EFI Class Definition Process

Version 1-Nov-2001

Wireless Application Protocol
WAP-263-EFICDP-20011101-a

This document is available online in PDF format at http://www.wapforum.org/.

Known problems associated with this document are published at http://www.wapforum.org/.

Comments regarding this document can be submitted to the WAP Forum™ in the manner published at http://www.wapforum.org/.

Document History	
WAP-263-EFICDP-20011101-a	Current - Approved
WAP-263-EFICDP-20011101-p	Proposed

Contents

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 WAP ForumTM 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 services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation the WAP Forum 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].

EFI specifications are divided into three types:

1. The EFI framework that specifies the architecture and the access mechanism to make use of functionality external to WAP. This framework provides a generic environment for accessing classes of functionality. [EFIFRM]

2. The Process document that defines the process to introduce, specify and maintain class specifications.

3. Independent EFI class specifications which describe a set of services to provide a common interface between WAE and EF Entities with similar functionality.

This document specifies the process to introduce, specify and maintain class specifications.

2. References

2.1. Normative References

[WAE] "Wireless Application Environment", WAP Forum™, WAP-236-WAESPEC,
URL: http://www.wapforum.org/

[WAPPROC] “WAP Work Processes”, WAP Forum™, WAP-181-TAWP,
URL: http://www.wapforum.org/

[WAPWINA] “WAP WINA Process Document”, WAP Forum™, WAP-212-WINAProcess,
URL: http://www.wapforum.org/

[WAPSCR] “WAP Conformance Requirement Specification”, WAP Forum™, WAP-221-CREQ,
URL: http://www.wapforum.org/

[EFIFRM] “External Functionality Interface Framework”, WAP Forum™, WAP-231-EFI,
URL: http://www.wapforum.org/

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

2.2. Informative References

[WAPARCH] “WAP Architecture”, WAP Forum™, WAP-210-WAPArch,
URL: http//www.wapforum.org/

[WML] “Wireless Markup Language Specification”, WAP Forum™, WAP-238-WML,
URL: http://www.wapforum.org/

[WMLScript] “WMLScript Language Specification”, WAP Forum™, WAP-193-WMLScript,
URL: http://www.wapforum.org/

3. Terminology and Conventions

3.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED","MAY", and "OPTIONAL", as written in all-capital letters in this document are to be interpreted as described in [RFC2119].

3.2. Definitions

API

Application Programming Interface: the means available for an application to access EFI functionality from its application environment; namely Markup and WMLScript.

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 (or EF Broker)

The conceptual entity that exists between the EF Units and 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 (or EF Class Agent)

The conceptual active element that provides added functionality on the basis of EF Units of the same EF Class Realisation.

Class Realisation (or EF 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.

DID

Document Identifier; unique number for a WAP-Forum document.

EFE (see Entity)

EF

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

EFI

Acronym for External Functionality Interface. The term EFI is used as a term in itself to collectively name all the elements of EFI conceptual architecture.

Entity (or EF 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.

Implementation

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

Input paper

Input documents are typically used to communicate ideas, technical critique, requirements and other information to WAP working groups. Contributions may be submitted to the WAP Forum using the process outlined on the web site of WAP Forum. [WAPPROC]

Service (or EF Service)

The specified functionality provided by one of the servers: EF Broker, EF Class Agent or EF Unit.

SIN

Specification Information Note is a Specification-track document. This type of document is used to change specifications etc. [WAPPROC]

Terminal (see WAP Terminal)

Unit (or EF Unit)

The conceptual component that resides in or outside the WAP terminal and provides access to the EF Services on the EF Entities.

WAE

Wireless Application Environment.

WAG

Wireless Application Group.

WAP Terminal

The physical unit where the WAE executes.

WINA

WAP Interim Naming Authority

WML

Wireless Markup Language [WML]

WMLScript

Lightweight scripting language specified in WAP for programming a mobile device. [WMLScript]

4. Introduction

An EF class specification defines the functionality and interface for a class, or grouping, of services. All EF Units and Class Agents that provide the specified functionality are grouped according to the class and must conform to the class interface. Defining a common interface for a class of services ensures a level of interoperability between WAP applications and EF Entities.

4.1. Service Types

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.

1. Mandatory services must be implemented to conform to the class specification.

2. 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.

Services that an EF Unit provides beyond the mandatory and optional services are allowed but are out of the scope of the class specification (cf. chap registered classes).

5. Class Specification Process (normative)

It is important to point out that any company, organisation, or person can create class specifications. You do not have to be affiliated with the WAP Forum, 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 the WAP Forum. The following flowchart shows how to decide between the two types of class specifications.

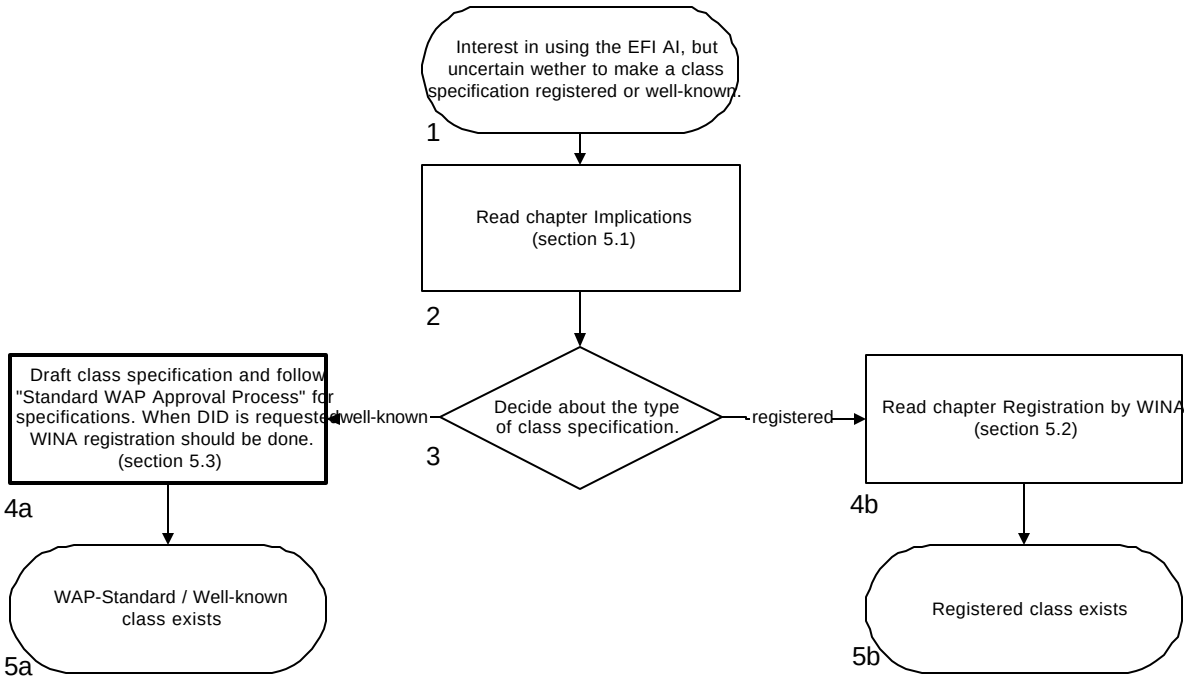

Figure 1: Class specification process

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 WINA
- The owner can make changes to the class specification without control of the WAP Forum
- 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 WAP-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 the WAP-Forum's purview conformance testing needs to be driven through the owner if wanted.

Well-known

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

Class specifications are the responsibility of WAP-Forum wherein WAG cares about it. If there is an appropriate drafting committee WAP-Forum can assign the responsibility to it. WAG is understood as a place holder for the charter-documents of any group appointed by WAG.

5.2. Standard WAP-Forum Specification Process

The actual WAP-Forum process is shown in [WAPPROC]. In addition to this the registration to WINA has to be done, which should be initiated as soon as the class Specification becomes frozen and the Specification is treated as a future feature of WAP. This is the time when the DID is requested.

5.3. Registering EF Class Names to WINA

Class names for both registered and well-known class specifications must be registered with WINA. The process for registering names can be found on WAP-Forum web site's [WAPWINA]. Registering the class name with WINA ensures there is a name space for the services in a class specification to avoid name clashes with other EF Classes.

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

5.4. Maintenance of existing Class Specification

5.4.1. Process for changing the Class Specification

Changes to an existing well-known class specification MUST be handled according to the actual WAP process. See [WAPPROC] for further details how to make changes to a Specification. If there is an additional owner or shepherd outside of the WAP-Forum he SHOULD be informed of the changes.

Changes to a registered class specification can be done without notification to WINA and any other body within the WAP-Forum.

6. Design Requirements (normative)

Designing a EF Class can be done by any group or individual within or outside the WAP Forum. 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 their issuer which made the registration request. The well-known class specifications are owned by the drafting committee if it still exists otherwise by WAG.

6.1. EF Class

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

- There should be an interest from more than one company to achieve interoperability.
- MUST rely on well-known functionality or capabilities which cannot 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 Services

6.5.1. Services using the WMLScript 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. Services using the Markup API

The Markup API behaves differently from the WMLScript API. Services accessed through Markup API MUST behave like an “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 therefore MUST define the proper security and privacy requirements.

EFI currently assumes that a deck 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 (normative)

Designing a EF Class can be done by any group or individual within or outside the WAP Forum. 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

EFI Class specification MUST contain at least the following parts.

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.

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.

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 [WAPSCR].

7.1.8. Reserved names

The authority for verification is the WAP Forum . 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 WAP feature 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. The 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 WMLScript).

7.2.4. Static Conformance Requirements

Statement of whether the service is mandatory or optional.

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 EFI.Control() function

For services defined using the WMLScript API the behaviour of the EFI.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 so that no names are used, except for descriptive names in the [EFIFRM].

8. Design guidelines (informative)

This chapter should help the author writing an appropriate class specification from beginning on. It is intended to provide further clarification of the requirements and reflect the combined experience of the EFI authors.

8.1. Template

A well-known class specification is treated as an standard WAP specification from the beginning. The appropriate template can be found on WAP-Forum web site. It is recommended to write registered class specification also with the same chapters which can be copied from other Class-Specifications or the following Questionnaire. If it is intended to provide an input paper the same Questionnaire can be used.

8.2. Questionnaire

The following is a suggested structure for describing a class of services for EFI. It is not mandatory to fill out every part, but can be used to create a thorough input paper.

1 Introduction and overview

This section gives an overview of the class of devices to that are covered by the definition.

1.1 Class definition

1.1.1 Intention of the class
- What is the reason for this class?
- Why are the services not kept in another existing class?

1.1.2 Chart for understanding the class or service sets
- Which entities are in the scope of this proposal?
- Which functionality will be enabled with this class?

1.1.3 Statement of Security
– Which parts of the class will have sensitive security issues regarding EFI Framework. [EFIFRM]

1.2 Explanation of interoperability

1.2.1 Architectural impact
Class specifications SHOULD NOT impact with the WAP architecture or other WAP specifications. If there is a strong need for change or addition, this has to be stated in the input paper, to start discussions on the issue.
- Does any part of the class have an impact on the WAP architecture as described in [WAPARCH]?

1.2.2 Class Static Conformance Requirements
- At minimum there has to be a list of all services marked as mandatory or optional.

1.2.3 Affected groups inside the WAP-Forum

1.2.4 Affected groups outside the WAP-Forum

2 **Description of services**
This section describes one service of a class specification. This chapter should be repeated for each service. It is recommended that mandatory and optional services not be mixed, to avoid confusion of implementers. Grouping services for Markup and WMLScript API is recommended.
The following parts should be touched:

2.1 Service Primitive / Function name

2.2 Access mechanism (Markup / WMLScript).

2.3 Description of semantics and syntax. If WMLScript is the given API the behaviour of the EFI.Control() MUST be clarified.

2.4 Definition for each of the parameters, return values and exceptions

2.4.1.1 Range

2.4.1.2 Valid values

2.4.1.3 Semantics and meaning

2.5 Behaviour in respect to the user (privacy)
- Must the user be asked for acceptance of executing the service in a certain manner?

2.6 Enforcement of security.
– Does the service need any particular security mechanisms like encryption or user authentication?
- Take into consideration that security mechanisms are out of the scope of the EFI Framework. If an application security is needed, it has to be specified inside the class specification.

2.7 Static Conformance Requirements table entry for all parameters, return values and exceptions.
(may be collected at the end of document)

3 Implementation notes

This section specifies implementation notes and any issues specific to particular devices that implements the functionality of the class.

Additionally service names may be stated hereunder as reserved for future use.

8.3. Markup or WMLScript

Due to the fact that the Markup and WMLScript 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 WMLScript 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.

8.4. Abstract Service Primitives

If it can't be decided whether a service should be realised through Markup, WMLScript 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 the WAP-Forum 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.

8.4.1. 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

8.4.2. 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

8.4.3. 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 RFC [RFC2119] see chap. 3.1.

Appendix B. Change History (Informative)

Type of Change	Date	Section	Description
Class 0	1-Nov-2001		The initial version of this document.