Specification Information Note

WAP-231_100-EFI-20011001-a

Version 18-October-2001

for

Wireless Application Protocol
WAP-231-EFI-20010511-a
EFI Framework
Version 11-May-2001

Contents

1. Scope

This document provides changes and corrections to the following document files:

- WAP-231-EFI-20010511-a

It includes changes from the following change requests:

- CR-WAP-231-EFI-MOT-20010928-fixSCRs
- CR-WAP-231-EFI-MOT-20010928-WMLAPI

2. Notation

In the subsections describing the changes new text is underlined. Removed text has ~~strikethrough~~ marks. The presented text is copied from the specification. Text that is not presented is not affected at all. The change descriptions may also include editor's notes similar to the one below. The notes are not part of the actual changes and must not be included in the changed text.

Editor's note: Framed notes like these only clarify where and how the changes shall be applied.

3. Fix SCR Table

3.1 Change Classification

Class 3 – Clerical Corrections

3.2 Change Summary

Fix SCR Table in the EFI Framework specification in response to the email "WAP 2.0 SCR review comments" sent Wednesday, Sept 12th to the WAP-EDITORS mailing list by Risto Kurki.

3.3 Change Description

References will be corrected when the changes are made to the EFI spec.

A.1. Script Encoder Options

Item	Function	Reference	Page	Status	Requirements
~~EFIFRM-S~~EFIFRM-LIB-S-1	Encoding of set()	7.2.1	23	M	
~~EFIFRM-S~~EFIFRM-LIB-S-2	Encoding of get()	7.2.2	24	M	
~~EFIFRM-S~~EFIFRM-LIB-S-3	Encoding of getFirstName()	7.2.3	24	M	
~~EFIFRM-S~~EFIFRM-LIB-S-4	Encoding of getNext Name()	7.2.4	25	M	
~~EFIFRM-S~~EFIFRM-LIB-S-5	Encoding of getAllAttributes()	7.3.1	26	M	
~~EFIFRM-S~~EFIFRM-LIB-S-6	Encoding of getAttribute()	7.3.2	27	M	
~~EFIFRM-S~~EFIFRM-LIB-S-7	Encoding of getClassProperty()	7.4.1	28	M	
~~EFIFRM-S~~EFIFRM-LIB-S-8	Encoding of getUnits()	7.5.1	29	M	
~~EFIFRM-S~~EFIFRM-LIB-S-9	Encoding of query()	7.5.2	30	M	
~~EFIFRM-S~~EFIFRM-LIB-S-10	Encoding of invoke()	7.6.1	35	M	
~~EFIFRM-S~~EFIFRM-LIB-S-11	Encoding of call()	7.6.2	35	M	
~~EFIFRM-S~~EFIFRM-LIB-S-12	Encoding of status()	7.6.3	36	M	
~~EFIFRM-S~~EFIFRM-LIB-S-13	Encoding of control()	7.6.4	36	M	

~~EFIFRM-S~~EFIFRM-LIB-S-14	Encoding of library ID	7	22	M	

A.2. Client Options

A.2.1. APIs

Item	Function	Reference	Page	Status	Requirements
EFIFRM-~~API~~API-C-1	WMLS	7	22	M	WMLScript:MCF
EFIFRM-~~API~~API-C-2	WML	8	40	M	WML2:MCF OR (WML1:MCF AND XHTMLMP:MCF)

A.2.2. Broker

Item	Function	Reference	Page	Status	Requirements
EFIFRM-~~BR~~BR-C-1	The user is informed, when communicating with the EFI implementation. Either WMLS or WML API is used	7, 8	22, 40	M	
EFIFRM-~~BR~~BR-C-2	The Broker supports EFI scheme.	5	16	M	
EFIFRM-~~BR~~BR-C-3	Broker allows server both, to be visible through service discovery or not. Broker exposes the names of all visible servers.	7.5	28	M	
EFIFRM-~~BR~~BR-C-4	Every instance return its instance number which is the non-negative integer.	7.6	34	M	
EFIFRM-~~BR~~BR-C-5	If it is not possible to invoke a particular service, the application is notified by the proper return code.	7.6	34	M	
EFIFRM-~~BR~~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.	7.6	34	M	
EFIFRM-~~BR~~BR-C-7	All suspended applications, waiting for one service, are resumed and receive the identical copy of the result data.	7.6	34	M	

EFIFRM -~~BR~~BR-C-8	The instance of a service retain its result data until at all suspended applications have been resumed.	7.6	34	M	
EFIFRM -~~BR~~BR-C-9	If no application waits for the completion of the instance, the instance retain its result data and the number is not re-used.	7.6	34	M	

A.2.3. Scheme

Item	Function	Refe-rence	Page	Status	Requirements
EFIFRM -~~SCH-~~SCH-C-1	Scheme-element in WML-Script API is omitted.	7.1	22	M	
EFIFRM -~~SCH-~~SCH-C-2	Scheme-element in WML API is used.	8.1	40	M	
EFIFRM -~~SCH-~~SCH-C-3	The Scheme component is treated case-insensitive WML-API.	5.1.1	16	M	
EFIFRM -~~SCH-~~SCH-C-4	The Server component is treated as case-insensitive	5.1.2	16	M	
EFIFRM -~~SCH-~~SCH-C-5	The Service component is treated as case-sensitive	5.1.3	17	M	
EFIFRM -~~SCH-~~SCH-C-6	The Parameters component is treated as case-sensitive	5.1.4	18	M	
EFIFRM -~~SCH-~~SCH-C-7	The Values of parameters are treated as case-sensitive	5.1.4	18	M	
EFIFRM -~~SCH-~~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-~~SCH-C-9	Unit identifiers are starting with the dot '.' character before the only segment.	5.1.2	16	M	

A.2.4. EFI-Library Functions

Item	Function	Refe-rence	Page	Status	Requirements
EFIFRM -~~LIB-~~LIB-C-1	set()	5.1.1	16	M	
EFIFRM -~~LIB-~~LIB-C-2	get()	7.2.2	24	M	
EFIFRM -~~LIB-~~LIB-C-3	getFirstName()	7.2.3	24	M	
EFIFRM -~~LIB-~~LIB-C-4	getNextName()	7.2.4	25	M	

EFIFRM-~~LIB-~~LIB-C-5	getAllAttributes()	7.3.1	26	M	
EFIFRM-~~LIB-~~LIB-C-6	getAttribute()	7.3.2	27	M	
EFIFRM-~~LIB-~~LIB-C-7	getClassProperty()	7.4.1	28	M	
EFIFRM-~~LIB-~~LIB-C-8	getUnits()	7.5.1	29	M	
EFIFRM-~~LIB-~~LIB-C-9	query()	7.5.2	30	M	
EFIFRM-~~LIB-~~LIB-C-10	invoke()	7.6.1	35	M	
EFIFRM-~~LIB-~~LIB-C-11	call()	7.6.2	35	M	
EFIFRM-~~LIB-~~LIB-C-12	status()	7.6.3	36	M	
EFIFRM-~~LIB-~~LIB-C-13	control()	7.6.4	36	M	
EFIFRM-~~LIB-~~LIB-C-14	Interpreting library ID	7	22	M	

A.2.5. Attributes, Properties

Item	**Function**		**Refe-rence**	**Page**	**Status**	**Requirements**
EFIFRM-~~ATTR-~~ATTR-C-1	Broker	VersionMajor	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-2		VersionMinor	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-3		Manufacturer	7.3	26	O	
EFIFRM-~~ATTR-~~ATTR-C-4		ManVersionMajor	7.3	26	O	
EFIFRM-~~ATTR-~~ATTR-C-5		ManVersionMinor	7.3	26	O	
EFIFRM-~~ATTR-~~ATTR-C-6	Unit or	VersionMajor	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-7	Class	VersionMinor	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-8	Agent	Name	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-9		Manufacturer	7.3	26	M	
EFIFRM-~~ATTR-~~ATTR-C-10		ManVersionMajor	7.3	26	O	

EFIFRM -~~ATTR-~~ATTR-C-11	ManVersionMinor	7.3	26	O	
EFIFRM -~~ATTR-~~ATTR-C-12	MinVersionMajor	7.4	27	M	
EFIFRM -~~ATTR-~~ATTR-C-13	MinVersionMinor	7.4	27	M	
EFIFRM -~~ATTR-~~ATTR-C-14	MaxVersionMajor	7.4	27	M	
EFIFRM -~~ATTR-~~ATTR-C-15	MaxVersionMinor	7.4	27	M	

A.2.6. Local Server

Item	Function	Refe-rence	Page	Status	Requirements
EFIFRM -~~LSRV~~LSRV-C-1	The request initiates the service that composes and returns a well-formed document to the browser.	8	40	M	
EFIFRM -~~LSRV~~LSRV-C-2	The EF Broker provides a no-name service.	8.2.1	41	O	EFIFRM -LSRV-C-3~~EFIFRM-LSRV-C-3EFIFRM-LSRV-C-3~~
EFIFRM -~~LSRV~~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	41	O	
EFIFRM -~~LSRV~~LSRV-C-4	The EF Class Agent provides a no-name service.	8.2.1	41	O	EFIFRM -LSRV-C-5~~EFIFRM-LSRV-C-5EFIFRM-LSRV-C-5~~
EFIFRM -~~LSRV~~LSRV-C-5	The EF Class Agent provide at least present the name of the server within the WML-Deck.	8.2.2	41	O	
EFIFRM -~~LSRV~~LSRV-C-6	The EF Unit provide a no-name service.	8.2.2	41	O	EFIFRM -LSRV-C-7~~EFIFRM-LSRV-C-7EFIFRM-LSRV-C-7~~
EFIFRM -~~LSRV~~LSRV-C-7	The EF Unit provide at least present the name of the server. Within the WML-Deck.	8.2.3	41	O	
EFIFRM -~~LSRV~~LSRV-C-8	Cache is NOT used for EFI services with WML-API	8.7	44	M	

4. Make WML API a generic markup API

4.1 Change Classification

Class 2 – Bug Fixes

4.2 Change Summary

During the WAP Forum meeting in Anchorage, the WAP Forum Board of Directors determined that the "wml" namespace shall not be made available to content authors and that XHTML Mobile Profile is the markup language for WAP 2.0.

To conform to this decision, the WML API examples in the EFI Framework specification must be changed from WML2 syntax to WML1 syntax. Furthermore, the WML API itself must be changed to a more generic Markup API that can be used by WML or XHTML Mobile Profile.

4.3 Change Description

Editor's note: change to last paragraph in Scope

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) ~~mapped onto two major APIs that are accessible to an application: WML~~ and WMLScript. The Framework requires the mobile client to support both WMLScript and the markup language specified by the Wireless Application Environment [WAE].

Add normative reference in section 2.1

[XHTMLMP]	"XHTML Mobile Profile", WAP Forum, WAP-277-XHTMLMP. http://www.wapforum.org/

Add definition for XHTML MP in section 3.2

XHTML Mobile Profile A language which extends the syntax of XHTML Basic as specified in [XHTMLMP]

Add abbreviation in section 3.3

XHTML	eXtensible HyperText Markup Language

Change "WML API" to "Markup API" in section 5.3

5.3 API

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

Change section 8 from a WML API to a more generic Markup API, and change examples to use WML syntax. Changes are scattered throughout the section.

8. Markup API~~WML API~~

To use the ~~WML API~~Markup API, the mobile client supports the markup language specified by [WAE].

The ~~WML API~~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 WAP stack on different levels and by different means. However, the application uses both regular WAP stack and EFI in the same way.

The interaction model that is provided by ~~WML API~~Markup API is significantly different from the model that is provided by WMLScript 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 which 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 ~~section (card) from~~ 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 exact definition of services is provided by the Class Specification. The Framework does not define any specific service. The Framework defines only the method to invoke services through ~~WML API~~Markup API.

When the document~~card~~ 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 WAP client

The implementation of EFI AI as ~~WML API~~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~~entity in the "wml:go" and "a" elements in WML~~. For example, using the WML <go> element:

<~~wml:~~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

<~~wml:~~go href="efi://wallet/pay?value=200&currency=USD" >

or

<~~wml:~~go href=" efi://wallet/pay">

```
        <~~wml:~~postfield name="value" value="200"/>
        <~~wml:~~postfield name="currency" value="USD"/>
    </~~wml:~~go>
```

Note that EFI makes it possible to access services standardised by WAP 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 WAP client 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 ~~WML API~~Markup API, except the service with no name. The EF Broker SHOULD provide a service with no name. The exact contents of the document~~deck~~ returned depends on EF Broker implementation, but the content of the document~~deck~~ MUST present the list of available servers in a readable format with content equivalent to the getUnits() function in the WMLScript interface, including also class agent if it exists. Note that some servers may elect not to be visible to getUnits() and query(). 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~~WML deck~~ that is returned by this service depends ~~on~~ upon the EFI Class specification, but the content of the document~~deck~~ 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 notation 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~~WML deck~~ that is returned by this service depends ~~on~~ upon the EFI Class specification, but the content of the document~~deck~~ 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.

8.3 Discontinuous mode

The EFI service, once started, takes control and generates its own decks 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~~card'~~ concept.

8.3.1 Continuation document~~card~~

The continuation ~~card~~ 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. ~~refers to the card within the current deck where control should be passed upon completion of the service. The continuation card may provide a reference to a new deck to be loaded upon completion of the service.~~

~~The service does not protect the current context. Specifically, another deck may become current when the service terminates.~~

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

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

    <~~wml:~~postfield name="dest" value="example.wml~~2~~#done"/>

</~~wml:~~go>
```

Names of the parameters that hold a reference to the continuation document~~card, the number of continuation cards~~ and circumstances upon which the service chooses the specific continuation document~~card~~ 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~~Note~~ that the implementation of the service may not protect the context of the current ~~deck~~WML document. Specifically, another ~~deck~~ 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:

```
<~~wml:~~go href="efi://picture/take">

    <~~wml:~~postfield name="ret" value="retvar"/>

</~~wml:~~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.

~~8.3.3Example~~

~~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 card 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~~

```
~~<wml:go href="efi://music/play~~
~~      ?title=banana&result=time&dest=example.wml2%23result"/>~~
```

~~or by composing the URI using the `<wml:postfield>` tag~~

```
~~<wml:go href="efi://music/play">~~
~~    <wml:postfield name="title" value="banana"/>~~
~~    <wml:postfield name="result" value="time"/>~~
~~    <wml:postfield name="dest" value="example.wml2#result"/>~~
~~</wml:go>~~
```

~~Following is the example that shows the first invocation within the context of the deck of cards.~~

```
~~<!-- example.wml2 -->~~
~~<html>~~
~~           <wml:card id="play">~~
~~                 <wml:do type="accept" label="Start">~~
~~                       <wml:go href="efi://music/play~~

~~      ?title=banana&result=time&dest=example.wml2%23result"/>~~
~~                 </wml:do>~~
~~           <p>~~
~~                       Select 'Start' to play the banana~~
~~                 </p>~~
~~      </wml:card>~~
~~           <wml:card id="result">~~
~~                       That's it, $time seconds of pleasure~~
~~      </wml:card>~~
~~</html>~~
```

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

```
~~<!-- EFI generated deck -->~~
~~<html>~~
~~<wml:card>~~
~~<wml:do type="accept" label="OK">~~
~~<!-- EFI service gets ‘example.wml2#result’ from $dest -->~~
~~<wml:go href="example.wml2#result">~~
~~<!-- EFI service gets ‘time’ from $result -->~~
~~<!-- EFI service calculates duration of the music -->~~
~~<wml:setvar name="time" value="10"/>~~
~~</wml:go>~~
~~</wml:do>~~
~~<p>~~
~~<!-- EFI service gets ‘banana’ from $title -->~~
```

```
~~Playing banana~~
~~</p>~~
~~</wml:card>~~
~~</html>~~
```

8.4 Context management

When accessed from a WML document, t~~T~~he service is ~~generally~~ 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.

8.5 Status codes

As the ~~WML API~~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]
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 1. Status codes

8.6 UAProf

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

As a side effect of the architecture, the EFI service generates ~~cards and decks~~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 WAP cache mechanism [CACHE] MUST NOT be used when EFI services are accessed through the ~~WML API~~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&result=time&dest=example.wml%23result"/>
```

or by composing the URI using the `postfield` tag

```
<go href="efi://music/play">
    <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&result=time&dest=example.wml%23result"/>
          </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>
```

Change wording in SCRs from WML to Markup. Only the affected SCRs are shown.

This section does not reflect the changes to the SCRs from section 3 “Fix SCR Table” above. Changes from both of these sections will be preserved in the rollup specification.

A.2.1. APIs

Item	Function	Reference	Page	Status	Requirements
EFIFRM-API-1	WMLScript API~~WMLS~~	7	22	M	
EFIFRM-API-2	Markup API~~WML~~	8	40	M	

A.2.2. Broker

Item	Function	Reference	Page	Status	Requirements
EFIFRM-BR-1	The user is informed, when communicating with the EFI implementation. Either WMLScript or ~~WML~~ Markup API is used	7, 8	22, 40	M	

A.2.3. Scheme

Item	Function	Reference	Page	Status	Requirements
EFIFRM-SCH-2	Scheme-element in ~~WML~~ Markup API is used.	8.1	40	M	
EFIFRM-SCH-3	The Scheme component is treated case-insensitive ~~WML~~ in Markup API.	5.1.1	16	M	

A.2.6. Local Server

EFIFRM-LSRV-5	The EF Class Agent provide at least ~~present~~ the name of the server within the ~~WML Deck~~generated document.	8.2.2	41	O	
EFIFRM-LSRV-7	The EF Unit provide at least ~~present~~ the name of the server~~.~~ ~~Within the~~ in the generated document~~WML Deck~~.	8.2.3	41	O	

EFIFRM-LSRV-8	Cache is NOT used for EFI services with ~~WML~~ Markup API	8.7	44	M	