EFI Test Class

Candidate Version 1.1 – 9 Jun 2004

Open Mobile Alliance

OMA-WAP-EFITEST-V1_1-20040609-C

Continues the Technical Activities
Originated in the WAP Forum

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 Open Mobile Alliance[M] 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 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].

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 EFI requirements. The External Functionality can be built-in or connected to a mobile terminal. This connection can be permanent or temporary.

This document defines the EFI test class. The basic idea of this test class is to allow the testing and certification of the EFI framework with a minimum set of functions. These functions should be accessible from WAP applications during the certification tests.

The functions of the test class and the corresponding attributes and properties (the test class) should be implemented in a mobile terminal undergoing EFI interoperability testing. This will allow the development of an EFI interoperability test suite that is the same for every mobile terminal under test.

2. References

2.1 Normative References

none

2.2 Informative References

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

[EFI] "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

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

3. Terminology and Conventions

3.1 Conventions

This is an informative document, which is not intended to provide testable requirements to implementations.

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 mobile 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.

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

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

Mobile Terminal The physical unit where the WAE executes.

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.

3.3 Abbreviations

AI	Application Interface.
API	Application Programming Interface
EF	External Functionality
EFI	External Functionality Interface
EFE	External Functionality Entity
OMA	Open Mobile Alliance
OMNA	Open Mobile Interim Naming Authority
TOG	The Open Group (the company responsible for WAP interoperability testing)
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WMLS	Wireless Markup Language Script

4. Introduction

The test class should help to test the implementation of the EFI framework within different terminals. If the functions of this test class are implemented in every EFI terminal, the EFI framework implementation of all terminals can be fully tested with the same set of test scripts.

The definition of the test class contains the following:

- Definition of fixed attributes and properties
- Services for Script API (2 services) and Markup API (1 service)

This is enough to basically test all the functions that are defined in the EFI framework.

4.1 Test Class Accessibility

The test class need only be accessible during the test phase. It is not necessary for the test class to be a fixed part of every terminal. If a terminal is able to dynamically load classes, it can load the test class only for the testing purposes.

If the test class is a fixed part of the terminal, there is a risk of "denial of service" because every test class contains an asynchronous service that can be started and continues running (including usage of hardware resources) until a timeout occurs. Therefore, it is recommended that a permanently installed EFI test class only be accessible in a terminal in "test mode" (e.g. a switch must be activated or the input of a key sequence should switch the terminal into a test mode).

4.2 Test Class Implementation

Every implementation of the test class (class realisation) should contain at least three units: two visible units and one invisible unit. With the two visible units, the ability to call functions from different units can be tested. With the invisible unit, the ability to call functions from an invisible unit can be tested.

Because the name of an invisible unit is not known, the name of the invisible unit must be clearly stated to TOG before the tests can be done.

5. Script Test Services

5.1 Script_CopyContainer

NOTE: This test function is only valid for the WMLScript API. Implementations that only support the ECMAScript API for EFI are not required to implement this test service.

SERVICE:	*test / Script_CopyContainer*
DESCRIPTION:	This service must copy all variables of the input container into an output container. It is not necessary to preserve the sequential order of the different name/value parameter pairs from the input container.
PARAMETERS:	Zero or more name/value parameter pairs that should be copied into the output container.
RETURN VALUE:	If the function executes correctly, it returns the output container that contains the same variables (name/value pairs) as the input container. If there are errors in the input parameters, the structure of the container is invalid, or when it is impossible to copy the container (e.g. due to memory constraints), the function returns *Invalid*.
EXAMPLE	

```
// Example to call with invoke function
  var cont;  // Input container
  cont=EFI.set("", "Parameter1", "123");
  cont=EFI.set(cont, "Parameter2", "456");

  OutCont=EFI.call("test/Script_CopyContainer", 0, cont);

// OutCont is the output container
```

5.2 Script_Continual

SERVICE: *test / Script_Continual*

DESCRIPTION: This service executes until it is stopped or a maximum timeout occurs. During runtime it increments an internal number periodically. If the service stops, it returns this internal number.

This service produces some output to the MMI during runtime, so it is possible to monitor if this service is still running or not. On a terminal with a display, it shows the representation of the incremented internal number. On a terminal without a display it should "beep" periodically.

This service should have a maximum timeout (MAX_TIMEOUT) of 5 minutes (300 seconds). The input parameter "timeout" of the EFI framework functions "call" and "invoke" should be set to 0 or any other value lower than MAX_TIMEOUT. The function's internal timeout MAX_TIMEOUT will be used to prevent "denial of service" attacks.

PARAMETERS: None

RETURN VALUE: A container with one parameter "number". This parameter represents the internal number that will be incremented from the service periodically.

If there are errors the function returns Invalid if invoked using WMLScript or throws and EfiError if invoked using ECMAScript.

WMLScript EXAMPLE:

```
// Launch the service synchronously using 'call'
  OutCont=EFI.call("test/Script_Continual", 0, null);

// Or launch the service asynchronously using 'invoke'
  instance=EFI.invoke("test/Script_Continual", 300, null);
// Wait for the service to complete
  OutCont=EFI.control(instance, 4, "");

// OutCont contains the internal number
```

ECMAScript EXAMPLE:

```
// Launch the service synchronously using 'call'
  output = Efi.call("test/Script_Continual", 0, null);

// Or launch the service asynchronously using 'invoke'
  instance = Efi.invoke("test/Script_Continual", 300, null);
// Wait for the service to complete
  output = Efi.control(instance, 4);

// 'output' contains the internal number
```

5.3 Test of the EFI Script API Functions

5.3.1 Containers (WMLScript only)

The EFI WMLScript API functions `set`, `get`, `getFirstName` and `getNextName` must be implemented for the handling of the containers which will be returned from the functions `Script_CopyContainer`, `Script_Continual` and the attributes and property functions. Containers can be prepared with the set functions. The `get`, `getFirstName`, `getNextName` can be used to analyse if the output containers contain all necessary parameters.

5.3.2 Attributes and Properties

The EFI Script API functions `getAllAttributes`, `getAttributes` and `getClassProperty` must be implemented in the terminal. They should return all the attributes and properties that are defined within this document for the test class.

5.3.3 Service discovery

The functions `getUnits` and `query` must be implemented in the terminal. They should return information about this test class and its functions. The functions `getUnits` and `query` can gather information about the existence of the functions from the test class. This allows the service discovery to be tested completely.

5.3.4 Services

The functions `invoke`, `status` and `control` must be implemented in the terminal to start the asynchronous service `Script_Continual`. The function `call` must be implemented to start the synchronous service `Script_CopyContainer`.

5.3.5 Restrictions of terminals

The test class contains both synchronous and asynchronous function calls. There might be terminals that cannot support asynchronous function calls. To fully support the test class, these terminals must return “Invalid” when asynchronous functions are called using WMLScript. If invoked using ECMAScript, the EfiError exception must be thrown.

6. Markup API Test Functions

To test the Markup API, a service should return a document that shows all the incoming parameters on the display. Because it is not possible to return result parameters in a container, the EFI_GetContent service must use the values of the incoming parameters to initialise variables with the same name and value in the current browser context as described in section 8.3.2 of [EFI].

6.1 EFI_GetContent

SERVICE:	*test / EFI_GetContent*
DESCRIPTION:	This service must return content that can be rendered by the resident browser on the terminal. Additionally this service must use all incoming name value pairs to set global variables in the current browser context.
PARAMETERS:	name / value pairs
RETURN VALUE:	If successful, a document should be displayed in the resident browser. The returned content must display the name and the value of the input parameters. Also for every name / value pair a global variable ”name” with the value “value” must be initialised. In case of error, one of the defined error codes will be returned

EXAMPLE:

```
<!-- This example uses WML syntax -->
<wml>
    <card name="test">
        <do type="accept" label="Start">
            <go href="efi://test/EFI_GetContent?
                                parameter1=123&amp;
                                parameter2=345"/>
        </do>

        <p>
            Select 'Start' to start test
        </p>
    </card>
</wml>
```

6.2 Test of the Markup API Functions

6.2.1 Parameters

An EFI test sequence is able to check if the Markup API can handle incoming parameters because the service EFI_GetContent must initialise a global variable for each input parameter. The tester can then verify these global variables are initialised correctly.

6.2.2 Service discovery

The EFI functions getUnits and query can gather information about the existence of the functions from the test class. So the service discovery can be tested completely.

7. Attributes and Properties

To test the attribute functions of the EFI framework, the following attributes must be set up for the implementation of the test class.

These values should be fixed, so it is possible to test the proper handling of the attribute functions. In the following tables, the values are given for the test class.

7.1 Attributes of the Unit

To test the attribute functions of the EFI framework, the EF Unit attributes must be assigned the following values in the test class implementation:

Attribute	Description	value
VersionMajor	Major version number of the Unit or the Class Agent.	“1”
VersionMinor	Minor version number of the Unit or the Class Agent.	“1”
Name	Descriptive name of the Unit	“Test”
Manufacturer	Descriptive name of the manufacturer of the Unit, may include the make and the model	“EFI test class”

7.2 Properties of the Class Realisation

The Broker must assign the following values to its attributes in the test class implementation:

Property	Description	Default value
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.	“1”
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.	“1”
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.	“1”
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.	“1”

7.3 Attributes of the Broker

Because the broker is not a fixed part of the test class, there is no need to define default values within this specification.

Appendix A. Static Conformance Requirements

No static conformance requirements are needed for the test class. It is the purpose of the test class to test the EFI implementation of different terminals.

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description

B.2 Draft/Candidate Version 1.1 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-WAP-EFITEST-V1_1	1 Nov 2001	n/a	Initial (WAP-272-EFITEST)
	12 Nov 2003	all	New OMA template merged
	19 Apr 2004		Update to address consistency comments Date on front matter updated.History updated. CREQ reference changed to IOPProc
Candidate Version OMA-WAP-EFITEST-V1_1	9 Jun 2004	n/a	Status changed to Candidate by TP TP ref # OMA-TP-2004-0189-EFI-V1_1-for-candidate