specific to the Class, yet beyond the scope of the single EF Unit, e.g. multi-criteria unit selection.

The EF Class Agent is optional in the sense that the Class Specification determines whether the Class Agent is required for the given class and what services it should perform.

4.2.1.5 EF Broker and EF Registry

The role of the EF Broker is to collect information about available EF Class Realisations, EF Class Agents, EF Units and EF Services in the EF Registry and subsequently to route service requests to the appropriate servers. Note that EF Broker is also able to handle some of the services by itself, acting as an EF Server.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

4.2.2 Servers and services

The main purpose of EFI is to provide services to applications. For that purpose, EFI can be perceived as a collection of servers that provide services. This view of the EFI architecture overlaps with the static architectural view presented in the previous chapter in that some components of the static architecture function as servers. All EF Servers are marked with diagonal lines on Fig.2.

4.2.2.1 EF Server

The EF Server is any component of the EFI conceptual architecture able to provide services to an application. The term 'server' is used as a collective name for EF Broker, EF Units and EF Class Agents.

4.2.2.2 EF Service

The EF Service is the EFI-related functionality that is available to the application through the EF-AI interface. EF Services are provided by three different components of EFI.

The EF Unit provides applications a set of EF Services that are built from functionality delivered by EF Entities. The EF Class Specification defines the mandatory and optional services for the class. The EF Unit may provide more services than defined for its Class, but must provide all services that are defined as mandatory for the Class.

The EF Class Agent provides services that allow applications to access functionality that is specific to the Class, yet beyond the scope of the single EF Unit, e.g. multi-criteria unit selection. Each EF Class Agent can define its own services as long as they fulfil obligations expressed in the EFI Framework.

The EF Broker provides EF Services that allow the application to discover server and services available within the given implementation of EFI.

EFI Framework does not provide any security measures that may be available for services. EFI Framework does not define any security framework that can be used by services. Services with exceptional security requirements may implement the appropriate security mechanisms outside of the EFI Framework.

4.2.3 Interfaces

The conceptual model of the EFI architecture defines several interfaces. Only the EFI-AI is required to be implemented. None of the remaining interfaces are required for any implementation. Only the EFI-AI interface falls into the scope of OMA. Other interfaces are used throughout this document to illustrate concepts and to establish the common terminology.

4.2.3.1 EF AI

The EF AI is the Application Interface to the EF Services offered by all EF Servers. In addition, the EF AI may provide access to other functionality necessary to support EF Services. This is the only interface specified by OMA.

4.2.3.2 Other interfaces

The EF Service Access interface defines EFI Broker interaction with the EF Units, and is outside the scope of OMA.

The EF Device Access interface exists between the EF Unit and the EF Entity. This interface is outside the scope of OMA.

4.2.4 Services

The EFI Framework makes distinctions between groups of services that are provided through EF AI. The purpose of those groups of services is defined below and then summarised in the following table.

- Broker services are provided by the EF Broker for EF Registry access. Such services are used to discover existing components of EFI, including class agents, units and services. EFI Framework defines all management services.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

- Class Agent services may provide unit management within the class realisation and/or value-added services on top of services already defined by units. A class specification may contain both mandatory and optional services provided by the Class Agent. Mandatory Class Agent services must be implemented by the Class Agent. Optional Class Agent services may be implemented by the Class Agent, but the Class Agent must not use the name of the optional service for any other purpose.
- Unit services are the set of services implemented to fulfil the class specification. A class specification may contain both mandatory and optional services. Mandatory unit services must be implemented by every unit of a given class. Optional unit services may be implemented by the unit, but the unit must not use the name of the optional service for any other purpose. The definition of unit services is provided by the class specification.
- Proprietary services allow access to specific functionality within any server beyond standardised services. EFI Framework defines the method to identify and access such services but makes no other provision for services define outside of OMA. The definition of such services is at the discretion of the implementor of the particular EF server.

The following table summarises the grouping of services.

Name	Provided by	Defined by
Management	EF Broker	Framework
Class-specific	EF Class Agent	Class specification
Unit	EF Unit	Class specification
Proprietary	any EF Server	outside OMA

Table 1. Groups of services

4.2.5 API Considerations

Services that are available from EFI must be accessible through different API’s to accommodate different application requirements. Specifically, EFI services are accessible through the Script API and Markup API. Calls to the service through any API are conceptually routed to EFI Broker implementation that may in turn route them to an EF Unit, EF Class Agent or handle them internally.

If an application has access to more than one API (e.g. has access to both the Markup and Script APIs), it can freely mix interaction through all the available interfaces but it should be aware of the possible interaction of services.

4.3 Example

The following example is intended to illustrate the relationship between EF Classes, Units, Entities, WAE, the Broker and other components of the EFI Framework. It is NOT intended to be a complete example of an implementation, nor to suggest that the illustration would be the appropriate way to implement the example. Use of the classes below does NOT imply a commitment from OMA to specify the classes.

In overview, the example attaches a digital camera to a mobile terminal. The camera can capture images, and has the capability to display digital images on its display. The mobile device also has a display, less capable than that of the camera. The WAP application on the mobile device is capable of retrieving images from the camera and transferring them to other storage, probably network based. A display class on the mobile device has knowledge of the camera and mobile device displays, and contains an agent capable of selecting the appropriate display for a particular image. Whether this occurs with or without the intervention of the device user is not germane to the example. Depending on the implementation, either or both methods could be used.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

4.3.1 Components of the Example

The components of the example illustrated below are:

- A digital camera, with a communications port and a colour display of sufficient quality and size to accurately render a photographic image. The camera is capable of producing, processing and rendering digital images.
- A mobile communication device, with a communications port matching the camera's port, and a display. For the purposes of the example, we'll assume that the display on the mobile device is less capable than the camera display (e.g. a monochrome, Liquid Crystal Display 6-line display with limited graphics).

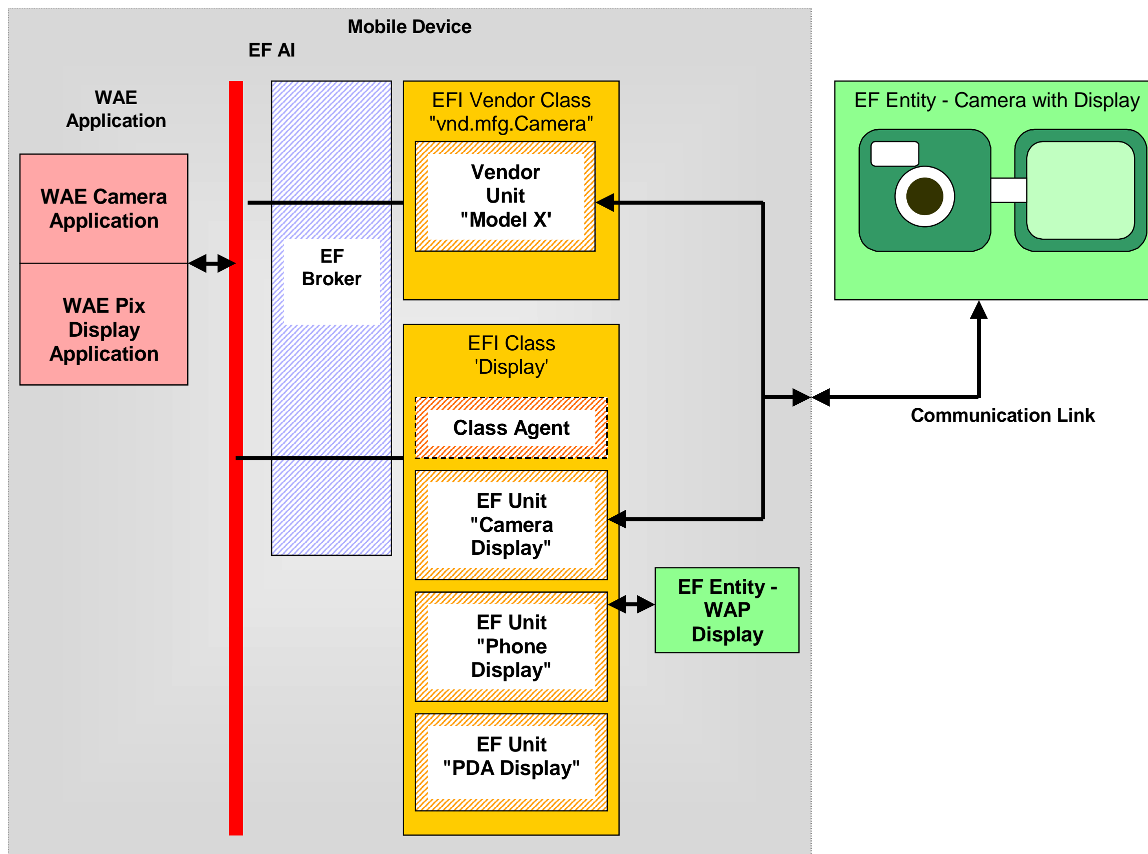

Figure 3. Example of EFI conceptual architecture

4.3.2 Walkthrough

The example decomposes into two applications that access EFI services through the EFI AI and broker. Application (1) is a manufacturer's camera application, running in the Wireless Application Environment (WAE). Application (2) is an image display application for rendering images retrieved from the web or other sources. The camera is assumed to be in communication with the mobile device, the means is not significant to the example, but could be serial, USB (Universal Serial Bus), IrDA (Infra-red Device Association protocol) or other type of connection.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

Moving from left to right in the example, the applications communicate with the EF Broker, which exposes the services available, and routs the calls to the appropriate servers. The EF Classes instantiated on the mobile device are 1) The Display Class which houses units controlling different types of display capabilities, and 2) the Camera Class, providing access to a manufacturer's camera, connected to the mobile device. The Display Class includes 3 units: a manufacturer's Camera Display unit, a mobile device display unit and a PDA (Personal Digital Assistant) Display unit. The PDA Display unit is not in use. In addition, the Display Class implements a Class Agent. The Class Agent in this case has knowledge of the relative capabilities of the two attached displays and can rout information to the appropriate display. The agent may choose to route an image from the PIX app to the camera where the appropriate image handling is available, instead of the device display where the image cannot be correctly processed.

The Camera Class instantiates one unit, communicating with the camera device. There are only two EF Entities represented, the mobile device Display and the Camera. Two EF Units access the Camera, the Camera Unit and the Camera Display Unit.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

5. Naming convention

EFI makes intensive use of the concept of 'namespace', the structured space of components that can be identified by their names. The EFI namespace extends the notion of server/service architecture as it is used to identify services that are provided by various EFI servers.

5.1 Notation

The namespace used by EFI consists of four segments, of which two can be built into the hierarchical structure. The notation used to express the namespace is compliant with [RFC2396]. As specified in [RFC2396], the grammar is that of [RFC822], except that "|" is used to designate alternatives. The necessary escape convention is also drawn from [RFC2396].

[ ]	Square brackets denote an optional section.
|	Vertical bar denotes alternatives
( )	Brackets are used to group elements
*	Star denotes that the next element can repeat none or multiple times
" "	Ampersands are used to denote terminal literal

The namespace that is fully defined by the EFI Framework contains of up to four segments, as shown below. There is a compulsory separator between the Scheme and the Server as well as between the Server and Service and between Service and Parameters (if parameters are present at all)

```
[ Scheme ] "://" Server "/" [Service] ["?" Parameters]
```

The hierarchical structure of the name of both Server and Service segments does not imply the hierarchical structure of classes, units or services. The hierarchy of names is used only to encourage logical grouping of names of servers and services.

5.1.1 Scheme

The scheme is the fixed component of each name that belongs to the EFI namespace. The scheme element identifies the name as belonging to the EFI namespace. On some APIs if it does not lead to ambiguity, the scheme component can be left empty. If used, the scheme is always denoted as 'efi'.

The scheme component MUST be case-insensitive, i.e. efi, Efi or EFI can be used.

```
Scheme =      "efi"
```

5.1.2 Server

The server part identifies the component of EFI conceptual architecture that provides the service. Following are the possible name structures that can be used to identify the server.

```
Server =      Broker |
       Def-Unit-Spec-Class |
       Classagent-Spec-Class |
       Def-Unit-Vnd-Class |
       Classagent-Vnd-Class |
       Identified-Unit
Broker = ""
Def-Unit-Spec-Class = Class-Name
Classagent-Spec-Class = Class-Name ".agent"
Def-Unit-Vnd-Class = "vnd." Class-Name
Classagent-Vnd-Class = "vnd." Class-Name ".agent"
Identified-Unit = "." Unit-Name
Class-Name = Segment * ("." Segment)
Unit-Name = Segment
Segment = alphanum * alphanum        MUST NOT be one of the reserved names
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

```
alphanum                              as in [RFC2396]
```

The server name MUST be case-insensitive, regardless of the type of the server name in use (class name, unit name or agent name).

The empty name of the server identifies EF Broker. When the empty name is used, both slash separators (before and after the server) are in place, resulting in three subsequent slashes.

When the default unit of the class realisation is addressed (Def-Unit-Spec-Class), the server requires only the name of the class. The name of the class may have one or more segments linked by the dot '.' where each segment is identified as a string of letters and digits. The suggested hierarchical structure of the server name does not imply any hierarchical structure of classes; the dot is used only to encourage better structure of class names.

All class names are to be registered with WINA (or an equivalent). Segments 'vnd' and 'agent' MUST NOT be used.

When the Class Agent is addressed as a server (Classagent-Spec-Class), the segment '.agent' is appended to the name of the class.

All names that start from vnd. are assigned for classes that are defined by vendors (Def-Unit-Vnd-Class and Classagent-Vnd-Class). A vendor is required to append its unique name to the 'vnd.' segment of the name. The class name specific to the vendor is be registered by WINA in the same manner as the class name defined by the Open Mobile Alliance.

When the unit is addressed as a server (Identified-Unit), its unique identifier must be provided. This identifier can be retrieved by EF Broker services. The identifier is not the name of the unit and can be assigned dynamically by the Broker.

Unit identifiers MUST start with the dot '.' character before the only segment. Note that any segment of the unit identifier MUST NOT be identical with reserved names.

Note that there is no method to address the class realisation itself, as the class realisation provides no services. In some cases (e.g. attributes, versions) the Broker can be called for information about the class realisation.

5.1.3 Service

The service component identifies the service that is provided by the server. Services bear names that are unique within the server.

```
Service = Service-Name
Service-Name = Segment *( ("/" Segment) | ("." Segment))
Segment = alphanum * alphanum
alphanum            as in [RFC2396]
```

The service name, as seen by the application, MUST be case-sensitive.

The name of the service may have one or more segments linked by the slash '/' or dot '.'. The visible hierarchical structure of the name does not imply any hierarchical structure of services, the slash and the dot characters are used only to encourage better structuring of service names.

The service component is optional and can be left empty. Such notation identifies the 'no-name' service.

5.1.4 Parameters

Parameters can be passed to the service by the namespace or by alternative methods defined by APIs. If passed by the namespace, parameters MUST take the form of named values as defined for the query component of an URI [RFC2396].

```
Parameters = param "=" value *("&" param "=" value)
param = alphanum * alphanum
value = * Char
Char = unreserved | escaped
alphanum            as in [RFC2396]
unreserved          as in [RFC2396]
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

```
escaped            as in [RFC2396]
```

NOTE: In consequence that the ampersand is being a reserved character for the query component, it must be escaped as such.

Names of parameters (Param) MUST be case-sensitive. Values of parameters (Value) MUST be case-sensitive.

5.2 Examples

Following are a few examples of valid names as they might appear in the context of EFI.

Name	Identifies
`efi:///foo`	Service 'foo' of EF Broker, no parameters
`EFI://wallet/select?name=purse`	Service 'select' of the default unit of class realisation 'wallet' with one parameter 'name' equal to 'purse'
`Efi://.U1234/books/register`	Service 'books/register' of the unit 'u1234'

Table 2. Examples of the namespace usage

5.3 Namespaces in the API

The namespace is an abstract concept that may be used differently by different APIs. Specifically, the Markup and Script APIs may make use of the namespace in different ways.

The mapping between the concept of the namespace and the access provided by the particular API is provided together with other aspects of API implementation.

5.4 Vendor-specific namespace

EFI Framework provides part of the namespace for classes that are specific to vendors. Vendor specific namespace allows vendors to experiment and introduce functionality that is not standardised within OMA but that can be accessed through EFI mechanisms.

EFI reserves the name 'vnd.' as the first segment of the class name to identify the branch of the namespace that belongs to vendors. The vendor has to use its registered namespace as the subsequent segments. The remaining part of the vendor-specific class name is left to the vendor with the exception that the vendor is not allowed to use reserved names for any segment of its class name.

Examples:

The vendor registers the name acme.boo with WINA. The following names are valid

reference to service request.it in the default unit of the class vnd.acme.boo.dowhateveryouwant

efi://vnd.acme.boo.dowhateveryouwant/request.it

reference to service seek/result in the class agent of the class vnd.acme.boo

efi://vnd.acme.boo.agent/seek/result

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

5.5 Reserved names

EFI reserves the following names in all variants of upper and lower cases. Regardless of their intended use, reserved names MUST NOT be used as any segment of the class name. Names that conflict with the reserved ones should not be registered and used.

vnd Reserved as a prefix for all the vendor-specific classes.

The valid name of the vendor-specific class is defined by one of the following terms:

Def-Unit-Vnd-Class

Classagent-Vnd-Class

agent Reserved as a suffix for class agents.

The valid name of the class agent is defined by one of the following terms:

Classagent-Spec-Class

Classagent-Vnd-Class

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

6. Versions (Informative)

EFI components may be distributed (e.g. by means of by accessories or by the ad-hoc network). EFI is designed to provide functionality to applications using components of different versions.

EFI recognises two sources of versions: the Framework and Class Specifications.

The Framework defines the version of the EF Broker. The Class Specification defines the version reported by the compatible Class Agent. The Class Specification defines versions of Units, i.e. each Unit reports the version of the Class Specification it is compatible with. Services are not distinguished by version and do not have separate version numbers.

In case of vendor-specific classes the vendor can implement an arbitrary version numbering scheme. Vendor-specific versioning is correctly recognised by EFI only to the extent that is compatible with rules defined by EFI Framework.

EFI defines the following rules for version control:

- The specification of the Framework defines the version of the EF Broker.
- The version of the Framework is defined by its major version number and the minor version number. If a new specification of the Framework is not backward compatible with the current one, the major number defined by the new specification is increased. If the new specification is backward compatible, the major number should be left unchanged and the minor number should be increased.
- The first specification of the Framework defines Framework version 1.0. A new Framework has a higher version than the old Framework. See section 6.1 below for the version history.
- Guidelines regarding backward compatibility are defined as follows:
    - The conceptual architecture of EFI Framework contains the elements and relationships defined in the previous version.
    - The namespace of the new version does not conflict with the old namespace.
    - All mandatory features from the old SCR are preserved.
    - Optional features from the old SCR may be removed, left as optional or become mandatory.
- The Class Specification defines the version that is reported by the compatible Class Agent and all the Units that are compatible with the Class Specification.
- The version of the Class Specification is defined by its major version number and the minor version number. Change in the major version number is used to identify significant changes to the Class Specification. A change in the minor version number identifies less important changes to the Class Specification.
- The first specification of the given Class defines version 1.0.
- A new Class Specification has a higher version than the previous (old) Class Specification.
- If the new Class Specification is not backward compatible with the current specification, the major version number of the new specification is increased. If the new specification is backward compatible, the major number should be left unchanged and the minor number should be increased.
- Guidelines regarding backward compatibility are defined as follows:
    - All mandatory services of Class Agent or Unit are preserved.
    - Optional services of Class Agent or Unit may be removed, left as optional or become mandatory.
    - All mandatory parameters of the service are preserved.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

  - Optional parameters of the service may be removed or left as optional; if removed, the new version of the service should not report an error when former optional parameters are used.
  - The default behaviour and default values are preserved.
- If a service is being removed from the Class Specification, it is recommended to identify it as 'deprecated' and then remove it in the next version of the Class Specification.
- It is possible that the Class Realisation controls Units and the Class Agent of different versions. If the implementation allows for such case, it is responsible for the correct handling of all the Units and the Class Agent. Specifically
  - All services from all Units, as specified by respective Class Specifications are available to the application regardless of the version of the Unit.
  - Versions of all the Units and the version of the Class Agent are correctly reported.
  - If the Unit has a version lower (older) than the Class Agent in the same Class Realisation, new services defined by the Class Specification for the Class Agent (comparing to services of the Class Agent from the Class Specification of the Unit) may not be available with regard to this Unit.
  - If the Unit has a version higher (newer) than the Class Agent in the same Class Realisation, services other than defined by the Class Specification of the Class Agent may not be available with regard to this Unit.
- An application has an access to the following information:
  - The version of the EF Broker which is the version of the compatible Framework
  - The version of each EF Unit and EF Class Agent which are versions of the respective compatible Class Specifications
  - The lowest and highest versions of EF Units (not including EF Class Agent) within the EF Class Realisation.

In addition to the above, each server may also provide a manufacturer version that is different from any version defined above. If the server provides a manufacturer version, EFI makes it available to applications. EFI does not assume any particular numbering scheme or compatibility requirements for manufacturer versions.

6.1 Version History

The version history for this Framework specification is as follows. The version history for each Class will be tracked in the Class specification itself.

Version	Date	Description
1.0	17-December-2001	The initial version of the Framework
1.1	01-October-2002	Added ECMAScript API to the Framework

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

7. Script API

To use the Script API, the mobile client MUST support either ECMAScript Mobile Profile or WMLScript, and MAY support both, as specified by [WAE]. The EFI AI is implemented as a single script library in WMLScript, or as an ECMAScript Object. All services are accessed either through function calls or by passing the name of a server or service along with a command to one of the functions.

A mobile client that implements EFI MUST support the EFI Script API in the scripting language the mobile client supports. If the mobile client supports both ECMAScript and WMLScript, the mobile client MUST support the EFI ECMAScript API and the EFI WMLScript API. The mobile client SHOULD use User Agent Profile [UAProf] to advertise that the mobile client implements EFI.

If any broker function interacts with the user, the user MUST be informed that the interaction is with the EFI implementation.

The following sections provide an overview of the functionality that is realized by the WMLScript and ECMAScript APIs. Sections 7.6 and 7.7 provide the respective APIs for WMLScript and ECMAScript.

7.1 Namespace usage

The Script API makes use of the namespace in the following manner:

- The scheme segment is 'EFI'. If the name is used to identify the object within the namespace (i.e. the server or the service), the 'efi:' component MUST be omitted from the name.
- Broker services that are defined by the Framework and are accessible through the Script API are expressed as function calls. The application has no other means to access such services. This does not preclude the implementation of the Broker from providing other services (not specified by the Framework), which can be accessed through designated functions.

Services that are defined for the class agent or for the unit are accessible through generic service calling functions like invoke or control.

7.2 Server Attributes

Each EFI server has a set of attributes that can be extracted by the application. EFI mandates which attributes are to be present for all servers of a given type. Any server can provide more attributes than mandated. EFI SHOULD pass those attributes to the application. Names of attributes MUST be case-sensitive.

The Framework defines the following attributes. Attributes marked with 'M' are mandatory. Attributes marked with 'O' are optional.

Server	Attribute	Description	Status
Broker	VersionMajor	Major version number of the Broker, i.e. major version number of the compatible Framework	M
	VersionMinor	Minor version number of the Broker, i.e. minor version number of the compatible Framework	M
	Manufacturer	Manufacturer of the Broker, may include the make and the model	O
	ManVersionMajor	Manufacturer major version of the Broker	O
	ManVersionMinor	Manufacturer minor version of the Broker	O
Unit or Class Agent	VersionMajor	Major version number of the Unit or the Class Agent.	M
	VersionMinor	Minor version number of the Unit or the Class Agent.	M

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

	Name	Descriptive name of the Unit	M
	Manufacturer	Descriptive name of the manufacturer of the Unit, may include the make and the model	M
	ManVersionMajor	Manufacturer major version of the Unit or Class Agent	O
	ManVersionMinor	Manufacturer minor version of the Unit or Class Agent	O

7.3 Class Properties

The Broker can provide properties of class realisations. Such properties can be retrieved by the application whenever necessary. EFI mandates which properties are to be present for the class. The Broker can provide more properties than mandated. Names of properties MUST be case-sensitive.

Following properties MUST be available:

Property	Description
MinVersionMajor	Major part of the lowest version of the Unit that is available through the Class Realisation. Note that only Units that are visible for service discovery functions are used to calculate this property.
MinVersionMinor	Minor part of the lowest version of the Unit that is available through the Class. Note that only Units that are visible for service discovery functions are used to calculate this property.
MaxVersionMajor	Major part of the highest version of the Unit that is available through the Class Realisation. Note that only Units that are visible for service discovery functions are used to calculate this property.
MaxVersionMinor	Minor part of the highest version of the Unit that is available through the Class Realisation. Note that only Units that are visible for service discovery functions are used to calculate this property.

7.4 Service Discovery

EFI makes it possible to query about the existence of servers and services, a capability called 'service discovery'. An application is able to identify all the servers within the given Class Realisation (including Class Agent, if required). The application can also query for the given server if the server identifier is known to the application.

The broker MUST allow servers to elect whether they are visible through service discovery functions or not. The implementation also allows servers to decide whether some or all services are visible through service discovery functions. If the server or the service is not visible, it is not reported by corresponding service discovery functions.

The visibility of the server or the service is relevant only to service discovery functions and does not impact the ability of the service to be started by service control functions.

The class realisation is not reported as existing by the service discovery functions if it has no server that reports its presence through service discovery functions. Note that the server is either the unit or the class agent. A class realisation where all units are hidden but the class agent is visible can be created.

7.5 Service Control

The EFI provides access to the set of services that are available from units. When the service is executed, it is instantiated in the same sense that an executed program is the instance of a program's code.

The instance of the service is started by an application in order to execute external functionality. An application provides parameters to the instance and receives results from the instance.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

All instances are perceived by an application as asynchronous, i.e. they are invoked by the application and then they continue in parallel. An application may start several instances of the same or of different services. EFI neither mandates nor precludes the possibility of several instances of the same service being executed at the same time. EFI provides means for the instance to notify the application when it cannot execute due to any conflicts.

The instance (as opposed to the call to the instance) may be interruptible by events that are coming from user agents (e.g. WTA events) or from other applications, including non-WAP applications.

When the service instance is invoked, EFI analyses whether the instance can be started at the given time. In some cases the lack of resources, implementation of the service or the unavailability of some components may prevent EFI from starting the instance. In this case an application receives an error code. The application is responsible for the correct handling of error codes defined in the Framework.

Every instance, if started, MUST return its instance identifier that MUST be an unique non-negative integer. The instance identifier can be used by an application to communicate with the instance of the service. The instance identifier has meaning only within the scope of the user agent that has invoked the service.

EFI assumes a simple state model of the instance, as depicted below. The instance is always in one of the three states and may move to another state as the result of its own action or as a response to external event, including a call from the application.

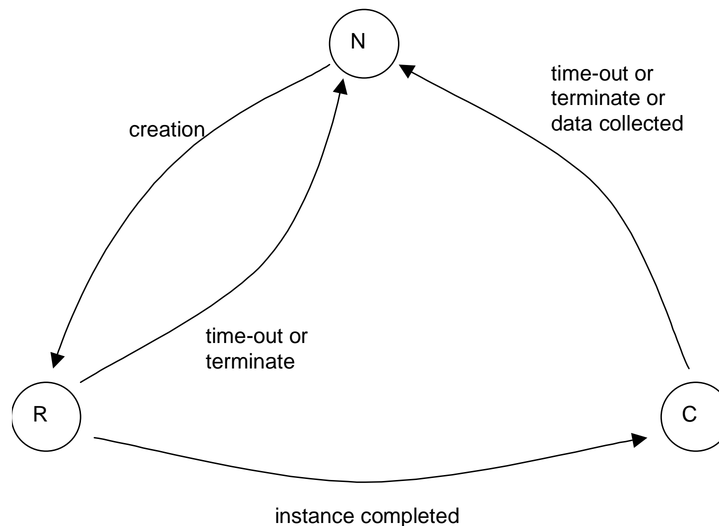

Figure 4. State model of the instance

The following table provides the detailed description of both states and transitions. Note that only transitions between states are included on both the diagram and the table. The instance may respond to certain functions or perform actions while it remains at the same state.

State	Name	Meaning	Transition to	As result of
N	non-existing	Denotes the state in which the instance is inactive and does not hold any resources	R	creation of the instance
R	running	The instance is running and performing its function; no result data is produced	N	time-out or forced termination

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

			C	the instance completed and data is ready
C	completed	The instance has completed and resulting data is ready to be collected	N	time-out, forced termination or collection of data

Table 3. States and transitions of the service instance

There are two fundamental methods to control the execution of the instance: polling or waiting. Both methods are illustrated below.

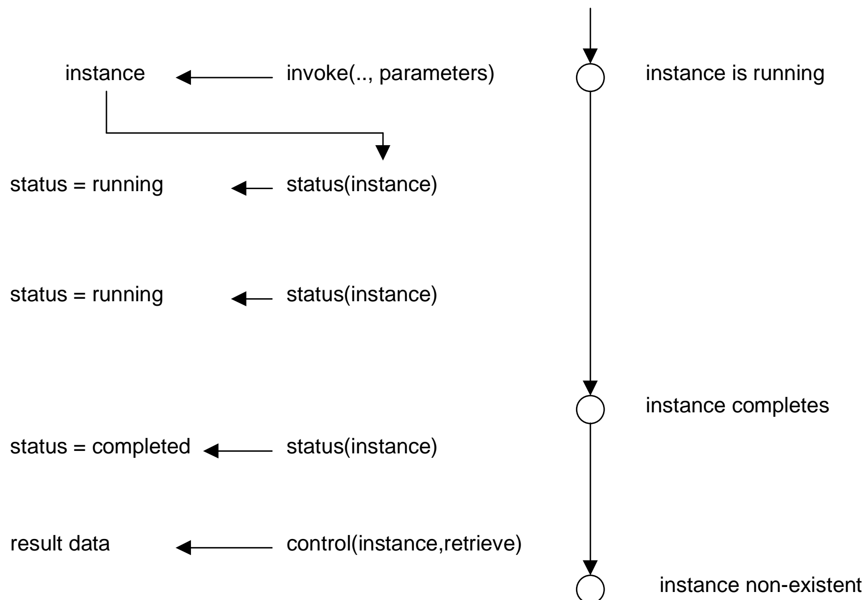

Figure 5. Polling

In polling, the application uses the instance identifier to query the current status of the instance. When execution is completed, the application uses separate functions to retrieve result data, which causes the instance to terminate

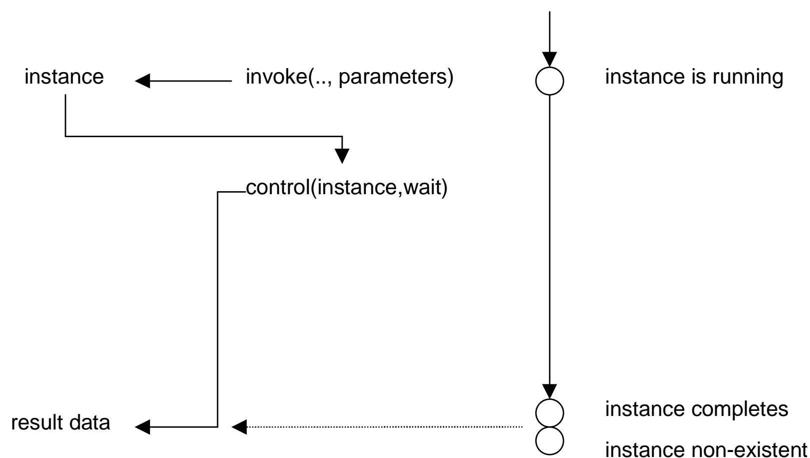

Figure 6. Waiting

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

When waiting is used, an application uses the instance identifier to wait for the completion of an execution. When the instance of the service completes, the application is resumed with result data and the instance becomes non-existent.

In order to simplify application development, EFI provides the 'call' invocation method that combines invoke and wait together, so that the application simply receives result data. This method is demonstrated on the following Figure.

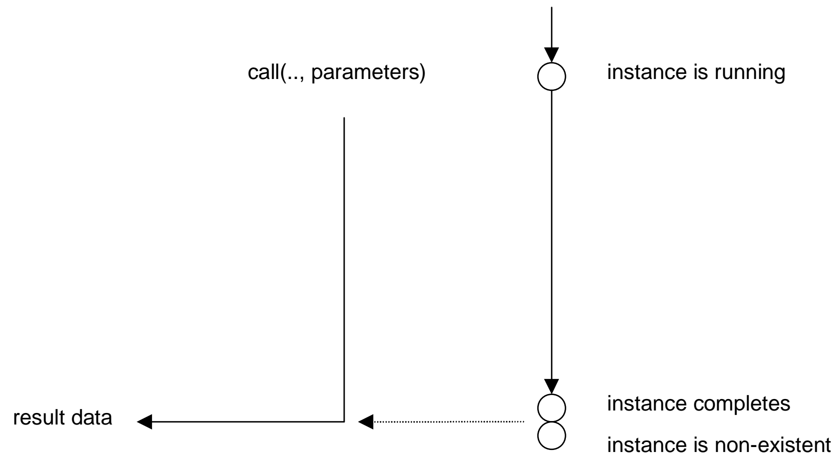

Figure 7. Calling (invoke and wait)

The following table defines all the possible interactions between the application and the instance. This table does not define the complete behaviour of the instance. Note that the instance may move to another state as a result of a call from the application, as a result of its own activity or at time-out. The meaning of each column of the table is as follows.

- Function - name of the function on the script API. For the interaction that is caused by the control function (terminate, wait and retrieve), the descriptive name of the 'action' parameter is provided in this column.
- Input - expected input for the function; 'parameters' are parameters for the service; see the definition of the particular function for more details
- Service instance state - the state of the instance when the function is called; the state may change as the result of function call
- Application waits for state - if the state is provided in this column, the application that is calling the function suspends its activity until the instance reaches the desired state;
- Output - expected output from the function; 'data' is data returned by the service; see the definition of the particular function for more details
- Next state - if the state is provided in the column and is different from the current state of the instance, the instance changes its state to the given one as the result of the function
- Comment - if the reference is provided in the column, see the relevant comment below the table.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

Function	Input	Service instance state	Application waits for state	Output	Next state	Comment
invoke	parameters	N	R	instance	R	
			N	invalid	N	(1)
status	instance	N		invalid	N	
		R		running	R	
		C		completed	C	
terminate	instance	N		invalid	N	
		R		empty	N	
		C		empty	N	
wait	instance	N		invalid	N	
		R	C	data	N	(2)
			N	invalid	N	(2) (3)
		C		data	N	
retrieve	instance	N		invalid	N	
		R		invalid	R	
		C		data	N	
call	parameters	N	N	invalid	N	(1)
			C	data	N	

Table 4. Interaction between the application and the instance

(1) If the new instance cannot be created, it immediately reaches the N state.

(2) If more than one application requested wait on the same instance, all applications must wait and then continue once the expected state is reached.

(3) If more than one application requested wait on the same instance, all applications must receive the identical copy of the result data.

NOTE: If the implementation does not allow for multiple applications to execute in parallel (therefore only one application can request wait or wait on the instance), requirements (2) and (3) are not applicable.

Time-out is mandatory and is specified at the time of invocation. The application cannot specify an 'indefinite' time-out. The time-out time is counted as the elapsed time in seconds from the moment the instance leaves the 'non-existent' state.

Following rules are to be observed by the implementation.

1. When the service is invoked, the implementation is responsible to verify whether it is possible to start the new instance of the service. If it is not possible, the application is notified by the proper error code. Error codes of values greater than -1000 are reserved for the Framework. Error codes that are currently used are defined in section 7.5.2. The class specification may use integers that are less then or equal to -1000 to specify error codes for its services.

2. When the instance of the service completes, it verifies whether any application waits for the completion. If the application waits, the instance resumes the application and passes all result data to it. If more than one application requested wait on the same instance, all applications are resumed and receive identical copies of the result data. After resuming waiting tasks and transferring data, the instance terminates and any internal copies of result data are deleted. The instance identifier may then be re-used.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

3. If no application waits for the completion of the instance of the service, the instance MUST retain its result data and the instance identifier is not re-used. The instance becomes non-existent, internal copies of result data should be deleted and instance identifier may be re-used in one of the following cases:
    a. An application asks for result data of the given instance
    b. The timeout for the service instance has expired
    c. The implementation has run into 'low memory' condition and recovers some of its memory resources.
4. Instance identifiers should not be re-used immediately.

7.5.1 Service control codes

The API method used to control services utilizes the following enumeration of control options.

A Class Specification MAY allow for more action codes for the given service. Codes assigned by Class Specifications MUST be greater than or equal to 100.

Name	Value	Description	Input (*in*)	Output (*out*)
terminate	3	Forces the instance to terminate immediately, so that the instance reaches the N state. The instance must be in the R or C state.	Not required (empty container)	No results are passed from the instance. Returns empty container if the instance has terminated Returns invalid if the instance does not exist (e.g. already terminated)
wait	4	Waits for the instance to complete. The instance must be in the R or C state. The application resumes once the instance completes. If the instance has completed before the wait control has been issued, the function immediately retrieves data and the application continues. The instance is moved to the N state.	Not required (empty container)	Returns data as returned by the instance. If there is more than one application waiting for the instance, all applications receive a copy of the return data. Returns invalid if the instance does not exist.
retrieve	5	Retrieves data from completed instance. The instance must be in the C state. The instance is moved to the N state.	Not required (empty container)	Returns data as returned by the instance. Returns invalid if the instance does not exist or if the instance is not in the C state.

Table 5. Actions of the service control method

7.5.2 Error and status codes

Following is the list of error and status codes that may be returned by methods that invoke and control services.

Code	Name	Meaning
-100	Continue	The execution of the service is in progress (the instance is in the R state); used by invoke() to report successful start of the instance that reached R state; used by status() to report that the instance is running.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

-200	OK	The execution of the service completed successfully (the instance is in the C state); used by invoke() to report successful start of the instance that reached C state during the execution of the invoke() function; used by status() to report that the instance has completed
-400	Bad Request	Error in input parameters of the function, including the ill-formed name of the service, the service cannot be executed
-404	Not Found	Server or service cannot be found
-409	Conflict	Service cannot be executed due to a temporary conflict (e.g. lack of shared resource); the conflict may be removed later.
-500	Internal Server Error	Internal error reported by any component of EFI
-503	Service Unavailable	Instance cannot be executed, e.g. due to the lack of resources or due to inability to handle multiple instances of the same service or other reasons.

Table 6. Error and status codes

7.6 WMLScript API

The WMLScript API has been *deprecated*, as is WMLScript in the Version 2 Wireless Application Environment. Please see [WAE] for more information on scripting in the wireless application environment.

A WMLScript encoder that supports EFI MUST support all EFI library functions defined in this chapter.

The EFI script library MUST have a LibraryID 7 and is named 'EFI'.

7.6.1 Containers

The WMLScript API makes extensive use of the concept of a container. The container is a structure that is capable of storing several named values. Each named value MUST have a unique name. Both names and values are case-sensitive as per section 5.1.4. With names, leading and trailing spaces in names are ignored and MUST NOT be stored into the container. Names that are identical to the empty string MUST NOT be stored in the container. Values that include spaces or are equal to the empty string MUST be stored as such.

Due to the inability of WMLScript to handle structures directly, the EFI library provides functions to simplify the usage of containers by applications. Containers are internal structures of EFI. Use of containers for purposes other than those defined by EFI is discouraged.

The internal structure of the variable that holds containers is specific to the implementation and is not defined by the Framework (the container is an opaque variable) with the exception that the empty string is equivalent to the empty container.

Even though the container is designed as an opaque variable, the language may allow for some direct operations on the container, e.g. if interpreted as a string. Such operations are discouraged as non-portable and may result in the incorrect structure of the container.

The maximum size of the container is equal to the maximum size of the variable of the type String [WMLScript]. The container implements internal ordering of name=value pairs. The insertion of a new name=value pair or the modification of the value may modify the order of other name=value pairs.

The WMLScript library defines a set of functions to handle containers. These are not EFI services, but should be considered variable manipulation functions that are placed in the EFI library for convenience.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

7.6.1.1 set

There is only one function that can be used to modify the container and this function is used both to add a new name=value pair to the container and to modify an existing one. There is no function to remove a name=value pair from the container.

The function operates on the container that is passed to the function as a parameter and returns the modified container.

The contents of the returned, modified container may have an internal order of name-value pairs different from the original one even if change affects only the value part. Care should be taken if set() is used within the getFirstName()/getNextName() loop, as modifications to the container may result in having some pairs not processed.

FUNCTION: outContainer=EFI.set(inContainer, name, value);

FUNCTION ID: 0

DESCRIPTION: Adds the name=value pair or modifies the value of the name=value pair in the *inContainer*, creating the *outContainer*. If the name does not exist, adds the name=value pair to the container. If the name already exists, modifies the value part by storing the new value. As a result of this function, the container returned by the function is modified accordingly.

This function may change the internal ordering of elements in the container. Specifically, if used between getFirstName/getNextName, not all name=value pairs may be processed

PARAMETERS:

inContainer
: The container whose contents should be modified

name
: The 'name' element of the pair, case-sensitive

value
: The 'value' element of the pair, case-sensitive

RETURN VALUE:

- If the function performs correctly, it returns the modified container
- If there are errors in parameters, if the structure of the container is invalid or when it is impossible to modify or add the name=value pair (e.g. due to memory constraints), the function returns *Invalid*.

EXAMPLE

```
var cont;  // container

cont=EFI.set("", "Parameter1", "123");
cont=EFI.set(cont, "Parameter2", "456");

// cont contains: Parameter1=123
//                Parameter2=456

cont=EFI.set(cont, "Parameter1", "ABC");

// cont contains: Parameter1=ABC
//                Parameter2=456
```

7.6.1.2 get

This is the only function to access values stored in the container. In order to access the value, the name part of the name=value pair must be known.

Values can be extracted from the container by providing the name part of the name=value pair. Value retrieval does not change the container. Specifically, the internal order of elements is not changed.

FUNCTION: *value* = EFI.get(*container, name*);

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

FUNCTION ID: 1

DESCRIPTION: Retrieves the value from the container by specifying the name. The container remains unchanged. Specifically, the internal order of elements is not changed.

PARAMETERS: *container*

The container from which the value is retrieved

name

The name that is sought, case-sensitive. Search for a name equal to the empty string returns the empty string.

RETURN VALUE:

- If the function performed correctly, it returns the 'value' element of the name=value pair
- If the container does not contain the specified name the function returns the empty string.
- If the structure of the container is invalid the function returns *Invalid*.

EXAMPLE

```
var cont;  // container
var value;

cont="";
cont=EFI.set(cont, "Parameter1", "123");
cont=EFI.set(cont, "Parameter2", "456");

value = EFI.get(cont, "Parameter1");
// value now is "123"
```

7.6.1.3 getFirstName

Two functions perform the iteration of the container. The first one, getFirstName() returns the first name stored in the container according to the internal order of name=value pairs. The second one, getNextName() provides the name that is 'next' to the given one according to the current internal order of name=value pairs.

Both functions are designed to be used together as a loop that parses the name=value pairs in the container. Note that the set() function may modify the internal order of the container and should not be used within the body of the loop.

FUNCTION: *name* = EFI.getFirstName(*container*);

FUNCTION ID: 2

DESCRIPTION: Returns the first name from the container according to the internal ordering of names in the container. The function can be used to iterate the contents of the container. The order of names is not specified, but the use of getFirstName and subsequent calls to getNextName guarantees that all names are processed.

PARAMETERS: *container*

Container that is examined.

RETURN VALUE:

- If the function performed correctly, it returns the first name from the container.
- If there are no names (i.e. container is empty) the function returns empty string.
- If the container is not correctly formed the function returns *Invalid*.

EXAMPLE

```
see "getNextName"
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

7.6.1.4 getNextName

FUNCTION: *name* = EFI.getNextName(*container, current*);

FUNCTION ID: 3

DESCRIPTION: Returns the name from the given container that is stored as the next one (according to internal ordering) after any name that exists in the container, for example after the name that has been retrieved with the previous getFirstName or getNextName. Can be used to iterate the container. The order of names is not specified, but the use of getFirstName and subsequent calls to getNextName guarantees that all names are processed.

PARAMETERS: *container*

Container that is examined.

current

Name of one of the name=value pairs that exists in the container.

RETURN VALUE:

- If the function performed correctly, it returns the next name after the *current* one.
- If there is no name after the *current* one, or if the *current* name cannot be found in the container, including the case when the *current* name is the empty string, the function returns empty string.
- If the container is not correctly formatted the function returns *Invalid*.

EXAMPLE

```
// look for the name of the parameter that has value 'xxx'

var cont;  // container in question
var name;
...

name=EFI.getFirstName(cont);
while(name!="")
{
  if(EFI.get(cont,name)=="xxx")
  {
    break;
  }
  name=EFI.getNextName(cont,name);
}
```

7.6.2 Server Attributes

The library defines two attribute management functions. The first one allows the application to collect all the attributes of the given server into the container so that they can be iterated or analysed. The second one can be used when the name of the attribute is known to the application and the application is interested in the value of the given parameter only.

7.6.2.1 getAllAttributes

FUNCTION: *container* = EFI.getAllAttributes(*server*);

FUNCTION ID: 4

DESCRIPTION: Returns all the attributes for the given server in a form of a container. Attributes that are not specified by EFI are also included.

PARAMETERS: *server*

The identifier of the server, as specified by the naming convention, without surrounding slash characters. Syntactically identical with the Server part of the namespace.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

RETURN VALUE:
- If the function performed correctly, it returns the container with name=value pairs, one pair for each attribute. Any previous content of the container is erased.
- If the server has no attributes the function returns empty container.
- If the server does not exist or if attributes cannot be returned e.g. due to memory constraints the function returns *Invalid*.

EXAMPLE

```
var cont;  // container
var value;

// get attributes of the default unit of the class 'wallet'

cont  = EFI.getAllAttributes("wallet");

value = EFI.get(cont, "Manufacturer"); // value = "MyCompany"
value = EFI.get(cont, "Name");         // value =
"TheNameOfMyUnit"
value = EFI.get(cont, "VersionMajor"); // value = "1"
value = EFI.get(cont, "Options");      // value = "OptionalInfo"
```

7.6.2.2 getAttribute

FUNCTION: *value* = EFI.getAttribute(*server, name*);

FUNCTION ID: 5

DESCRIPTION: Returns the value of a specified attribute for the given server. Can be used to retrieve values of attributes that are specified by EFI or defined by the server.

PARAMETERS: *server*

The identifier of the server, as specified by the naming convention, without surrounding slash characters. Syntactically identical with the Server part of the namespace.

name

Name of the attribute of the given server. Case-insensitive.

RETURN VALUE:
- If the function performed correctly, it returns value of the given attribute.
- If the server does not exist or if the value cannot be returned e.g. due to memory constraints the function returns *invalid*.

EXAMPLE

```
var   value;

value = EFI.getAttribute("wallet", "Manufacturer");

// value = "MyCompany"
```

7.6.3 Class Properties

The library defines one property management function. This function allows an application to get the value of the property of the known name. There is no method to verify what properties other than the mandated set are available from the Broker.

7.6.3.1 getClassProperty

FUNCTION: *value* = EFI.getClassProperty(*class, name*);

FUNCTION ID: 6

DESCRIPTION: Retrieves the value of the specified property for the given class realisation. Can be used to retrieve values of attributes that are specified by EFI or are defined additionally for the given class realisation.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

PARAMETERS: *class*

Name of the class, syntactically identical with the following parts of the namespace
Def-Unit-Spec-Class
Def-Unit-Vnd-Class

name

Name of the property of the given class.

RETURN VALUE:
- If the function performed correctly, it returns the value of the given property.
- If there is no property of the given name the function returns empty string.
- If the class does not exist or if the value cannot be returned due to memory constraints the function returns *Invalid*.

EXAMPLE

```
var   value;

value = EFI.getClassProperty("wallet", "MinValueMajor");
```

// value = 1

7.6.4 Service Discovery

The service discovery functions allow applications to find out what services are available. Class realizations may decide to hide certain services, in which case they will not be accessible using the service discovery functions. Service discovery and visibility is described in section 7.4.

7.6.4.1 getUnits

This function returns the list of all units that are visible within the given class realisation. The list is provided as a string that can be parsed with functions from the String library [WMLLib].

FUNCTION: *list* = EFI.getUnits(*class*);

FUNCTION ID: 7

DESCRIPTION: Lists all units within the given class realisation.

PARAMETERS: *class*

Name of the class. Must match the complete name of the class, including dots between segments and is syntactically identical with the following parts of the namespace.
Def-Unit-Spec-Class
Def-Unit-Vnd-Class

The name is case-insensitive.

RETURN VALUE:
- If the function performed correctly, it returns the string that contains names of all the units linked by ampersand "&". No spaces are inserted between the name of the unit and an ampersand. No ampersand is placed before the first name or after the last name. Units are reported by their identifiers, including the dot at the beginning of the identifier. The default unit of the class is not reported by the name of the class, but it is rather included in the list of units that are listed by their identifiers. Units are reported at no specific order.
- If the parameter does not match any class realisation the function returns *Invalid*. Note that the class realisation must contain at least one visible unit in order to be visible to this function. Therefore this function never returns the empty string.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

EXAMPLE

```
var   value;
var   count;

value = EFI.getUnits("wallet");

// note that this particular implementations use cardinal
numbers
// as unique names in the container returned

// value = ".uw001&.uw002"
// see how many units are reported

count=String.elements(value,"&");
```

7.6.4.2 query

This function can be used when the application knows the server's identifier. It can also be used in conjunction with getUnits() to scan all servers for the presence of the given service.

Note that query is about servers, not about class realisations. Specifically, if the default unit of the class realisation elects not to be visible, a query that uses only the name of the class (i.e. a query about the default unit of the class realisation) will return 'false' even though the class realisation may exist.

Note that there is no function to discover the complete list of services provided by the server.

FUNCTION: *yesno* = EFI.query(*name*);

FUNCTION ID: 8

DESCRIPTION: Allows the application to determine whether a particular server or service exists.

PARAMETERS: *name*

Name of the server or service. For servers use the notation identical with Server part of the namespace. All variants of the name are accepted, including the use of class name to identify the default unit. Names are case-insensitive.

For services use the name that is identical with the Server/Service parts of the namespace.

RETURN VALUE:

- If the component exists the function returns *true*.
- If the component does not exist or is not visible the function returns *false*. Note that the server may elect whether it is visible for this function.
- If the parameter format is incorrect the function returns *Invalid.*

EXAMPLE

```
var   v;

v = EFI.query("wallet"); // default unit of the class wallet
v = EFI.query("vnd.acme"); // default unit of vendor-specific
                          // class
v = EFI.query("brake.agent"); // class agent of  class 'brake'
v = EFI.query("wallet/pay");  // service pay at the default unit
                              // of the class wallet
v = EFI.query(".u001/status");// service 'status' on server u001
```

7.6.5 Service Control

The following functions allow applications to launch services and optionally block on the completion of the service. Additionally, services that have been launched can be controlled by the application. Service instantiation and control is described in more detail in section 7.5.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

7.6.5.1 invoke

FUNCTION: *out* = EFI.invoke(*service, timeout, inContainer*);

FUNCTION ID: 9

DESCRIPTION: Invokes the instance of the service, with input values defined in the *inContainer* and returns control back to the application.

PARAMETERS: *service*

Name of the service, identical with the namespace segments Server/Service

timeout

Maximum allowed time (in seconds) for the instance to run. The instance will move into the non-existent state when it exceeds the *timeout*.

Setting *timeout* to zero sets the timeout to the pre-defined value that is specified for the service by class specification.

inContainer

The container that contains all the input parameters that are necessary for the service in a form of 'name=value' pairs. The list of required and optional parameters is provided by the specification of the service.

RETURN VALUE:

- If the instance started successfully the function returns the non-negative instance identifier.
- If the instance did not start, the method returns a negative error code as described in section 7.5.2 or in the class specification.

EXAMPLE See section 7.6.5.5 below.

7.6.5.2 call

FUNCTION: *out* = EFI.call(*service, timeout, inContainer*);

FUNCTION ID: 10

DESCRIPTION: Invokes the instance of the service identified by name with input values defined in the *inContainer*. Waits for the completion of the service and returns data to the application.

PARAMETERS: *service*

Name of the service, identical with the namespace segments Server/Service

timeout

Maximum allowed time (in seconds) for the instance to run. The instance will move into the non-existent state when it exceeds the *timeout*.

Setting *timeout* to zero sets the timeout to the pre-defined value that is specified for the service by class specification.

inContainer

The container that contains all the input parameters that are necessary for the service in a form of 'name=value' pairs. The list of required and optional parameters are provided by the specification of the service.

RETURN VALUE:

- If the instance completed successfully, the function returns the container that contains all the values returned by the service in a form of 'name=value' pairs. The list of returned parameters is provided by the specification of the service.
- If the service does not exist or if the service cannot be executed for any reason the function returns *Invalid*. Detailed error diagnostic is not available.

EXAMPLE See section 7.6.5.5 below.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

7.6.5.3 status

FUNCTION: *status* = EFI.status(*instance*);

FUNCTION ID: 11

DESCRIPTION: Provides the current status of an instance.

PARAMETERS: *instance*

An instance of the service, as reported by the 'invoke' function.

RETURN VALUE:

- If the function performed correctly, it returns the current status of the instance, as listed in the table below. Only the following codes are used:

-100	instance is in the running state
-200	instance is in the completed state

- If the instance identifier cannot be recognised the function returns *Invalid.* This applies also to instances that are in the non-existent state.

EXAMPLE See section 7.6.5.5 below.

7.6.5.4 control

FUNCTION: *out* = EFI.control(*instance, action [, parameters]*);

FUNCTION ID: 12

DESCRIPTION: Sends control commands to the instance.

PARAMETERS: *instance*

An instance of the service, as reported by the 'invoke' function.

action

Numerical identifier of the required action. See section 7.5.1 for the detailed description of actions.

parameters

The container that contains all the input parameters that are necessary for the service in a form of 'name=value' pairs. The list of required and optional parameters are provided by the specification of the service. Some actions for some services may not require input parameters.

RETURN VALUE:

- If the function performed correctly and if the instance has data to be passed to the application, the function returns the container that contains all the values returned by the service in a form of 'name=value' pairs. Only some services return data as a result of some actions. The list of actions and returned values are provided by the specification of the service.
- If the function performed correctly but the instance has no data to be passed to the application, the function returns empty container (empty string).
- If the instance cannot be recognised or if the given action cannot be performed in the given context the function returns *Invalid*.

EXAMPLE See section 7.6.5.5 below.

7.6.5.5 Example of Service Instantiation and Control

The following example illustrates a script invocation of EFI services.

In this example, the unit of the hypothetical class 'printer' is used to print the required text on the banner printer. The script verifies the existence of the class performing the actual print request. The print service is controlled by the 'wait' function.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

```
extern function printBanner(text)
{
       var input;   // container with input parameters
       var out;  // container with return value(s)
       var instance;

       // check the existence of the class realisation
       // by examining its default unit
       if(!EFI.query("printer"))
       {
             return false;
       }

       // Prepare your input container for the call
       input=EFI.set("", "colour", "black");
       input=EFI.set(input, "text", text);
       input=EFI.set(input, "size", "128");

       // call the print service
       instance = EFI.invoke("printer/print", 300, input);

       // check if service started OK
       if(instance<0)
       {
             return false;
       }

       // the application can do something in parallel while
       // doing the printout

       // wait for the service to complete
       out = EFI.control(instance,4,"");

       // return, service done
       return true;
}
```

The next example modifies the previous one by calling the service rather than invoking it. The application waits for the completion of the service. This example also does not check for the existence of the class realisation.

```
extern function printBanner(text)
{
       var input;   // container with input parameters
       var out;  // container with return value(s)

       // Prepare your input container for the call
       input="";
       input=EFI.set(input, "colour", "black");
       input=EFI.set(input, "text", text);
       input=EFI.set(input, "size", "128");

       // call the print service
       out = EFI.call("printer/print", 300, input);

       // return what came out (assumes success)
       return out;
}
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

7.7 ECMAScript API

The ECMAScript API is realized as a single ECMAScript Object that is named “Efi”. This Object contains all the methods required to interact with servers, classes, and services.

The ECMAScript API also utilizes a new native error type called ‘EfiError’. This error type is a constant with the value ‘200’, a reserved Native Error Type in the ECMAScript Mobile Profile Error (Exception) Object, see [ESMP]. The instances in which this error is thrown are described in the individual methods below.

Mobile clients supporting the ECMAScript API MUST support the Efi Object, and MUST use the EfiError error type to indicate error conditions as specified in the API.

7.7.1 Name/value collections

Parameter passing between ECMAScript applications and EFI services makes extensive use of name/value pairs. In the ‘Efi’ Object, these name/value pairs are represented using a two dimensional array. It is important to note that ESMP does not support true multidimensional arrays. However, ESMP does however support an array of arrays and a collection of parameters should be constructed as such. When creating a name/value collection, the parameter name MUST be the first element in the secondary array, and the value MUST be the second element in the secondary array. In practice, this would be implemented in the following way:

```
var param1 = new Array( name, value );
var param2 = new Array( name, value );
var myParams = new Array( param1, param2 );
```

In this example, myParams[0][0] would be the name, and myParams[0][1] would be the value; myParams[1][0] would be the name, and myParams[1][1] would be the value, and so on.

7.7.1.1 Parameter names

The formatting of parameter names is defined in section 5.1.4. With names, leading and trailing spaces in names are ignored and MUST be ignored by EFI services. Names MUST NOT equate to the empty string. All names in the collection MUST be unique.

7.7.1.2 Parameter values

The formatting of parameter values is defined in section 5.1.4.Values that include spaces or are equal to the empty string MUST be stored as such.

7.7.1.3 Version History

Version	Affected	Comment
1.1		Initial release.

7.7.1.4 Properties

7.7.1.4.1 version

The current version of the object is defined in 7.7.1.3. This property is a string in the format defined in the Object Management section of [ESMP].

7.7.1.5 Server Attribute Methods

The Object defines two attribute management methods. The first one allows the application to collect all the attributes of the given server into an array so that they can be iterated or analysed. The second one can be used when the name of the attribute is known to the application and the application is interested in the value of the given attribute only.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

7.7.1.5.1 getAllAttributes()

Syntax:	*anArray* = Efi.getAllAttributes(*server*)
Argument List:	*server* - The identifier of the server, as specified by the naming convention, without surrounding slash characters. Syntactically identical with the Server part of the namespace.
Description:	Allows the application to collect all the attributes of the given server. Server attributes are described in section 7.2.
Return Value Type:	Name/value collection as described in section 7.7.1.
Errors or Exceptions:	EfiError is thrown if the specified server does not exist.

Example:	`var anArray = Efi.getAllAttributes("wallet");`
	`for ( i=0; i<anArray.length; i++ )`
	`{`
	`if ( anArray[i][0] == "Manufacturer" )`
	`var manufacturer = anArray[i][1];`
	`}`

7.7.1.5.2 getAttribute()

Syntax:	*anAttr* = Efi.getAttribute(*server, name*)
Argument List:	*server* - The identifier of the server, as specified by the naming convention, without surrounding slash characters. Syntactically identical with the Server part of the namespace. *name* – Case-insensitive name of the attribute.
Description:	Returns the value of a specified attribute for the given server. Can be used to retrieve values of attributes that are specified by EFI or defined by the server. Server attributes are described in section 7.2.
Return Value Type:	Value of the attribute as described in section 7.7.1.2
Errors or Exceptions:	EfiError is thrown if the specified server or attribute does not exist.

Example:	`// anAttr is a name/value collection`
	`var manufacturer = Efi.getAttribute("wallet",`
	`"Manufacturer");`

7.7.1.6 Class Property Methods

The Object defines one property management method. This method allows an application to get the value of the property of the known name. There is no method to verify what properties other than the mandated set are available from the Broker.

7.7.1.6.1 getClassProperty()

Syntax:	*value* = Efi.getClassProperty(*class, name*);
Argument List:	*class* - Name of the class, syntactically identical with the following parts of the namespace Def-Unit-Spec-Class Def-Unit-Vnd-Class *name* - Name of the property of the given class.
Description:	Retrieves the value of the specified property for the given class realisation. Can be used to retrieve values of attributes that are specified by EFI or are defined additionally for the given class realisation.
Return Value Type:	Any or empty string if property does not exist
Errors or Exceptions:	EfiError if class or property does not exist

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

Example:	`var value;`
	
	`value = Efi.getClassProperty("wallet", "MinValueMajor");`
	
	// value = 1

7.7.1.7 Service Discovery Methods

The service discovery methods allow applications to find out what services are available. Class realizations may decide to hide certain services, in which case they will not be accessible using the service discovery methods. Service discovery and visibility is described in section7.4.

7.7.1.7.1 getUnits()

Syntax:	*list* = Efi.getUnits(*class*);
Argument List:	*class* – Case-insensitive name of the class, syntactically identical with the following parts of the namespace Def-Unit-Spec-Class Def-Unit-Vnd-Class
Description:	Lists all the units within a class realisation.
Return Value Type:	Array
Errors or Exceptions:	EfiError if class does not exist

Example:	`units = Efi.getUnits("wallet");`
	`// Note that actual unit names are specific to the`
	`// implementation, for example:`
	`// units[0] = ".uw001"`
	`// units[1] = ".uw002"`

7.7.1.7.2 query

This function can be used when the application knows the server's identifier. It can also be used in conjunction with getUnits() to scan all servers for the presence of the given service.

Note that query is about servers, not about class realisations. Specifically, if the default unit of the class realisation elects not to be visible, a query that uses only the name of the class (i.e. a query about the default unit of the class realisation) will return 'false' even though the class realisation may exist.

Note that there is no function to discover the complete list of services provided by the server.

Syntax:	*yesno* = Efi.query(*name*);
Argument List:	*name* - Name of the server or service. For servers use the notation identical with Server part of the namespace. All variants of the name are accepted, including the use of class name to identify the default unit. Names are case-insensitive. For services use the name that is identical with the Server/Service parts of the namespace.
Description:	Allows the application to determine whether a particular server or service exists.
Return Value Type:	boolean
Errors or Exceptions:	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

Example:	`// default unit of the class wallet`
	`yesno = Efi.query("wallet");`
	
	`// default unit of vendor-specific class`
	`yesno = Efi.query("vnd.acme");`
	
	`// class agent of class 'brake'`
	`yesno = Efi.query("brake.agent");`
	
	`// service pay of the default unit of class wallet`
	`yesno = Efi.query("wallet/pay");`
	
	`// service 'status' on server u001`
	`yesno = Efi.query(".u001/status");`

7.7.1.8 Service Control Methods

The following methods allow applications to launch services and optionally block on the completion of the service. Additionally, services that have been launched can be controlled by the application. Service instantiation and control is described in more detail in section7.5.

7.7.1.8.1 invoke()

Syntax:	*out* = Efi.invoke(*service, timeout, parameters*);
Argument List:	*service* - Name of the service, identical with the namespace segments Server/Service *timeout* - Maximum allowed time (in seconds) for the instance to run. The instance will move into the non-existent state when it exceeds the *timeout*. Setting *timeout* to zero sets the timeout to the pre-defined value that is specified for the service by class specification. *parameters* – Input parameters needed by the service. The *parameters* variable is a collection of name/value pairs as described in section 7.7.1. The list of required and optional parameters is provided by the specification of the service.
Description:	Invokes the instance of the service, with *parameters* and returns control back to the application.
Return Value Type:	integer
Notes:	If the instance started successfully the function returns the non-negative instance identifier. If the instance did not start, the method returns a negative error code as described in section 7.5.2 or in the class specification.
Errors or Exceptions:	EfiError is thrown if the service does not exist. TypeError is thrown if *parameters* is not a collection of name/value pairs as described in section 7.7.1.
Example:	See section 7.7.1.8.5 below.

7.7.1.8.2 call()

Syntax:	*out* = Efi.call(*service, timeout, parameters*);

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

Argument List:	*service* - Name of the service, identical with the namespace segments Server/Service *timeout* - Maximum allowed time (in seconds) for the instance to run. The instance will move into the non-existent state when it exceeds the *timeout*. Setting *timeout* to zero sets the timeout to the pre-defined value that is specified for the service by class specification. *parameters* – Input parameters needed by the service. The *parameters* variable is a collection of name/value pairs as described in section 7.7.1. The list of required and optional parameters is provided by the specification of the service.
Description:	Invokes the instance of the service, with *parameters* and returns data to the application.
Return Value Type:	Name/value collection as described in section 7.7.1.
Notes:	If the instance completed successfully, the method returns all the values returned by the service in a form of a collection of name/value pairs as described in section 7.7.1. The list of returned parameters is provided by the specification of the service.
Errors or Exceptions:	EfiError is thrown if the service does not exist. TypeError is thrown if *parameters* is not a collection of name/value pairs as described in section 7.7.1.
Example:	See section 7.7.1.8.5 below.

7.7.1.8.3 status()

Syntax:	*status* = Efi.status(*instance*);
Argument List:	*instance* - An instance of the service, as reported by the 'invoke' function.
Description:	Provides the current status of the instance as defined in section 7.5.2. Only the following codes are used: -100 instance is in the running state -200 instance is in the completed state
Return Value Type:	integer
Errors or Exceptions:	EfiError is thrown if the instance does not exist.
Example:	See section 7.7.1.8.5 below.

7.7.1.8.4 control()

Syntax:	*out* = Efi.control(*instance, action [, parameters]*);
Argument List:	*instance* - An instance of the service, as reported by the 'invoke' function. *action* - Numerical identifier of the required action. See section 7.5.1 for the detailed description of actions. *parameters* - Input parameters needed by the service. The *parameters* variable is a collection of name/value pairs as described in section 7.7.1. The list of required and optional parameters is provided by the specification of the service. Some actions for some services may not require input parameters.
Description:	Sends control commands to the instance.
Return Value Type:	If the instance has data to be passed to the application, the method returns a collection of name/value pairs as described in section 7.7.1 that contains all the values returned by the service. Only some services return data as a result of some actions. The list of actions and returned values are provided by the specification of the service. If instance has no data to be passed to the application, the function returns an empty Array.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

Errors or Exceptions:	EfiError is thrown if the instance does not exist or the action cannot be performed. TypeError is thrown if *parameters* is not a collection of name/value pairs as described in section 7.7.1.
Example:	See section 7.7.1.8.5 below.

7.7.1.8.5 Example of Service Instantiation and Control

The following example illustrates a script invocation of EFI services.

In this example, the unit of the hypothetical class 'printer' is used to print the required text on the banner printer. The script verifies the existence of the class performing the actual print request. The print service is controlled by the 'wait' function.

```
function printBanner(text)
{
        Array input;   // input parameters
        Array out;  // return value(s)
        var instance;

        // check the existence of the class realisation
        // by examining its default unit
        if( !Efi.query( "printer" ) )
        {
                return false;
        }

        // Prepare your input parameters for the call
        input = new Array(3);
        input[0] = new Array( "colour", "black" );
        input[1] = new Array( "text", text );
        input[2] = new Array( "size", 128 );

        // call the print service
        instance = Efi.invoke( "printer/print", 300, input );

        // check if service started OK
        if( instance<0 )
        {
                return false;
        }

        // the application can do something in parallel while
        // doing the printout

        // wait for the service to complete
        out = Efi.control( instance, 4, "" );

        // return, service done
        return out;
}
```

The next example modifies the previous one by calling the service rather than invoking it. The application waits for the completion of the service. This example also does not check for the existence of the class realisation.

```
function printBanner(text)
{
        Array input;   // input parameters
        Array out;  // return value(s)

        // Prepare your input parameters for the call
        input = new Array(3);
        input[0] = new Array( "colour", "black" );
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

```
        input[1] = new Array( "text", text );
        input[2] = new Array( "size", 128 );

        // call the print service
        out = Efi.call( "printer/print", 300, input );

        // return what came out (assumes success)
        return out;
}
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

8. Markup API

To use the Markup API, the mobile client MUST support the markup language specified by [WAE].

The Markup API maps available services into the namespace and makes them available through the architectural concept where EFI namespace co-exists with other namespaces so that the browser can direct requests that start from 'efi:' to the local broker rather than sending them out. The concept positions EFI as a server, located at the mobile client rather than at the other end of the wireless link. The implementation MAY integrate EFI and the transfer protocol stack on different levels and by different means. However, the application uses both regular transfer protocol stack and EFI in the same way.

The interaction model that is provided by Markup API is significantly different from the model that is provided by the script API. The interaction follows the simplified Web model and consists of the browser's request-response pairs. The invocation of the service is interpreted as a request to retrieve the contents of a certain URI. The request initiates the service that MUST compose and return the document in the markup language that is accepted by the User Agent, as defined by [WAE]. The browser renders and displays the retrieved document to the user. The service may be also capable of directing the browser to display a particular fragment of the document. Future releases of the framework may include mechanisms that allow EFI services to return content other than markup language that is acceptable to the User Agent.

The EFI service SHOULD honour preferences of the User Agent when it comes to the preferred markup language and it SHOULD NOT send the document in a markup language that the User Agent does not support. The method to discover the preferred markup language is not within the scope of the EFI Framework.

The Class Specification provides the exact definition of services. The Framework does not define any specific service. The Framework defines only the method to invoke services through Markup API.

When the document generated by the service is presented to the user, the user MUST be informed that the interaction is with the EFI implementation.

8.1 Behaviour of the mobile client

The implementation of EFI AI as Markup API makes extensive use of the namespace. The service is actually accessible only through the URI name scheme. The scheme is identical with the one defined by the Framework with the exception that the scheme prefix MUST be always present. The general format of URI is as follows:

```
Scheme "://" Server "/" [Service] ["?" Parameters]
```

The proper usage of the namespace allows access to services that are provided by different servers, including access to services that have no name.

Services may be initialised by using the "href" attribute in the WML and XHTML navigational elements. For example, using the WML `<go>` element:

```
<go href="efi://location/displaypos"/>
```

or the <a> element from WML or XHTML:

```
<a href="efi://location/displaypos">
    Current position
</a>
```

Parameters (if any) are passed to the service through the URI name=value format or through the wml.postfield structure or through a mix of both methods. For example

```
<a href=”efi://wallet/pay?value=200&amp;currency=USD" >
```

or

```
<go href=”efi://wallet/pay”>
    <postfield name="value" value="200"/>
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

```
        <postfield name="currency" value="USD"/>
    </go>
```

Note that EFI makes it possible to access services standardised by OMA and vendor-specific services. In both cases the same notation applies.

8.2 Servers

Following is the detailed discussion of the name scheme and the behaviour of the mobile terminal when different servers are accessed.

8.2.1 EF Broker

To access the EF Broker, the following general notation is used:

```
"efi:///" Service ["?" Parameters]
```

There are no services currently defined by the Framework that can be requested from the Broker through Markup API, except the service with no name. The EF Broker SHOULD provide a service with no name. The exact contents of the document returned depends on EF Broker implementation, but the content of the document MUST present the list of available servers in a readable format with content equivalent to the service discovery function in the script interface, including also class agent if it exists. Note that some servers may elect not to be visible to service discovery functions. This election applies also to this service. If the EF Broker does not provide the no-name service, the 404 service code ("Not Found") is returned.

8.2.2 EF Class Agent

In order to access the class agent one of the following general notation is used

```
"efi://" Classagent-Spec-Class "/" Service ["?" Parameters]

"efi://" Classagent-Vnd-Class "/" Service ["?" Parameters]
```

The EF Class Agent SHOULD provide a service with no name. The contents of the EF Class Agent document that is returned by this service depends upon the EFI Class specification, but the content of the document MUST present the name of the server. If the EF Class Agent does not provide the no-name service, the 404 service code ("Not Found") is returned.

The Class Specification defines details of the no-name service for its Class Agent.

8.2.3 EF Unit

In order to access the unit, one of the following notations is used.

```
"efi://" Identified-Unit "/" Service ["?" Parameters]

"efi://" Def-Unit-Spec-Class "/" Service ["?" Parameters]

"efi://" Def-Unit-Vnd-Class "/" Service ["?" Parameters]
```

In the first case the specified unit is accessed. In the latter two cases the default unit of the specific class realisation is accessed.

The EF Unit SHOULD provide a service with no name. The contents of the EF Unit document that is returned by this service depends upon the EFI Class specification, but the content of the document MUST present the name of the server. If the EF Unit does not provide the no-name service, the 404 service code ("Not Found") is returned.

The Class Specification defines details of the no-name service for its Units.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

8.3 Discontinuous mode

The EFI service, once started, takes control and generates its own documents that are processed by the browser. At this point the application effectively releases control and must rely on the service to carry on the functionality and the control flow as expected.

This discontinuity may be seen as disadvantageous for some applications. In order to partly alleviate the shortcomings of the discontinuous mode, the class specification may require services to support some of the concepts described below, namely the 'continuation document' concept.

8.3.1 Continuation document

The continuation document is used to allow the user to navigate from the EFI-generated document to a new document, or a document fragment, as specified by the content author. This prevents users from getting “stuck” in the EFI-generated document and having to back out of the generated document.

The concept of a continuation document does not require any specification within the Framework. Application developers may specify the continuation document as one of the parameters that are passed to the service, for example:

```
<go href="efi://music/play">
    <postfield name="dest" value="example.wml#done"/>
</go>
```

Names of the parameters that hold a reference to the continuation document and circumstances upon which the service chooses the specific continuation document should be defined in Class Specification.

8.3.2 Return variable

In order to return values back to the application the service can use browser variables. Return variables may be set by the service at one or more of its cards and passed back to the application as a part of the context. Note that the use of return variables is restricted to WML. Also note that the implementation of the service may not protect the context of the current WML document. Specifically, another document may become current when the service terminates.

The concept of return variable does not require any specification within the Framework. Application developers may specify the name of the return variable as one of the parameters that are passed to the service, for example:

```
<go href="efi://picture/take">
    <postfield name="ret" value="retvar"/>
</go>
```

Names of the parameters that hold the name of the return variable and the meaning of particular variables should be defined in Class Specification.

8.4 Context management

When accessed from a WML document, the service is executed within the context of a browser and makes use of the context of the caller. No new context is created unless the service decides to create one. An application has no control over the context.

The service may interfere with the context of the caller by incidentally overwriting WML variables used by the application if they are identical with variables used internally by the service. Note that the service does not protect the current WML context.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

8.5 Status codes

As the Markup API makes use of URL request/response protocol, the service may report one of the return codes as defined in [WSP] in accordance with [RFC2616].

The fact that the EFI server is local to the browser influences the interpretation of some of the codes. The following table summarises codes, their names and their meaning within EFI. The implementation of EFI uses only status codes that are not marked below as '(not used)'.

Code	Name	Meaning
100	Continue	as in [RFC2616]
101	Switch Protocols	(not used)
200	OK	as in [RFC2616]
201	Created	(not used)
202	Accepted	(not used)
203	Non-Authoritative Information	(not used)
204	No Content	(not used)
205	Reset Content	(not used)
206	Partial Content	(not used)
300	Multiple Choices	(not used)
301	Moved Permanently	(not used)
302	Found	(not used)
303	See Other	(not used)
304	Not Modified	(not used)
305	Use Proxy	(not used)
306	(Unused)	(not used)
307	Temporary Redirect	(not used)
400	Bad Request	as in [RFC2616]; to be used for ill-formed names, requests and parameter errors
401	Unauthorised	(not used)
402	Payment Required	(not used)
403	Forbidden	(not used)
404	Not Found	as in [RFC2616]; SHOULD be used for non-existing class realisations, server or services
405	Method Not Allowed	(not used)
406	Not Acceptable	(not used)
407	Proxy Authentication Required	(not used)
408	Request Timeout	as in [RFC2616]; SHOULD be used when the processing of the request takes more than the specified time.
409	Conflict	as in [RFC2616]; SHOULD be used when EFI cannot access all the required resources or devices due to possible access conflicts that may be removed later
410	Gone	(not used)
411	Length Required	(not used)
412	Precondition Failed	(not used)
413	Request Entity Too Large	as in [RFC2616]
414	Request-URI Too Long	as in [RFC2616]
415	Unsupported Media Type	as in [RFC2616]

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

416	Requested Range Not Satisfied	(not used)
417	Expectation Failed	(not used)
500	Internal Server Error	as in [RFC2616]; SHOULD be used for all broker errors and all internal errors
501	Not Implemented	as in [RFC2616]; SHOULD be used if the request is not supported
502	Bad Gateway	(not used)
503	Service Unavailable	as in [RFC2616]; SHOULD be used if there are not enough resources to handle the request or if the request has been called in the context where it cannot be handled
504	Gateway Timeout	(not used)
505	HTTP Version Not Supported	as in [WSP]; SHOULD be used if the encoding version of the request cannot be handled by the server

Table 7. Status codes

8.6 UAProf

The User Agent Profiling mechanism is defined in [UAProf].

As a side effect of the architecture, the EFI service generates documents that are displayed by the browser. This information does not pass through the gateway so that the User Agent Profile information cannot be utilised. The application and the service cannot assume that the UAProf information can be applied.

This may lead to certain inconsistencies in user experience where similar contents are rendered differently depending on whether they arrive from the origin server or from EFI service.

EFI Framework recommends that the mobile client SHOULD minimise those inconsistencies without changes to the current WAP architecture.

8.7 Cache

The cache mechanism [CACHE] MUST NOT be used when EFI services are accessed through the Markup API, regardless of information in the header.

8.8 Example

The following example shows how an EFI service might be invoked from a WML document. When invoked, the EFI service performs some action and returns a new WML document. WML is used only for explanatory purposes; other markup languages supported by the WAE may also be used to invoke EFI services. Some of the constructs in this example, such as the continuation document and return variables are discussed in Section 8.3.

For this example, assume that there is a class 'music' that defines various playback/recording functionality. A service 'play' plays back the given song. The service accepts the following three parameters:

title - the title of the given song,

dest - name of the continuation document where control is transferred upon completion of the playback,

result - name of the variable in which the duration of actual playback (in seconds) is stored.

These three parameters can be passed to the EFI service by including them in the URI itself

```
<go href="efi://music/play
       ?title=banana&amp;result=time&amp;dest=example.wml#result"/>
```

or by composing the URI using the postfield tag

```
<go href="efi://music/play">
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

```
        <postfield name="title" value="banana"/>
        <postfield name="result" value="time"/>
        <postfield name="dest" value="example.wml#result"/>
    </go>
```

Following is the example that shows the first invocation within the context of the document.

```
<!-- example.wml -->
<wml>
 <card id="play">
        <do type="accept" label="Start">
               <go href="efi://music/play
                      ?title=banana&amp;result=time&amp;dest=example.wml#result"/>
        </do>
        <p>
               Select 'Start' to play the banana
        </p>
 </card>
 <card id="result">
        That's it, $time seconds of pleasure
 </card>
</wml>
```

In this example, the EFI service might generate the following WML document to give the user a visual indication that the music is playing. The EFI-generated document also assigns the duration to the return variable and provides a soft key so the user can navigate to the continuation document.

```
<!-- EFI generated document -->
<wml>
       <card>
              <do type="accept" label="OK">
                     <!-- EFI service gets 'example.wml#result' from $dest -->
                     <go href="example.wml#result">
                            <!-- EFI service gets 'time' from $result -->
                            <!-- EFI service calculates duration of the music -->
                            <setvar name="time" value="10"/>
                     </go>
              </do>
              <p>
                     <!-- EFI service gets 'banana' from $title -->
                     Playing banana
              </p>
       </card>
</wml>
```

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

Appendix A. Static Conformance Requirements

This static conformance requirement [IOPProc] lists a minimum set of functions that can be implemented to help ensure that EFI implementations will be able to inter-operate.

The “Status" column indicates if the function is mandatory (M) or optional (O).

A.1 Script Encoder Options

Item	Function	Reference	Page	Status	Requirements
EFIFRM-LIB-S-1	Encoding of set()	7.6.1.1	30	M	
EFIFRM-LIB-S-2	Encoding of get()	7.6.1.2	30	M	
EFIFRM-LIB-S-3	Encoding of getFirstName()	7.6.1.3	31	M	
EFIFRM-LIB-S-4	Encoding of getNextName()	7.6.1.4	32	M	
EFIFRM-LIB-S-5	Encoding of getAllAttributes()	7.6.2.1	32	M	
EFIFRM-LIB-S-6	Encoding of getAttribute()	7.6.2.2	33	M	
EFIFRM-LIB-S-7	Encoding of getClassProperty()	7.6.3.1	33	M	
EFIFRM-LIB-S-8	Encoding of getUnits()	7.6.4.1	34	M	
EFIFRM-LIB-S-9	Encoding of query()	7.6.4.2	35	M	
EFIFRM-LIB-S-10	Encoding of invoke()	7.6.5.1	36	M	
EFIFRM-LIB-S-11	Encoding of call()	7.6.5.2	36	M	
EFIFRM-LIB-S-12	Encoding of status()	7.6.5.3	37	M	
EFIFRM-LIB-S-13	Encoding of control()	7.6.5.4	37	M	
EFIFRM-LIB-S-14	Encoding of library ID	7	22	M	

A.2 Client Options

A.2.1 Broker

Item	Function	Reference	Page	Status	Requirements
EFIFRM-BR-C-1	The user is informed, when communicating with the EFI implementation. Either Script or Markup API is used	7, 8.2.1	22, 47	M	
EFIFRM-BR-C-2	The Broker supports EFI scheme.	5	16	M	
EFIFRM-BR-C-3	Broker allows server to be either visible through service discovery or not. Broker exposes the names of all visible servers.	7.4	23	M	
EFIFRM-BR-C-4	Every instance returns its instance number, which is the non-negative integer.	7.5	23	M	
EFIFRM-BR-C-5	If it is not possible to invoke a particular service, the application is notified by the proper return code.	7.5	23	M	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

EFIFRM-BR-C-6	The instance identifier returned by the broker is unique within all the instances of the services that are maintained by EFI at any time.	7.5	23	M	
EFIFRM-BR-C-7	All suspended applications, waiting for one service, are resumed and receive the identical copy of the result data.	7.5	23	M	
EFIFRM-BR-C-8	The instance of a service retains its result data until at all suspended applications have been resumed.	7.5	23	M	
EFIFRM-BR-C-9	If no application waits for the completion of the instance, the instance retains its result data and the number is not re-used.	7.5	23	M	

A.2.2 Scheme

Item	Function	Reference	Page	Status	Requirements
EFIFRM-SCH-C-1	Scheme-element in Script API is omitted.	7.1	22	M	
EFIFRM-SCH-C-2	Scheme-element in Markup API is used.	8.1	46	M	
EFIFRM-SCH-C-3	The Scheme component is treated case-insensitive in Markup API.	5.1.1	16	M	
EFIFRM-SCH-C-4	The Server component is treated as case-insensitive	5.1.2	16	M	
EFIFRM-SCH-C-5	The Service component is treated as case-sensitive	5.1.3	17	M	
EFIFRM-SCH-C-6	The Parameters component is treated as case-sensitive	5.1.4	17	M	
EFIFRM-SCH-C-7	The Values of parameters are treated as case-sensitive	5.1.4	17	M	
EFIFRM-SCH-C-8	A Segment of a servers name space is NOT one of the reserved names	5.1.2, 5.5	16, 19	M	
EFIFRM-SCH-C-9	Unit identifiers are starting with the dot '.' character before the only segment.	5.1.2	16	M	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

A.2.3 APIs

Item	Function	Reference	Page	Status	Requirements
EFIFRM-API-C-1	WMLScript API	7.6	29	O	WMLScript:MCF AND EFIFRM-LIB-C-14 AND EFIFRM-LIB-C-15 AND EFIFRM-LIB-C-16
EFIFRM-API-C-2	Markup API	8	46	M	EFIFRM-API-C-5 OR EFIFRM-API-C-6
EFIFRM-API-C-3	Script API	7	22	M	EFIFRM-API-C-1 OR EFIFRM-API-C-4
EFIFRM-API-C-4	ECMAScript API	7.7	39	O	ESMP:MCF AND EFIFRM-ES-C-1 AND EFIFRM-ES-C-2 AND EFIFRM-ES-C-3
EFIFRM-API-C-5	Markup API using XHTML Mobile Profile	8	46	O	XHTMLMP:MCF
EFIFRM-API-C-6	Markup API using WML	8	46	O	WML1:MCF OR WML2:MCF

A.2.4 WMLScript API

NOTE: Item 1 to 13 are collectively required in item 16.

Item	Function	Reference	Page	Status	Requirements
~~EFIFRM-LIB-C-1~~	~~set()~~	~~5.1.1~~	~~16~~	~~O~~	
~~EFIFRM-LIB-C-2~~	~~get()~~	~~7.6.1.2~~	~~30~~	~~O~~	
~~EFIFRM-LIB-C-3~~	~~getFirstName()~~	~~7.6.1.3~~	~~31~~	~~O~~	
~~EFIFRM-LIB-C-4~~	~~getNextName()~~	~~7.6.1.4~~	~~32~~	~~O~~	
~~EFIFRM-LIB-C-5~~	~~getAllAttributes()~~	~~7.6.2.1~~	~~32~~	~~O~~	
~~EFIFRM-LIB-C-6~~	~~getAttribute()~~	~~7.6.2.2~~	~~33~~	~~O~~	
~~EFIFRM-LIB-C-7~~	~~getClassProperty()~~	~~7.6.3.1~~	~~33~~	~~O~~	
~~EFIFRM-LIB-C-8~~	~~getUnits()~~	~~7.6.4.1~~	~~34~~	~~O~~	
~~EFIFRM-LIB-C-9~~	~~query()~~	~~7.6.4.2~~	~~35~~	~~O~~	
~~EFIFRM-LIB-C-10~~	~~invoke()~~	~~7.6.5.1~~	~~36~~	~~O~~	
~~EFIFRM-LIB-C-11~~	~~call()~~	~~7.6.5.2~~	~~36~~	~~O~~	
~~EFIFRM-LIB-C-12~~	~~status()~~	~~7.6.5.3~~	~~37~~	~~O~~	
~~EFIFRM-LIB-C-13~~	~~control()~~	~~7.6.5.4~~	~~37~~	~~O~~	
EFIFRM-LIB-C-14	Interpreting library ID	7.6	29	O	
EFIFRM-LIB-C-15	Support for Containers	7.6.1	29	O	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

EFIFRM-LIB-C-16	EFI function calls	7.6.2, 7.6.3, 7.6.4, 7.6.5	32, 33, 34, 35	O	

A.2.5 ECMAScript API

Item	Function	Reference	Page	Status	Requirements
EFIFRM-ES-C-1	EfiError native error type	7.7	39	O	
EFIFRM-ES-C-2	Support for name/value collections	7.7.1	39	O	
EFIFRM-ES-C-3	Efi Object including all properties and methods	7.7	39	O	

A.2.6 Attributes, Properties

Item	Function		Reference	Page	Status	Requirements
EFIFRM-ATTR-C-1	Broker	VersionMajor	7.2	22	M	
EFIFRM-ATTR-C-2		VersionMinor	7.2	22	M	
EFIFRM-ATTR-C-3		Manufacturer	7.2	22	O	
EFIFRM-ATTR-C-4		ManVersionMajor	7.2	22	O	
EFIFRM-ATTR-C-5		ManVersionMinor	7.2	22	O	
EFIFRM-ATTR-C-6	Unit or	VersionMajor	7.2	22	M	
EFIFRM-ATTR-C-7	Class	VersionMinor	7.2	22	M	
EFIFRM-ATTR-C-8	Agent	Name	7.2	22	M	
EFIFRM-ATTR-C-9		Manufacturer	7.2	22	M	
EFIFRM-ATTR-C-10		ManVersionMajor	7.2	22	O	
EFIFRM-ATTR-C-11		ManVersionMinor	7.2	22	O	
EFIFRM-ATTR-C-12	Class	MinVersionMajor	7.3	23	M	
EFIFRM-ATTR-C-13	Properties	MinVersionMinor	7.3	23	M	
EFIFRM-ATTR-C-14		MaxVersionMajor	7.3	23	M	
EFIFRM-ATTR-C-15		MaxVersionMinor	7.3	23	M	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document.

A.2.7 Local Server

Item	Function	Reference	Page	Status	Requirements
EFIFRM-LSRV-C-1	The request initiates the service that composes and returns a well-formed document to the browser.	8	46	M	
EFIFRM-LSRV-C-2	The EF Broker provides a no-name service.	8.2.1	47	O	EFIFRM-LSRV-C-3
EFIFRM-LSRV-C-3	The content of "efi:///" at least present the list of available servers in a readable format with the content equivalent to the getUnits() function.	8.2.1	47	O	
EFIFRM-LSRV-C-4	The EF Class Agent provides a no-name service.	8.2.1	47	O	EFIFRM-LSRV-C-5
EFIFRM-LSRV-C-5	The EF Class Agent provides at least the name of the server within the generated document.	8.2.2	47	O	
EFIFRM-LSRV-C-6	The EF Unit provides a no-name service.	8.2.2	47	O	EFIFRM-LSRV-C-7
EFIFRM-LSRV-C-7	The EF Unit provides at least the name of the server in the generated document	8.2.3	47	O	
EFIFRM-LSRV-C-8	Cache is NOT used for EFI services with Markup API	8.7	50	M	

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description
WAP-231-EFI-20010511-a	11 May 2001	Initial
WAP-231-EFI-20011217-a, OMA V1_0	17 Dec 2001	Addition of XHTML API
OMA-WAP-EFI-V1_1-20110315-A	15 Mar 2011	Status changed to Approved by TP: OMA-TP-2011-0082-INP_EFI_V1_1_ERP_for_Final_Approval

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.
Used with the permission of the Open Mobile Alliance Ltd. under the terms as stated in this document. [OMA-Template-EnablerTestSpec-20110101-I]