EFI Class Definition Process

Candidate Version 1.1 – 9 Jun 2004

Open Mobile Alliance
OMA-WAP EFICDP –V1_1-20040609-C

Continues the Technical Activities
Originated in the WAP Forum

Contents

1. Scope

Wireless Application Protocol (WAP), as part of Open Mobile Alliance (OMA), is a result of continuous work to define an industry wide specification for developing applications that operate over wireless communication networks. The scope for OMA is to define a set of specifications to be used by service applications. The wireless market is growing very quickly and reaching new customers and providing new services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation, and fast/flexible service creation, WAP defines a set of protocols in transport, session and application layers. For additional information on the WAP architecture, 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 terminal or by allowing new types of devices to be connected to the 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 mobile 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.

This document shall enable the reader to produce a Class Specification even though he does not intend to introduce it as a well-known standard specification within OMA but to fulfil requirements for interoperability to that extend needed in the special case.

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/

[EFIFRM] "External Functionality Interface Framework", Open Mobile Alliance™, OMA-WAP-EFI-v1_1. http://www.openmobilealliance.org/

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

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

[WAE] "Wireless Application Environment Version 2.1", Open Mobile Alliance™. OMA-WAP-WAESpec-V2_1, http://www.openmobilealliance.org/

[OMNA] "WAP OMNA Process Document", WAP Forum™, WAP-212-OMNAProcess, http://www.openmobilealliance.org/

2.2 Informative References

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

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

[WML] "Wireless Markup Language, Version 1.3", WAP Forum™, WAP-191-WML.

"Wireless Markup Language, Version 2.0", WAP Forum™, WAP-238-WML.

http://www.openmobilealliance.org/

[WMLS] "WMLScript Language Specification", WAP Forum™, WAP-193-WMLScript,

http://www.openmobilealliance.org/

[XHTMLMP] "XHTML Mobile Profile 1.1", Open Mobile Alliance™. OMA-WAP-XHTMLMP-V1_1.

http://www.openmobilealliance.org/

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”, “Introduction” and “Design Guidelines” are normative.

AI	Application Interface (more general then API)
API	Application Programming Interface
EF	External Functionality
EFI	External Functionality Interface
MAG	Mobile Applications Group
OMA	Open Mobile Alliance
OMNA	Open Mobile Alliance Interim Naming Authority
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WML	Wireless Markup Language
XHTML	eXtensible HyperText Markup Language

”, 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.

External Functionality Functionality that is external to the Wireless Application Environment and internal or external to the mobile device.

Gateway WAP gateway as specified in [WAPARCH].

Implementation The software and hardware that is used in the particular terminal to implement the functionality.

Input paper Input documents are typically used to communicate ideas, technical critique, requirements and other information to OMA. Contributions may be submitted to OMA using the process outlined on the web site of OMA.

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.

Script Scripting language specified for programming a mobile device. [WMLS, ESMP]

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, -Class Agent or -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]

3.3 Abbreviations

AI	Application Interface (more general then API)
API	Application Programming Interface
EF	External Functionality
EFI	External Functionality Interface
MAG	Mobile Applications Group
OMA	Open Mobile Alliance
OMNA	Open Mobile Alliance Interim Naming Authority
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WML	Wireless Markup Language
XHTML	eXtensible HyperText Markup Language

4. Introduction

The Class Specification defines which Services can be accessed from an application and how they will behave. To come to a sufficient Class Specification as a first step the EF Entity needs to be factorised in its atomic services to enable classification of these. This can cause that more than one Class Specification is needed. Together with other EF Entities classification tries to group the Services with common functionality as much as possible, to reduce the variety of APIs for similar Services. EF Entities with common functionality may be produced as a well-known Class Specification under certain circumstances (see chapter 5)..

It is also possible for an application environment to provide proprietary classes using the EF AI to allow script and markup applications access to external functionality. This allows both application developer and device producer to concentrate the work on the specific part and to reuse all functionality available from the set of OMA Specifications (e.g. Security, WAE and protocols).

4.1 Classes

EFI defines access to common high-level abstraction of Services that spans across several entities (devices, software modules). Entities that share the same abstraction form the class (e.g. different modules for payment). Some entities are at the focus of a given EFI Class, some are marginal. The Class specifies the common part of them.

A Class is the abstract collection of functionality collected in a set of Services and Properties. Entities may belong to several EFI Classes if they provide functionality that fall into different Classes (e.g. a payment module supporting debit and credit payment methods).

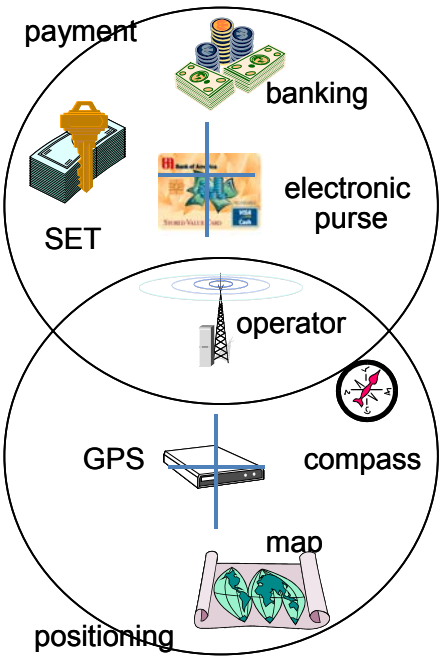

Figure 1: Possible groups of Classes

The conceptual architecture of EFI is specified in the Framework but to bring clarity for the succeeding sections it is shortly presented in Figure 2. For more detailed explanation of the figure please refer to the Framework [EFIFRM].

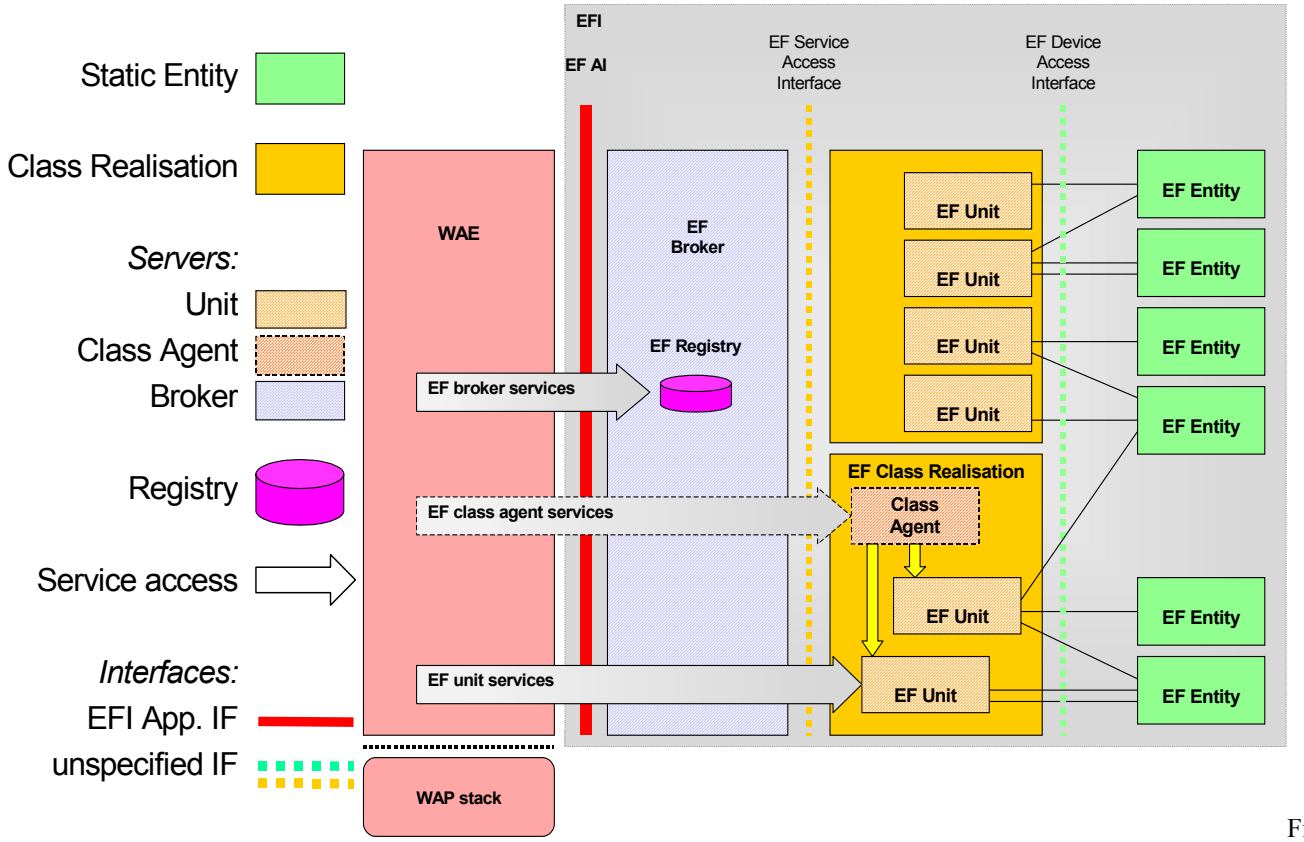

Figure 2: Conceptual Architecture of EFI

4.2 Types of Services

Within a Class Specification there may be two types of services, **mandatory services** and **optional services**. There may be also services that are implemented but not specified by the relevant Class Specification. Such proprietary services are allowed but not further discussed in this document.

4.2.1 Mandatory Services

Mandatory services must be implemented to conform to the Class Specification.

An application can assume the existence of mandatory services after it has generally proved existence of EFI and existence of the Class. Mandatory services typically provide common functionality like discovery of further services (e.g. by way of Class Agents or through some form of enumeration service).

4.2.2 Optional Services

Optional services are defined in the Class Specification to give a consistent interface to those EF Units that do provide these services, but an EF Unit may conform to the Class definition without providing these services.

There are different reasons to define services as optional. Some of them are:

- The service is specific to a part of the Class that has equivalent options to solve the requirements of this service.
- The service requires activity from a part of the Class which may be left out of a realisation for resource reasons without loosing validness of its scope.

Optional services generally require a discovery of its existence in addition to ensuring the existence of EFI in general and this Class. This can be done through any combination of conditional SCR requirements or through a direct query to according Servers (e.g. Unit, Broker or Agent). A conditional SCR requirement could be a cascade of capabilities: If the Unit supports cap. “A” it supports also cap. “B”.

4.3 Properties

Class Properties are intentionally an informational element of the realisation. The Framework defines a list of mandatory Properties to enable discovery mechanisms. A Class Specification can define additional properties if needed.

Properties belong to the realisation of the Class and not to its individual Servers it obliges the realisation which entity produces the property’s value which might be the Broker, a Class Agent or any other peace of implementation not further defined here. This allows creation of the values of Properties like “the best choice” for a special request. Properties may be derived on time of request covering the actual situation of the realisation (e.g. absence of some Servers).

4.4 Attributes

Attributes are scoped to show information like the status of a given Server. Each Server has a set of attributes that can be extracted by the application. The Framework defines an initial set of attributes to be present for all Servers of a given type.

A Class can define additional Attributes like Quality or Availability of results. A Positioning Class could define a Unit Attribute called *Available* and this would be testable before a position is requested from the application.

5. Class Specification Process

It is important to point out that any company, organisation, or person can create Class Specifications. You need not to be affiliated with OMA, or if you are, part of any particular working group to create a Class Specification.

The type of Class Specification, either well known or registered, determines the level of interaction with OMA. The following flowchart shows how to decide between the two types of Class Specifications.

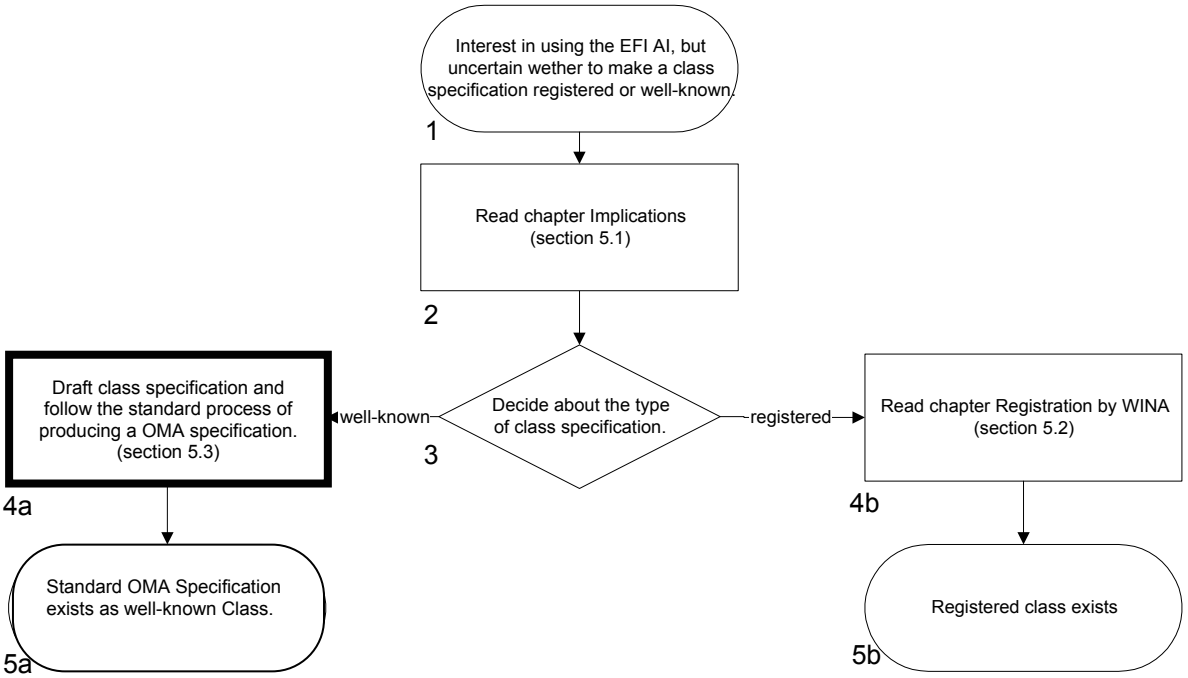

Figure 3: Class Specification process

If the issuer decides to propose the Class Specification as a well-known class later on he has to follow the process again.

5.1 Implications

The different types can be summarised with the following attributes:

Registered

- Vendor specific namespace can be protected by OMNA
- The owner can make changes to the Class Specification without control of OMA
- The owner maintains the Class Specification themselves
- It is also under the owner's control to provide any means for interoperability

The Class Specification written according to the above is chosen to be under control of the issuer only or to be proposed as a well-known Class Specification. If a class fails to get support, it can be registered as a registered class and left in that state.

When the Class Specification is in the state registered, it needs to have an assigned owner. The owner could be a person, a company or an organisation.

Registered Class Specifications are outside of OMA's purview so conformance testing needs to be driven through the owner if wanted.

Well-known

- Uses the standard name space publicly available through OMA web sites
- Class Specifications are issued through standard OMA process
- Reliability and interoperability is provided by OMA means like test assertions and conformance statements

Class Specifications are the responsibility of OMA wherein MAG cares about it. If there is an appropriate committee OMA can assign the responsibility to it. MAG is understood as a placeholder for the charter-documents of any subgroup appointed by MAG.

5.2 Registering EF Class Names to OMNA

Class names for both registered and well-known Class Specifications must be registered with OMNA. The process for registering names can be found on the web site of [OMNA]. Registering the class name with OMNA ensures there is a name space for the services in the Class Specification and to avoid name clashes with other EF Classes.

Be aware that reserved words defined in [EFIFRM] MUST NOT be used for EF Class names.

5.3 Standard OMA Specification Process

The OMA Organisation and Process document governs the development of specifications. The process document can be found on http://www.openmobilealliance.org. In addition to this, registering class specifications with OMNA has to be done, which should be initiated as soon as the class Specification is in a stable state.

5.4 Maintenance of existing Class Specification

Changes to an existing well-known Class Specification MUST be handled according to the actual OMA specification process. If there is an additional owner or shepherd outside of OMA he SHOULD be informed of the changes.

Changes to a registered Class Specification can be done without notification to OMNA and any other body within OMA.

6. Design Requirements

Designing an EF Class can be done by any group or individual within or outside of OMA. In order to achieve rapid standardisation of a Class, a proposal for a new Class Specification SHOULD address the following requirements from the very beginning, even if the specification will stay as a registered Class. If the specification will stay as a well-known class the following statements MUST be fulfilled.

There MUST be an owner or shepherd for the Class Specification. By default registered Class Specifications are owned by the issuer which made the registration request. The well-known Class Specifications are owned by MAG.

6.1 EF Class

Reasons for defining an EF Class (provide a standardised interface from WAE to EF Entity):

- There MUST be an interest from more than one company to achieve interoperability.
- MUST rely on well-known functionality or capabilities which are intended to be shared with other parties.

If one or both of these requirements don't fit, the vendor-specific registration MUST be taken (cf. chap 5.2).

6.2 Use Cases

Use cases MUST be built for a class.

The purpose of the use cases is to:

- understand functionality of the service / class
- ensure correctness of Class Specification
- enable creation of test assertions
- find mostly fully defined service environment regarding architectural aspects

Use cases MUST have the following parts:

- Description from a user point of view
- Description of services from an application developer's point of view
- User requirements on services
- Sequence of information flow

6.3 Accessibility

Services provided by EFI MUST appear to WAE to be under exclusive control of the application. This does not preclude services from being unavailable due to lack of available resources or other similar reasons.

The EFI service MAY use the physical device or part of the software that is shared by other services or by other components of the client terminal. A WAE application using EF AI assumes that it is the only controller of the EF unit.

6.4 High Level Interface

Class Specifications SHOULD support access at a high-level to functionality, by defining an application-independent interface. The EF Services SHOULD abstract the functionality at a high level. Consequently, the EF Service SHOULD NOT provide functionality that is relevant to the:

- physical interfaces used to interact with any extension or device
- internal structure of any add-on device or functionality.

The service SHOULD attempt to abstract the functionality of a given application area, regardless of its implementation or structure.

6.5 Types of APIs

6.5.1 Script API

It is important to understand the temporal relationship between an application and a service. All services MUST be perceived by an application as asynchronous, i.e. an instance of a service is invoked by the application and then the application and service continue in parallel. An application MAY start several instances of the same or different services.

When the service is invoked, EFI analyses whether the service 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.

6.5.2 Markup API

The Markup API behaves differently from the Script API. Services accessed through Markup API MUST behave like a "web server" returning a document in the Markup language that is accepted by the User Agent, as defined by [WAE], as the only type of response.

The service MAY return the document anytime during it's ongoing work or immediately after termination. This behaviour MUST be fully defined in the Class Specification.

The specification MUST take into account that the interaction between the browser and the service may be interrupted or overridden (even unintentionally) at any time.

6.6 Security and Privacy

The Class Specification MUST define the proper security and privacy requirements.

EFI currently assumes that a document or script is NOT trusted. A Class Specification MUST state the means and mechanisms that apply to achieve this requirement.

6.7 Parameters and Return Values

The services may take parameters as input and return result values, each of which needs to be defined. The definition must take into account the type, the range of valid values, and the expected semantics.

If the Markup API is used, the service will return a document to be displayed by WAE. As such, there will be no precise definition of what is returned by the service. The document returned should satisfy the semantics of the service.

7. Class Specification Requirements

Designing an EF Class can be done by any group or individual within or outside of OMA. In order to achieve rapid standardisation of a Class, a proposal for a new Class Specification SHOULD address the following requirements from the very beginning, even if the specification will stay as a registered Class. If the specification will stay as a well-known class the following statements MUST be fulfilled.

7.1 General Content

An EFI Class Specification MUST provide the following information.

7.1.1 Name

The name of the class is case-insensitive. It MUST be unique within the scope of EFI and MUST NOT be one of the reserved words defined in the EFI Framework [EFIFRM].

7.1.2 Scope of the class

Explanation of what constitutes the scope of applications that are supported with the given class.

7.1.3 Mandatory services of Class Agent

Definition of any services which are mandatory for the Class Agent, i.e. they MUST be implemented by the class realisation.

7.1.4 Optional services of Class Agent

A definition of the services that are optional for the Class Agent. The class realisation still fulfils the specification if optional services are not implemented. However, proprietary class services MUST NOT have the same name as optional Class Agent services. If an optional service (i.e. SCR item) is being implemented, all SCR items that are required by this item MUST also be implemented.

7.1.5 Mandatory Unit services

A definition of mandatory services that MUST be implemented for every EF Unit that belongs to the given class.

7.1.6 Optional Unit services

A definition of the services that are optional for the EF Unit. The class realisation still fulfils the specification if optional services are not implemented. However, proprietary class services MUST NOT have the same name as an optional EF Unit service. If an optional service (i.e. SCR item) is being implemented, all SCR items that are required by this item MUST also be implemented.

7.1.7 Static Conformance Requirement

The requirements which have to be fulfilled to be conformant with the Class Specification. The static conformance requirements must be testable statements and MUST follow the guidelines specified in [IOPProc].

7.1.8 Reserved names

The authority for verification is OMA. If the EF Class name is requested for registered part it MUST have the appropriate notation as stated in [EFIFRM]. Note that this is "`vnd.`" in this moment.

7.1.9 Dependencies

If the Class has a dependency on the presence of an optional OMA functionality or enablers then this dependency MUST be stated.

7.2 Service definition

The service definition specifies all the necessary details of the service. This MUST include at least the following components stated below. Note that each service is defined for a particular API. If similar services are accessed through different APIs, they MUST be treated as separate services and MUST have separate definitions.

7.2.1 Name of the Service

Names are case-sensitive. The name MUST be unique within the class. Similar services that are accessible through different API MUST have different names.

7.2.2 Description

Explanation of the functionality provided by the service and the semantics of the service.

7.2.3 API-Type

The API through which the service is accessible (e.g. Markup or Script).

7.2.4 Static Conformance Requirements

Statement of whether the service is mandatory or optional. If an optional service (i.e. SCR item) is being implemented, all SCR items that are required by this service MUST also be implemented.

7.2.5 Usage Restrictions

Some services are used to access secure information. For instance, a service might require user consent before accessing the user's personal information. If so, privacy or security requirements MUST be defined.

7.2.6 Input parameters

The description for each input parameter MUST have all following statements:

- name
- semantics
- expected type
- default value if applicable
- whether it is optional
- range

7.2.7 Return values

Description for each return value MUST have all following statements:

- name
- semantics
- expected type
- whether it is optional
- range
- error values

7.2.8 Behaviour with the Script `Control()` function

For services defined using the Script API the behaviour of the Script Control() function MUST be specified. This MUST cover at least the response to the control commands defined in [EFIFRM], and may extend this list. Any additional commands MUST NOT be in conflict with standardised commands. To prevent Name and ID clashes the command MUST start with "x-", and the ID MUST be higher than 100. Note that the notion of constants prevents names from being used, except for descriptive names in the [EFIFRM].

8. Markup or Script (informative)

Due to the fact that the Markup and Script APIs behave differently in terms of context management, it has to be decided which API fits the needs of a particular service. The following questions may help to decide this:

- If the service needs to compute return values and to follow up with further functionality, then access through Script API might be useful.
- If the service needs to display rich content which can be processed by WAE, then access through Markup API might be useful.
- If the service needs somehow to have both types, it might be useful to separate the service into two services with different APIs and different descriptions.

Further information about the different APIs can be taken from the EFI Framework Specification [EFIFRM].

Note that this decision has to be made for every service. If at the starting point of a draft Class Specification it is not feasible to make a distinction between the different APIs the abstract service primitives as described in the following chapter should be used, to express which parameters does apply to the service. At a later time of draft specification the decision might be easier to make.

8.1 Abstract Service Primitives

If it can't be decided whether a service should be realised through Markup, Script or any other API which might be specified in future releases of the EFI-Framework while drafting the Class Specification, an interim format can be used to enable expressing the service without a clear decision for one API. This can help moving forward with the real work of drafting a Class Specification. Later on the drafting process might easier come to an agreement on the best API to use. Using the abstract primitives does not imply that OMA will generate the Class Specification.

Service primitives represent, in an abstract way, the logical exchange of information and control between adjacent layers. Service primitives consist of commands and their respective responses associated with the particular service provided.

Service primitives are not the same as an application programming interface (API) and are not meant to imply any specific method of implementing an API. Service primitives are an abstract mean of illustrating the services provided by the EF Entity to the WAE user agent. In particular, the service primitives and their parameters are not intended to include the information that an implementation might need to route the primitives to each implementation object, which corresponds to some abstract user or service provider instance. The mapping of these concepts to a real API and the semantics associated with a real API are the subject of the Class Specification before it can be approved.

8.1.1 Primitive Types

The primitives types defined in this paper are described in table 1:

Type	Abbreviation	Description
Request	Req	Used when the User agent is requesting a service from the EFI device
Confirm	Cnf	The EFI device uses the confirm primitive type to report that the requested activity has been completed.

Table 1: Primitive Types

8.1.2 Primitive Parameter Tables

The service primitives are defined using tables indicating which parameters are possible and how they are used with the different primitive types. If some primitive type is not possible, the column for it will be omitted.

The entries used in the primitive type columns are defined in table 2:

M	Presence of the parameter is mandatory - it MUST be present

C	Presence of the parameter is conditional depending on values of other parameters
O	Presence of the parameter is a user option - it MAY be omitted
-	The parameter is absent

Table 2: Parameter Usage Legend

8.1.3 Example of an abstract Service Primitive

An example of a primitive is:

Primitive / **Parameter**	**PrimitiveX**	
	req	***cnf***
Parameter1	M	O
Parameter2	-	C

Table 3: Example Primitive

In this example *primitiveX* has two types, *primitiveX.req* and *primitiveX.cnf*. *Parameter1* is mandatory in the request and optional in the confirmation. The presence of *Parameter2* in the confirmation is conditional on the value of *Parameter1* in the request.

Appendix A. Static Conformance Requirements (Normative)

Because this specification is not a testable specification in terms of interoperability testing, conformance tables and test suites are not applicable to it. But for approval of Class Specifications all critical MUSTs are written in capital letters according to [RFC2119].

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description
WAP-263-EFICDP-20011101-a	1 Nov 2001	The initial version of this document.

B.2 Draft/Candidate Version 1.1 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-WAP-EFICDP-V1_1	12 Nov 2003	all	New OMA template merged
	19 Apr 2004		Addressing review comments Date on front matter updated.History updated. Changed CREQ to IOPProc. Reviewed normative language and need for SCRs. None required as non-testable. Removed appendix C
Candidate Version OMA-WAP-EFICDP-V1_1	9 Jun 2004	n/a	Status changed to Candidate by TP TP ref # OMA-TP-2004-0189-EFI-V1_1-for-candidate