SmartTrust Growing Mobile

Guidelines

Development of WIB Services Delivery Platform 6

Document number: 50316003

Revision: A. 2003-12-03.

Contents

Document number: 50316003
Revision: A. 2003-12-03.

Document number: 50316003
Revision: A. 2003-12-03.

1 Introduction

The purpose of this document is to provide guidelines and help to developers of WIB services. The document comes as a complement to the WIG WML specification (see *WIG WML Specification – Version 4* [7]), and is specifically aiming to empower WIB services developers to create services that:

- are robust with regard to variable aspects, e.g. like the mobile phone capabilities
- utilize WIB and DP capabilities to the maximum
- are well written and maintainable
- are user friendly

The document in its present form is derived from another document (obsolete in DP 6.1) – “WIG Application Guidelines, Delivery Platform 6.0” – but has been rewritten with a clear focus on the needs of the WIB service developer.

The document may not be used for any other purposes than the ones described above. Specifically, it may not be used as a source of information when implementing WIB.

1.1 Revision History

Rev.	Comments
A	First version.

Document number: 50316003
Revision: A. 2003-12-03.

2 References

[1] 3GPP. *TS 23.040. Technical realization of the Short Message Service (SMS)*. Version 5.1.0 (2001-09). Available: http://www.3gpp.org/

[2] Ericsson Mobile Communications AB. *Enhanced Messaging Service – White Paper*. April 2001. Publication nr. LZT 108 4854 R1C. Available: http://www.ericsson.com

[3] ETSI. *GSM 11.14. SIM Application Toolkit (SIM-ME) Interface*. Version 8.5.0. Release 1999.

[4] Nokia. *Smart Messaging Specification*. Rev3.0.0. 2000-12-18. Available: https://secure.forum.nokia.com/

[5] SmartTrust. *WIG Browser Request Protocol Specification, DP 6*. Doc.no. 60084049.

[6] SmartTrust. *WIG Push Request Protocol Specification, DP 6*. Doc.no. 60084050.

[7] SmartTrust. *WIG WML Specification – Version 4*. Doc.no.50316002.

[8] SmartTrust. *WIB Plug-in Specification for Application Developers*. Doc.no. 60077036.

[9] Sony Ericsson, et al. *How to Create EMS Services*. Version 1.2 September 2002. Available: http://www.ericsson.com

[10] Sony Ericsson. *Enhanced Messaging Service (EMS) – Developers Guidelines*. September 2002. Publication nr. EN/LZT 108 5256 R2A. Available: http://www.ericsson.com

[11] Wireless Application Protocol Forum. *WAP Billing Framework*. Prototype Version 7 Aug 2001.

3 Definitions and abbreviations

Acronym	Definition
CGI	Common Gateway Interface
DP	SmartTrust Delivery Platform
EMS	Enhanced Message Service
HTML	HyperText Markup Language
NSM	Nokia Smart Messaging
SAT	SIM Application Toolkit
SMS	Short Message Service
SM	Short Message
(U)SIM	(Universal) Subscriber Identity Module
URL	Universal Resource Locator
WAP	Wireless Application Protocol
WIB	SmartTrust WIB™
WIB command	The smallest executable unit in WIB.
WIBlet	A program that may be executed in the WIB runtime platform.
WIG	Wireless Internet Gateway
WIG WML v3	WIG WML version 3. Supported by all version of DP. Sometimes referred to as “Old WML”.
WIG WML v4	WIG WML version 4. Supported by DP 6.1.
WML	Wireless Markup Language

4 Background

This chapter aims at presenting concepts and terminology which are fundamental to the understanding of a WIB service as well as the environment in which WIB services are developed and deployed.

4.1 Use Cases

The following use-cases are of great importance since they constitute the most common uses of WIB.

WIB request

Here the end-user brings up the WIB menu and selects one of the menu items. The selection will invoke an installed WIBlet which in turn sends a request to the content provider. Included in the WIB request is a URL and possibly also a query string. DP (WIG) receives the request and translates it into a standard HTTP GET or POST request. The MSISDN of the originating mobile phone may be optionally attached at the end of the HTTP request as a query parameter. The HTTP request is then sent of to the content provider Web server, which responds (in the normal case) with a WIG WML document. The WIG transforms the document into a WIBlet and finally sends the WIBlet back to the waiting WIB, who completes the loading and starts executing the new WIBlet.

Details of the HTTP communication between the WIG and the content provider Web server may be found in *WIG Browser Request Protocol Specification, DP 6* [5].

WIB push

This use-case is fundamentally different from the previous one since the sequence of actions in this case is initiated from the content provider side, and the end-user might not even be aware of that he or she will soon be involved in a WIB service.

The first action is when the content provider Web server (or some other entity at the content provider side) sends a push message to the WIG. This message contains information about the recipient(s) MSISDN as well as a WIG WML document. The WIG transforms the WIG WML document into a WIBlet and forwards the WIBlet to the WIB owned by the recipient, which executes the WIBlet. Optionally, the originator of the push message may also get confirmation that the WIBlet was successfully delivered to WIB.

Details of the HTTP communication as well as the WAP Push Access Protocol used in this use-case can be found in *WIG Push Request Protocol Specification, DP 6* [6]

4.2 WIB

From an original perspective, WIB may be described as a browser in much the same way as a WAP browser. A more precise and probably also more useful way to look at WIB, is as a platform for executing special programs known as WIBlets. As such, WIB provides an interpretation environment and acts as a virtual machine for WIBlets. A WIBlet is a sequence of WIB commands that may be executed by WIB. WIBlets are discussed in more detail in section 4.3 .

Since WIB originally was designed as a flexible way of offering SAT applications, a significant number of the commands offered by WIB have a direct counterpart in the SAT

Document number: 50316003
Revision: A. 2003-12-03.

command set (defined in *GSM 1.14* [3]). As an example, there are WIB commands like "Display Text", "Send SM" and "Set Up Call", to mention a few.

Nevertheless, many WIB commands are unrelated to SAT and offer functions that are internal to WIB, extending the capabilities of WIB far beyond what SAT offers. Taken together, these groups of WIB commands span over functionalities such as data communication, program flow, user interaction and data conversion.

Over the years, several versions of WIB have emerged:

- (WIB 1.0 – Year 1999, prototype version)
- WIB 1.1 – Year 2000
- WIB 1.2 – Year 2001
- WIB 1.3 – Year 2003

It is important to note that the different WIB versions are always backward compatible. This means that features supported in WIB 1.1 are also supported in the same way by more recent WIB version. The reverse is obviously not true, and therefore this document always tries to point out if a certain feature described in the text or by WIG WML in an example, is not supported by all WIB versions.

Appendix D summarizes the differences between the WIB versions released to date.

4.2.1 User interaction

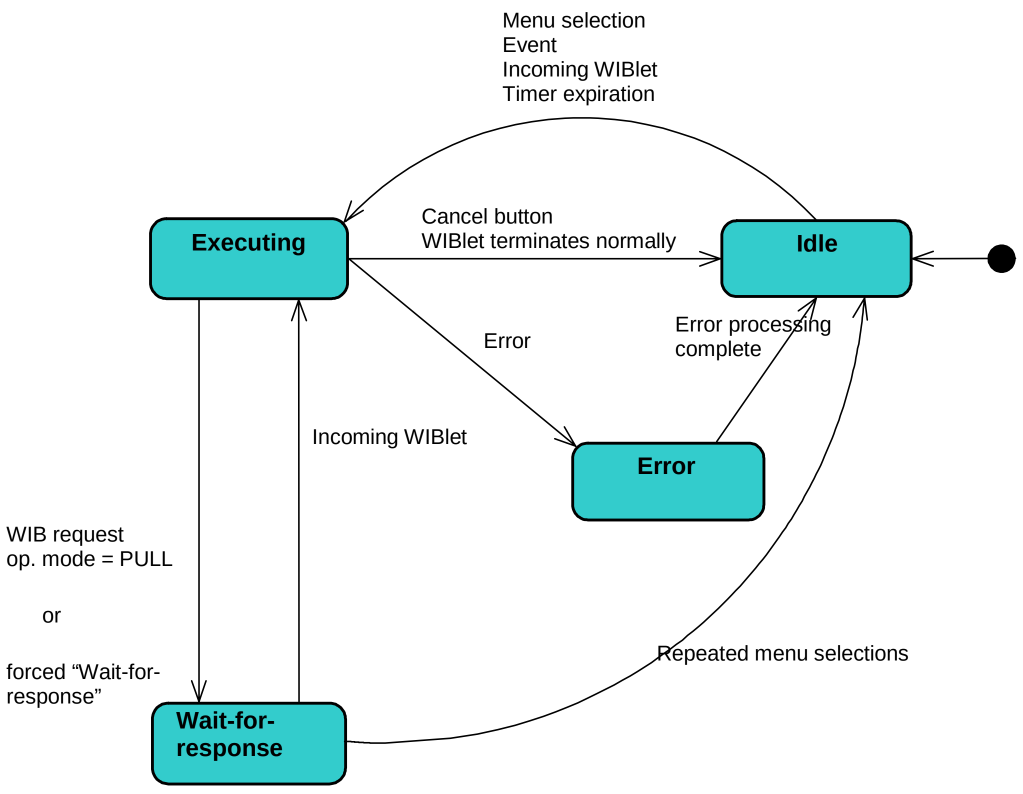
WIB is not equipped with any device for user interaction, like a screen or a keyboard. Instead WIB relies completely on the mobile phone to perform this task. Thus, for WIB to function properly, the mobile phone shall provide:

- A screen capable of displaying text and possibly also icons, if the WIB version is 1.3 or later.
- A keyboard for entering text and digits.
- Buttons for navigation:
  - Cancel button; used to terminate the WIBlet execution.
  - OK button; used to select a choice or acknowledge the information on the screen.
  - Backward move button; used to restart the currently executing WIBlet, clearing all the variables. If the WIBlet is already at the start position, the execution will stop.
- A tone-generator (not strictly required)

4.2.2 Execution model

An in-depth discussion of the execution model of WIB falls outside the scope of this document. Still, some basic facts are required to fully understand the behavior of WIB in certain situations.

In the state diagram below, the solid boxes represent the different states that WIB may occupy and the arrows represent transition between the states due to some action which is indicated beside the arrow.



The “Error” and “Wait-for-response” states are described in more detail later in this document.

The “Executing” state is where WIB is busy actually executing a WIBlet. In may be seen in contrast to the “Idle” state when WIB is doing nothing.

The transition from “Idle” to “Executing” state due to a “Menu selection”, “Event” or “Timer expiration” action, requires the WIBlet to be installed in WIB in order to be executed.

An “Incoming WIBlet” action starting from the “Idle” or “Wait-for-response” state leads to execution of the incoming WIBlet, which has in this case been dynamically loaded from the content provider.

WIB enters “Wait-for-response” state based on a set of rules which are quite complex and with details that are outside the scope of this document. Still, the most important rule to know for a WIB service developer, is that “Wait-for-response” state is entered based on the so called WIB *operational mode*.

The operational mode is determined by how the currently executing WIBlet was invoked. If it was invoked via a WIB menu selection, the operational mode is *PULL*. Conversely, if it was invoked by a WIB push and thereby coinciding with the “WIB push” use-case described in section 4.1 , the operational mode is *PUSH*.

The operational mode is important since it may determine the behavior of WIB when it sends a WIB request (commonly as a result of a `go` WIG WML element) to the content provider.

PUSH mode – After sending a request to the content provider, WIB will continue execution of the current WIBlet with the next WIB command. It will not wait for a response.

PULL mode – After sending a request to the content provider, WIB will halt execution of the current WIBlet and enter the “Wait-for-response” state. This also leads to that all WIB commands following the command that caused “Wait-for-response” to be entered, will not be reached by the WIBlet execution.

If, for some reason, the above described logic does not fit with the desire of the WIB service developer, the operational mode may be changed using the WIG WML element `setreturntarvalue`. In WIB 1.3 and later, it is also possible to explicitly control whether WIB should enter the “Wait-for-response” state or not, through the `go:enterwait` attribute.

For an WIG WML example how to manipulate the operational mode, see section 6.7 .

4.2.3 Variables

WIB has a reserved area of memory where variables may be written to or read from as WIB executes a WIBlet. The most common way of setting a variable in WIB is through the WIG WML elements `setvar`, `select` or `input`. Reading a variable occurs automatically whenever the variable is referred to by name. In WIG WML, variables have names describing their use like $(PRICE) or $(IMEI). In a WIBlet, variables are identified through numerical identifiers commonly known as *variable IDs*.

Variables come in two different flavors linked to their maximum persistence in WIB:

Local variables are variables that are created during the execution of a WIBlet and deleted automatically when the same WIBlet stops executing. Local variables occupy variable IDs in the range ‘00’h to ‘DF’h. This variable type is supported by all WIB versions.

Global variables are variables that are persistent throughout the execution of multiple WIBlets. Their life-length is limited by SIM reset, normally caused by a ME power-off. This type of variable is intended primarily for passing data to and from WIBlets, and is supported by WIB 1.3 and later.

The amount of memory available to variables is limited in all versions of WIB.

WIB 1.1 and WIB 1.2 – Support for up to 252 variables in each WIBlet and a maximum variable size of 255 bytes. The total size of the variable area is manufacturer specific.

WIB 1.3 – Support for up to 252 variables in each WIBlet and a maximum variable size of 255 bytes. The guaranteed size of the variable area is at least 1000 bytes for 30 variables.

4.2.4 “Wait-for-response” state

The purpose of the “Wait-for-response” state is to provide a user friendly waiting-period after WIB has stopped executing a WIBlet and the next WIBlet is being loaded. User friendly means that WIB should give the end user clear guidance what is actually going on, and provide updated progress information as frequently as possible.

Different WIB versions succeed differently well to achieve this goal:

- *WIB 1.1 and WIB 1.2*. WIB falls back to showing the WIB menu, which is not very user friendly since the end user may easily think that the WIBlet execution has stopped.
- *WIB 1.3 and later*. WIB provides textual and graphical (icon) information on the screen of the mobile phone in three different phases.
  - When the request is sent from WIB.
  - In the intermediate phase after sending the request but before reception of the response has started.
  - When WIB is receiving the response.

  The textual information is modifiable by the WIB service developer.

The “Wait-for-response” state also provides an opportunity to cancel the “Wait-for-response” state and force WIB back to the “Idle” state by repeated menu selections from the WIB menu.

4.2.5 Events

Since version 1.2, WIB supports invocation of installed WIBlets through a mechanism known as events. An event is a message sent asynchronously to WIB when any one of a set of special conditions occurs in the mobile phone.

When a WIBlet is invoked due to an event, WIB will optionally set one or possibly two variables to values that are tightly associated with the event, so that additional information may be propagated to the invoked WIBlet.

The following table shows events mapped to affected variables and what data they hold.

Event	Variable id and the data to be stored
MT call	‘90’h: Address of calling line identity ‘91’h: Called party subaddress
Call connected	‘90’h: Device identities
Call disconnected	‘90’h: Device identities ‘91’h: Cause of disconnection
Location status	‘90’h: Location status ‘91’h: Location information
User activity	

Document number: 50316003
Revision: A. 2003-12-03.

Idle screen available	
Card reader status	‘90’h: Card reader status
Language selection	‘90’h: Language selection
Browser termination	‘90’h: Termination cause
Data available	‘90’h: Channel status ‘91’h: Channel data length
Channel status	‘90’h: Channel status

For more information regarding these events, please consult *GSM 11.14* [3].

WIB 1.3 adds a new event, the *Start-up event*, to the list above. This event is sent to WIB when the (U)SIM has finished initializing after a power-up or reset.

4.2.6 Timers

From version 1.3, WIB is equipped with 8 timers that may be utilized by the WIB service developer. A timer can be thought of as a clock that can be set to alarm at a certain time into the future relative to the present time. In other words it functions as a count-down.

When the timer “alarms” it executes an installed WIBlet specified when the timer was started.

Since the actual work is performed by a WIBlet, it is up to the WIB service developer to decide the use-case for timers. Typically it may be used for periodic tasks like monitoring or periodic retrieval of information.

4.2.7 Plug-ins

In addition to the built-in WIB commands, WIB offers an extension mechanism known as plug-ins. This mechanism enables functions which were not originally thought of when WIB was designed, to be added at a later stage. WIB 1.3 even optionally supports downloading and installing new plug-ins over-the-air.

Currently there are around 20 standard plug-ins defined by SmartTrust, mainly covering functionality in the area of security and data retrieval. For a detailed description of these plug-ins and examples how they are used, please consult *WIB Plug-in Specification for Application Developers* [8].

4.2.8 Error handling

Occasionally an error situation arises where WIB has no other option than to prematurely abort the WIBlet execution. As the last action before it stops, WIB will display an error code and possible also an error-message on the screen of the mobile phone, with information what caused the error.

Most likely this situation will be confusing to the end user if it occurs in a “live” WIB service. Therefore it is very important to test WIB service thoroughly before they are deployed in order to remove as many potential errors as possible.

As a means of debugging a WIB service, the error codes are still useful. Appendix C lists the different error codes along with associated error messages.

4.3 WIBlet

As described earlier, a WIBlet is a sequence of WIB commands. It is important to note that a WIBlet is not the same as a WIG WML document. The relation between WIG WML and WIBlet is similar to the relation between source code and object code observed in almost any system for program development. The source code (WIG WML) is compiled into object code (WIBlet) which is understood by a computer, in this case WIB. The compilation from WIG WML to WIBlet is always carried out by DP in some way or another, with the WIG as the foremost example.

The fact that a WIBlet is actually object code that WIB execute may seem obvious when described in this context. Still, It has sometime subtle implications on the way WIB services should be designed, since WIG WML is more of a “page-description” language in the spirit of HTML and WAP WML where the notion of linear program order is suppressed in favor of a more tree-like view.

4.3.1 Installed WIBlets

The term *installed WIBlet* is used throughout this document to indicate that a WIBlet is stored locally in WIB, prior to the moment when it is executed. In other words, loading is not required. It should be seen in sharp contrast to the term *dynamically loaded WIBlets*, which is used to express that a WIBlet is loaded from the content provider before it is executed.

The details of WIBlet installation are rather complex and falls outside the scope of this document.

4.4 WIB Services

A WIB service may be perceived in different ways depending on the point-of-view:

- From WIB point-of-view, it is a one or more WIBlets that may be executed in WIB.
- From an end-user point-of-view, it is the collective experience created by repeated interactions with WIB in order to reach a certain goal. Using this rather vague “definition” it is not always easy to know when transition between WIB services occurs.
- From a service development point-of-view, it is mainly a collection of WIG WML documents, CGI scripts and related data residing on a Web server, together forming what is commonly known as a Web application.

All these attempts to capture the nature of a WIB service should be taken rather informally, since they all omit information that is not so easily categorized.

4.4.1 Installed WIB services

Holding on to the view that a WIB service is a collection of WIBlets, it is possible to *install* a WIB service, in parts or as a whole.

The main reason for installing parts of a WIB service is to reduce the total loading time of the service and possibly also reduce over-the-air traffic. In an extreme case all the pieces of a WIB service may be installed in WIB, even if this is an unlikely case.

If part of the WIB service is installed and part of the WIB service is loaded dynamically, certain limitations related to transition between WIBlets arise:

- *WIB 1.1 and WIB 1.2*. A WIB Service may have its starting point in a installed WIBlet, but once transition occurs to a WIBlet loaded dynamically, there is no way of “getting back” to the (any) installed WIBlet without the end-user restarting the WIB service.
- *WIB 1.3 and later*. WIB services may be designed so that transition between installed and dynamically loaded WIBlets can occur in both directions without restrictions.

4.5 DP and the WIG

DP offers a versatile environment for developing and deploying WIB services. It hides from the WIB service developer many of the complex issues related to the GSM network, such as over-the-air security, formatting of SMS messages and SMS-C protocols. Instead the WIB service developer is offered, through the WIG, a Web based interface which should be reasonably familiar to developers that have at some point developed Web based applications for the Internet.

4.5.1 WIG WML

The principal language used to develop WIB services is WIG WML, which is also the application language supported by the WIG. Initially WIG WML was aligned with WAP WML, but from version 4 (supported by DP 6.1/WIG 4) it has evolved into a language by its own right. Appendix A covers frequently asked questions concerning the rationale behind WIG WML v4, while Appendix B covers migration of WIB services written in an earlier version of WIG WML to version 4.

4.5.2 Communication via SMS

Even if the over-the-air communication aspects are hidden from the WIB service developer, certain limiting conditions should nevertheless be pointed out.

The Short Message Service (SMS) which is used to send data to WIB is a fairly narrow banded communication channel by today’s standards. Each SM can hold around 120 bytes of service (WIBlet) data, and if the payload exceeds this limit, it will be split into two or more SMs. The delay experienced by the end-user accessing the WIB service will increase with the number of SMs required to send a complete WIBlet to WIB.

There is also a fixed limit to how many SMs that may be consumed by a single WIBlet if the WIBlet is loaded dynamically.

Document number: 50316003
Revision: A. 2003-12-03.

- *WIB 1.1 and WIB 1.2*. 5 SMs from the content provider to WIB and 3 SMs from WIB to the content provider.
- *WIB 1.3*. 7 SMs from the content provider to WIB and 5 SMs from WIB to the content provider.

5 Robust WIG WML design

This chapter deals with a number of constraints related to the mobile phone and the SIM card that must be kept in mind when developing a WIB service.

5.1 Splitting text

When working with WIG WML documents containing text, it is important to understand that there is a limit for how many characters the mobile phone can display at the same time. The limit differs between different SIM card manufactures and mobile phones vendors, but it is usually between 110 - 140 characters.

If a text contains more characters that the display limit, the WIG will impose a split of the text when the document is transformed to a WIBlet. The text will then be contained in two or more consecutive screens on the mobile phone, regardless of how the text is constructed.

The developer may take control over this process by splitting the text manually using several p elements in succession in the WIG WML document. This is the recommended way of dealing with long texts since the outcome is now under the control of the developer.

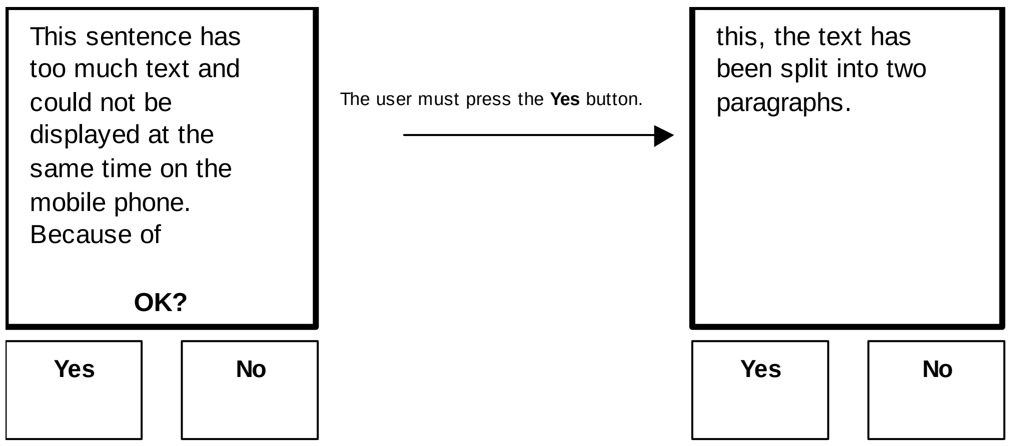
In the two examples below, the display limit is 110 characters.

Example [1]

Here the WIG “auto-split” feature is used to handle the text.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Main">
    <p>
      This sentence has too much text and could not be displayed at the
      same time on the mobile phone. Because of this, the text has been
      split into two paragraphs.
    </p>
  </card>
</wml>
```

Result:

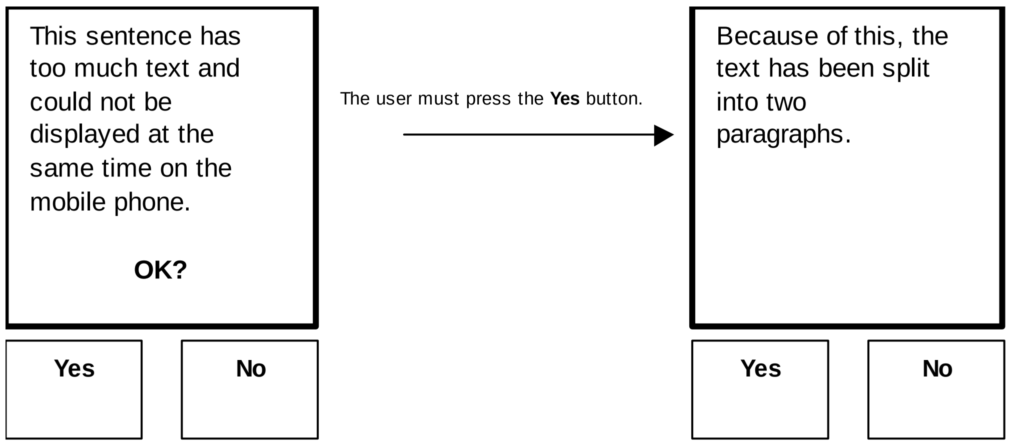

This sentence has too much text and could not be displayed at the same time on the mobile phone. Because of
OK?
Yes
No
The user must press the Yes button.
this, the text has been split into two paragraphs.
Yes
No

Example [2]

Here two <p> tags are used to handle the text.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Main">
    <p>
      This sentence have too much text and could not be displayed at the
      same time on the mobile phone.
    </p>
    <p>
      Because of this, the text has been split into two paragraphs.
    </p>
  </card>
</wml>
```

Result:


This sentence has too much text and could not be displayed at the same time on the mobile phone.
OK?
Yes
No
The user must press the Yes button.
Because of this, the text has been split into two paragraphs.
Yes
No

5.2 Alphanumeric passwords

Some mobile phones do not allow the use of the `input:type="password"` attribute together with alphanumeric input. Thus, for the `type="password"` to work on all mobile phones, it has to be used together with `format="*N"`.

Example [3]

```
<input name="PWD" type="password" format="*N"/>
```

5.3 Text size in select element

The `select` element may only be used with a limited number of characters. The sum of all character displayed on some mobile phones, that is the title plus the text in each option, may not exceed 110 characters.

Example [4]

```
<card>
  <p>
    <select title="Colour" name="C">
      <option value="1">Blue</option>
      <option value="2">Red</option>
      <option value="3">Yellow</option>
    </select>
  </p>
</card>
```

In this example the number of characters is ("Colour" = 6) + ("Blue" = 4) + ("Red" = 3) + ("Yellow" = 6) = 19.

The `select:title` attribute is unfortunately displayed in many different ways and sometimes not at all, depending on the mobile phone manufacturer.

WIB and/or the WIG configuration also puts limitations on how much information (i.e. `select:title`, `option` text, `option:value` and `option:onpick` URLs) you can totally have within one `select` element. It is advisable to try to keep the total length of all the information as short as possible to ensure proper operation for all configurations.

5.4 SAT limitations in the mobile phone

WIB supports several features that put quite strong requirements on the SAT implementation in the mobile phone. For example:

- Icons (WIB 1.3)
- Events (WIB 1.2)
- Timers (WIB 1.3)
- Launch browser (WIB 1.2)

When developing a WIB service that builds on one of these features, it is inevitable that the service will fail on some mobile phones. This fact must be considered by the service developer, and weighted against the value gained by using the feature(s).

Testing may reveal to what extent a WIB service works on different mobile phones, but it can not fix the problem itself.

6 WIG WML How-to

6.1 Coding style

Because of the limitations pointed out in section 4.5.2 , it is desirable to design a WIB service in such a way that the number of SM's required in the SMS communication is kept to a minimum. There are no universal rules how this can be achieved, but a number of general recommendations are still in place:

- Small WIG WML documents are better that big
- Short URL's are better that long
- Simple logic is better that complex
- Exploit the use the static URL references to reduce the WIBlet size. See section 6.6 for details.
- Make dynamically loading of a WIBlets occur in places where it seems natural from an end user point-of-view, and thereby causing least annoyance.
- Use *Wireless Application Creator* (supplied by SmartTrust) to analyze and optimize the WIBlet size.
- WIG supports caching of WIG WML documents. This may be utilized by the WIB service developer to reduce WIBlet loading time.
- Avoid repeating text strings within the same WIG WML document. Instead use variables wisely.
- Learn how to design compact end-user dialogs without sacrificing usability. In fact, using too many words on a small display has a negative impact on comprehension.

The last recommendation is probably the most important one, since designing a WIB service requires a somewhat different mind-set than when designing an ordinary Web application where screen size and bandwidth are issues of minor importance.

6.2 Closing card element stops WIBlet

The WIBlet execution will stop when a closing `card` element is encountered. Jumping between cards is possible using a card reference in the `go:href` attribute.

The examples below illustrate the two cases:

Example [5]

In this example only the text "Hi!" is displayed since WIB stops after the first card.

Document number: 50316003
Revision: A. 2003-12-03.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Main">
    <p>
      Hi!
    </p>
  </card>

  <card id="Next">
    <p>
      Hi again!
    </p>
  </card>
</wml>
```

Example [6]

In this case both "Hi!" and "Hi again!" are displayed since WIB will jump to the next card in the document.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Main">
    <p>
      Hi!
     <go href="#Next"/>
    </p>
  </card>

  <card id="Next">
    <p>
      Hi again!
    </p>
  </card>
</wml>
```

6.3 Jumping as the result of a selection

The `option:onpick` attribute is very powerful since it allows three types of “jumps” to occur as the result of a selection.

Example [7]

This example shows how to jump to a card depending on the selection.

```
<card>
  <p>
    <select title="Colour" name="C">
      <option onpick="#Blue">Blue</option>
      <option onpick="#Red">Red</option>
    </select>
  </p>
</card>
.
<card id="Blue">
.
</card>
.
<card id="Red">
.
</card>
```

Example [8]

This example shows how to load and execute a new WIBlet depending on the selection.

```
<card>
  <p>
    <select title="Colour" name="C">
      <option onpick="http://server/path/bluefile.wml">Blue</option>
      <option onpick="http://server/path/redfile.wml">Red</option>
    </select>
  </p>
</card>
```

Example [9]

Compatibility note: This example is only applicable for WIB version 1.3 or later.

This example shows how to execute an installed WIBlet depending on the selection.

```
<card>
  <p>
    <select title="Colour" name="C">
      <option onpick="wiblet://server/path/bluefile.wml">Blue</option>
      <option onpick="wiblet://server/path/redfile.wml">Red</option>
    </select>
  </p>
</card>
```

6.4 Using icons

Compatibility note: This section is only applicable for WIB version 1.3 or later.

To be able to use icons in a WIB service, the following tasks must be accomplished first:

- The SIM card must have icon data installed. Exactly how this is performed is outside the scope of this document.
- The WIB service developer must be provided with a list of installed icons and the icon identifier for each of them.

When icons are in place in the SIM, actually using them from WIG WML is rather straightforward.

Example [10]

In this example, an icon is displayed alongside with the text “Icons are cool!!”.

```
<card>
  <p iconid="3">
    Icons are cool!!
  </p>
</card>
```

6.5 Retrieving events specific information

Compatibility note: This section is only applicable for WIB version 1.2 or later.

To be able to read variables set by WIB when an event occurs, a special syntax must be used where the variable ID is specified as part of the variable name.

Example [11]

```
<card>
  <p>
    MT Call Event Ocurred.<br/>
    Address: $(ADDRESS:IDx90)<br/>
    Subaddress: $(SUBADDRESS:IDx91)
  </p>
</card>
```

6.6 Static URL references

This feature allows the WIG operator to predefine a set of URLs that are often used and store them permanently in the WIG. Instead of sending the full URL back and forth between the WIG and WIB, only a reference ID is sent thus reducing the WIBlet size and minimizing the number of SMs over the air interface. Static URL references may be used in installed WIBlets as well.

Also, using static URL references instead of full URLs makes is possible to change the URL (or even change between HTTP and HTTPS) for dynamically loaded WIBlets, without updating any of the installed WIBlets.

Figure 1. illustrates a scenario where WIB sends a request containing a static URL reference to the WIG. Originally, the static URL reference was specified in a WIG WML `go:href` attribute like this:

```
<go href="!myref!b"/>
```

When the WIG receives the request, it looks up "myref" in the database and constructs the full URL. Then it handles the URL as if it was received directly from WIB.

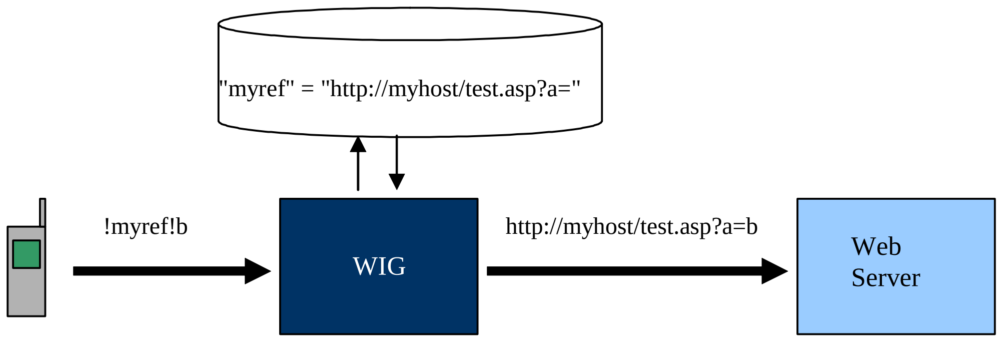

"myref" = "http://myhost/test.asp?a="
!myref!b
WIG
http://myhost/test.asp?a=b
Web
Server

Figure 1. Static URL References.

6.6.1 Static URL references in WIG WML

In WIG WML, the static URL reference is indicated in a URL with a leading `!` character followed by the reference ID and another `!` character. E.g.:

```
<go href="!refID!restofmyurl"/>
```

Only one static URL reference is allowed per URL, and the static URL reference has to be first in the URL.

Variables are not supported in a static URL reference.

Example [12]

This example illustrates use of static URL references.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">

<wml>

<card id ="card1">
  <p>
    <setvar name="VAR1" value="A"/>
    <setvar name="VAR2" value="B"/>
    <select name="VAR3" title="Please select">
      <option onpick="!a!">Site A</option>
      <option onpick="#card2">Card 2</option>
    </select>
  </p>
</card>

<card id="card2">
  <p>
    <do type="accept">
      <go href="!myref!$(VAR1)&amp;b=$(VAR2)"/>
    </do>
  </p>
</card>


</wml>
```

6.7 Changing the WIB operational mode

As described earlier in this document, it is possible to change the operational mode of WIB through the `setreturntarvalue` element. Changing the operational mode affects how WIB behaves upon sending a WIB request.

Example [13]

In this example, WIB is forced into PUSH operational mode through the `setreturntarvalue` element, which will prevent WIB from entering the “Wait-for-response” state when executing the `go` element.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
     <setreturntarvalue recordid="2"/>
     <!-- Now WIB is in PUSH op. mode!! -->
     <go href="http://www.smarttrust.com/no_response.pl"
         enterwait="mode-dependent"/>
    </p>
  </card>
</wml>
```

Example [14]

In this example, WIB is forced into PULL operational mode through the `setreturntarvalue` element, which will cause WIB to enter “Wait-for-response” state after executing the `go` element.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
     <setreturntarvalue recordid="1"/>
     <!-- Now WIB is in PULL op. mode!! -->
     <go href="http://www.smarttrust.com/send_a_response.pl"
         enterwait="mode-dependent"/>
    </p>
  </card>
</wml>
```

6.8 WIG WML examples

This section contains a number of WIG WML examples to get an idea of how to write a WIB service.

Please note that all URL’s in the example code are invalid.

Example [15]

This example illustrates navigation between cards within a WIG WML document.

Document number: 50316003
Revision: A. 2003-12-03.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="main">
    <p>
      Card selection example.
      <select title="Select card">
        <option onpick="#CARD1">Card1</option>
        <option onpick="#CARD2">Card2</option>
      </select >
    </p>
  </card>

  <card id="CARD1">
    <p>
      Now you’re in CARD1.
    </p>
  </card>

  <card id="CARD2">
    <p>
      Now you’re in CARD2.
    </p>
  </card>
</wml>
```

Example [16]

This example illustrates use of the `playtone` element. A dial tone will be played during 3 second and the text "Tone!" will be displayed at the same time.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="main">
    <p>
      This example will play a tone!
      Press ok.
      <playtone toneid="dial" title="Tone!" duration="3"/>
    </p>
  </card>
</wml>
```

Example [17]

This example illustrates navigation between WIG WML documents, where the new WIG WML documents are fetched using two different URLs.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="main">
    <p>
      Document selection example.
      <select title="Select card" name="service">
        <option value="document1.wml">Link1</option>
        <option value="document2.wml">Link2</option>
      </select>
      <go href="http://www.content.com/$(service)"/>
    </p>
  </card>
</wml>
```

File document1.wml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Service1">
    <p>
      Welcome to service 1!
      <input title="Please enter your firstname."
             type="text" name="firstname"/>
      You entered $(firstname).
    </p>
  </card>
</wml>
```

File document2.wml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card id="Service2">
    <p>
      Welcome to service 2!
      Location data will be sent to location service!
      Press ok!
      <providelocalinfo cmdqualifier="location" destvar="Info"/>
      <go href="http://server/loc/pos.asp?info=$(Info)"/>
    </p>
  </card>
</wml>
```

7 Building WIB services

This chapter describes features that need special attention when building WIG WML services.

7.1 Caching

The WIG acts like a proxy cache for requests coming from WIB. This means that the WIG will save a copy of the WIG WML document it receives from Web servers, so that the next time there is a request for the same document, the WIG will use the copy it has instead of asking the original Web server.

The main reason for caching WIG WML documents is to reduce the time it takes to dynamically load a WIBlet in WIB.

It is recommended to use the WIG cache as far as possible.

The WIG cache may be controlled by the WIB service developer using cache control HTTP headers. For details see *WIG Browser Request Protocol Specification, DP 6* [5].

7.2 Cookies

When the WIG requests a WIG WML document from a Web server, the Web server may also respond with a piece of state information in addition to the WIG WML document. Included in the state object is a description of the range of URLs for which that state is valid. Any further requests made by the WIG which fall in that range will also include transmittal of the current value of the state object back to the Web server. The state object is commonly referred to as a *cookie*.

Please note that cookies are never actually sent to WIB and therefore do not consume bandwidth. Instead they are managed and stored in the WIG, on behalf of WIB.

Cookies are most often used to enable user-side customization of Web applications.

See *WIG Browser Request Protocol Specification*, *DP6* [5] for more information on cookies.

7.3 Sending SMs

The WIG/WIB supports two different ways of sending an SM. One that is called *server SM* is sent as an ordinary SM directly to the mobile phone. The other, here called *WIB SM* is sent as a WIBlet to WIB where WIB generates an SM which it sends to another mobile phone. To generate either of the two types from a WIG WML document, use the WIG server-side plug-in call `sendserversm` (or `sendserverdatasm`) for server SM and the `sendsm` element for WIB SM.

The originating address in server SM will be the same as if the SM had been sent using a WIB SM, i.e. the MSISDN of the mobile phone.

It is possible to send both Enhanced Message Service (EMS) and Nokia Smart Messaging (NSM) messages by using the `sendserverdatasm` plug-in or the `sendsm` element (requires WIB 1.3 or later).

Document number: 50316003
Revision: A. 2003-12-03.

Details regarding the format and creation of EMS messages may be found in *Enhanced Messaging Service – White Paper* [2], *Enhanced Messaging Service (EMS) – Developers Guidelines* [10] and *How to Create EMS Services* [9].

Details regarding the technical realization of EMS may be found in *TS 23.040. Technical realization of the Short Message Service (SMS)* [1].

Details regarding the NSM format may be found in *Smart Messaging Specification* [4].

7.4 Tariff class

To charge for a WIG WML document, a tariff class may be used. The tariff class should be included among the HTTP headers in the Web server response to a WIG request.

The syntax for the tariff class follows the WAP syntax according to *WAP Billing Framework [11]*.

```
X-WAP-Payment-Info: content-value-class = 1*10 digit
```
(Zero is not allowed.)

Before a content provider can use a tariff class, it has to be defined in DP. The tariff class and a corresponding SMSC id should be defined by the DP Administrator. Then each content provider has to be provided with a list of tariff classes that he is allowed to use.

When the response/push request reaches the WIG Server, the tariff class is checked against the content provider's white list of tariff classes. If the content provider is not allowed to use the tariff class, it is removed from the response/push request. If the content provider is allowed to use the tariff class it is used internally by DP for creating the correct charging information.

Tariff classes can be used together with caching. The tariff class will then be cached according to the same rules that apply to the WIG WML document.

Example [18]

This is an example of how to generate the tariff class header in a JSP document.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <% response.addHeader("X-WAP-Payment-Info",
       "content-value-class=123");%>
      Tariff class included in HTTP header.
    </p>
  </card>
</wml>
```

Example [19]

This example shows an ASP document that will generate the same result as the previous example.

```
<%=Response.AddHeader ("X-WAP-Payment-Info", "content-value-class=123")
%>
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
  "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Tariff class included in HTTP header.
    </p>
  </card>
</wml>
```

Appendix A WIG WML v4 FAQ

Q1. Why do we need a new WML?

There are two main reasons why we created a new WML syntax:

1) The old WIG WML syntax, from now on called WIG WML v3, is not very user friendly when it comes to functionality which is not covered by similar functionality in WAP WML. A good example is the WML Script calls which are used to call some of the more infrequently used SAT commands. This syntax is error prone and with a low verbosity, affecting the possibility to create and later on comprehend a WML document.
2) WIB 1.3 introduces many new features that must be reachable from WIG WML. Continuing on the WIG WML v3 track would create syntax that breaks the (although vague) principle of WAP WML compatibility and at the same time makes a mess out of the language. Instead we decided to break it with style by redesigning certain aspects of the language, and thereby improving the usability of the language as well as the fitness for its purpose.

Q2. Do we have to update all our existing applications?

No. An application must be updated only if you migrate to WIB 1.3 and at the same time want to use WIB 1.3 features.

Q3. Can the new WML be used for WIB 1.1 and WIB 1.2?

Generally yes, but only using features that are available in WIB 1.2 and earlier. If fact we strongly recommend all new applications to use WIG WML v4 since it is more user friendly and also catches more errors at an earlier stage. It also simplifies future migration to WIB 1.3.

Q4. Why do I have to update my application completely just to add icons?

You don’t have to update your application completely. Only those documents that require icons have to be updated, and in many cases the change involves only adding the WIG WML v4 header and adding the icons in the right places (which you would have to do in any case).

Q5. I want to upgrade from DP 5. Will it cause any problems?

No. WIG WML v3 is still supported in DP 6.1.

Q6. I want to upgrade from DP 6.0. Will it cause any problems?

No. Same answer as Q5.

Q7. Our developers can not learn this new WML. Why can't we use the old one?

You can use WIG WML v3 in DP 6.1. The only drawback is that you will not be able to use WIB 1.3 specific features.

Q8. Can new and old WML be mixed?

Not within the same WML document. A WIB service consisting of several WML documents may mix WIG WML v3 and WIG WML v4 freely between the documents.

Q9. The WIBlet size appears bigger for WIB 1.3 compared to WIB 1.2. Is this the case?

In general the answer is yes. Many of the WIB 1.3 commands are extended versions of the same commands in WIB 1.2 and WIB 1.1. They offer more features but also occupy slightly more space.

Document number: 50316003
Revision: A. 2003-12-03.

Appendix B WIG WML v4 migration guide

This appendix aims to be a quick-reference for those who migrates WIB services written in pre version 4 WIG WML to WIG WML v4.

B.1 Add Doctype

WIG WML v4 requires that the following document type declaration occurs before any of the WIG WML elements.

```
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
     "http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
```

Please note that the declaration is case-sensitive.

Failing to reproduce this document type declaration in the beginning of the document will cause the WIG to interpret the document as a pre version 4 document, which will most likely lead to errors.

B.2 Change document encoding

Earlier versions of WIG WML had a concept whereby the document encoding in the XML declaration also determined the default character encoding in WIB. In addition, the default document encoding was ISO-8859-1 if the XML declaration was omitted or did not contain an encoding declaration. These two concepts have been abandoned in WIG WML v4. Instead, the default document encoding is now UTF-8, which is in line with the XML standard, and the default encoding in WIB is no longer determined by the document encoding, but by the `wml:wibletenc` attribute.

To convert pre version 4 WIG WML to WIG WML v4, do one of the following:

a) If the WIG WML document does not contain an XML declaration or the document encoding is missing in the XML declaration, add or modify the XML declaration in the beginning of the document so that it reads:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

b) If the WIG WML document does contain the document encoding in the XML declaration, and the encoding is ISO-8859-1 or ISO-8859-7, nothing needs to be done.

c) If the WIG WML document does contain the document encoding in the XML declaration, and the encoding is UTF-8, modify the `wml` element according to:

```
<wml wibletenc="UCS2">
```

B.3 Remove a and anchor elements

All occurrences of the `a` and the `anchor` element must be re-written using the `select` element with options using the `onpick` attribute instead.

B.4 Remove emphasis elements

WIG WML v4 does not allow the following elements:

`b`, `i`, `u`, `big`, `string`, `em`, `small`

Since none of these elements had any visible effect, they may all be safely removed.

B.5 Remove img elements

All occurrences of the `img` element must be removed. This is always safe since the element had no effect in pre version 4 WIG WML.

B.6 Remove do elements

All occurrences of the start and end `do` tags should be removed. Elements contained within the `do` element should be kept as before.

B.7 Attribute values are case sensitive

Since all attribute values are case sensitive in WIG WML v4, the document must be corrected with this in mind.

B.8 Tag names are case sensitive

In WIG WML v4 all tag names are case sensitive. I.e. `<P>` is different from `<p>`. The rule in WIG WML v4 is that tag names are written using exclusively lower case letter.

B.9 WMLScript function calls must be rewritten

The WMLScript syntax has been removed as part of WIG WML v4. This is actually the biggest change compared to pre version 4 WIG WML. Consequently, all WMLScript function calls must be rewritten using new language elements as specified in the WIG WML v4 specification.

The general rule to find the right element to use in WIG WML v4 is to remove the ‘wig’ prefix from the WMLScript function name and use the new name as the look up in the WIG WML v4 specification. Often attributes in the element that replaces a WMLScript function call have a close resemblance with the parameters to the WMLScript function, which makes migration rather simple in most case. In the process, make sure that URL escaped values (like %XX where XX is a hexadecimal number) are replaced with their unescaped value.

B.10 No automatic title in select element

In an earlier version of WIG WML, there was a rule that said if the `select:title` attribute was omitted, the text preceding the `select` element would be used as title for the select command in WIB. This is not the case in WIG WML v4, and omitting the `title` attribute in the select or `input` element will cause the title to be empty in WIB, independent of any preceding text.

Document number: 50316003
Revision: A. 2003-12-03.

B.11 userdata parameter for wigSendServerDataSM must be devided into two parameters

When converting the `wigSendServerDataSM` WMLScript function call to the `sendserverdatasm` server-side plug-in call in WIG WML v4, the `userdata` parameter value must be divided into a user data header parameter and possibly also a user data parameter. The user data header length (UDHL) which is the first byte in the original `userdata` parameter must also be removed.

B.12 Strict card id attribute value

`card:id` attributes which does not start with an underscore or a letter must be rewritten to do so.

B.13 Replace &nbsp; and &shy;

Character entities `&nbsp;` and `&shy;` are not supported in WIG WML v4 and must be replaced with numeric character entities `&#160;` and `&#173;` respectively.

B.14 Rewrite WIGVARxHH

WIGVARxHH variable names, where HH is a hexadecimal number, must be replaced with :IDxHH

It is recommendable, although not mandatory, to have a describing (variable) name in front of the colon, e.g.:

LOCATION:IDx91

B.15 Rewrite dial-strings

In all dial-strings, occurrences of lower-case characters “a” - “e” must be replaced with “A”- “E”, respectively. This affects the `wigSendSM` and the `wigSetupCall` WMLScript function call.

Appendix C Error codes

If an error occurs, the error is presented to the end user according to the format description described below. Note that different WIB version present errors somewhat differently.

WIB 1.3 and later:

`[ErrorMessage] ErrorCode CommandTag [TerminaleResponse | FileIdentifier]`

WIB 1.1 and WIB 1.2:

`[ErrorMessage] ErrorCode CommandTag`

(Note: [] encloses an optional field(s) and | symbols an alternative)

`ErrorMessage` is a descriptive text explaining the nature of the error, but may not always be presented to the end user. It may also be a general text message like “Error”.

The `ErrorCode` is always displayed, and may be used to locate the error message associated with the error in cases where the `ErrorMessage` is omitted.

`CommandTag` is the (numeric) tag of the WIB command that was executing when the error occurred. If the error did not occur as the result of a failed WIB command, the `CommandTag` field is set to "XX"

`TerminalResponse` is the terminal response general result value if a proactive SIM command was issued as part of the WIB command.

When the error code is ‘1F’h, “Failed to access file”, the file identifier of the file that could not be accessed shall be given instead of the `TerminalResponse`.

	ErrorCode	Description
File Access Errors	‘01’h	Failed to find/read EF$_{Bytecode}$
	‘02’h	Failed to find/read EF$_{TAR}$
	‘03’h	Failed to access/read EF$_{ErrorText}$
	‘04’h	Failed to find/read EF$_{SMSHeader}$
	‘05’h	Failed to read key file.
	‘06’h	Failed to find/read EF$_{VersionInformation}$
	‘1F’h	Failed to access file.
Byte Code Errors	‘20’h	Unknown WIB command found.
	‘21’h	Variable substitution failed.
	‘22’h	Too many variables used.
	‘23’h	Out of variable memory.
	‘25’h	SMS TPDU Tag in incoming SMS not found.
	‘26’h	Creation of SELECT ITEM failed.
	‘27’h	Encryption/decryption failed.

Document number: 50316003
Revision: A. 2003-12-03.

	‘28’h	Out of buffer space.
	‘29’h	Plug-in not found.
	‘2A’h	Bad format on proactive SAT command.
	‘2B’h	"Goto" out of bounds.
	‘2C’h	E2PROM memory problem.
	‘2D’h	Command error in client bound message.
	‘2E’h	Configuration error.
	‘2F’h	SET RETURN TAR VALUE not allowed.
	‘30’h	Script not found.
	‘31’h	Timer management failure.
	‘32’h	Return from script not allowed.
	‘33’h	Invalid input data to WIB command.
	‘34’h	Invalid incoming message.
Mobile Phone Errors	‘40’h	Proactive command rejected by ME.
	‘41’h	Wrong type of command returned by ME in terminal response.
	‘42’h	GET INPUT did not return a string.
	‘43’h	No item identifier was returned by ME in the terminal response to a SELECT ITEM.
	‘44’h	Temporary error occurred in application. Please try again later.
	‘45’h	Error in format of received SM.
	‘46’h	Command not supported by the mobile.
	‘47’h	SET UP CALL failed.
	‘48’h	SET UP EVENT LIST failed.
Plug-in Errors	‘60’h	Invalid input parameters.
	‘61’h	Input out of bounds.
	‘62’h	Output overflow.
	‘63’h	RSA error.
	‘64’h	Illegal operation.
	‘65’h	Integrity error.
	‘66’h	PIN length error.

Default Errors	‘D0’h	Error in application occurred. Please call support..
Proprietary Errors	Error codes in the (‘E0’h..’FF’h) range are proprietary and depends on the WIB implementation.	

Appendix D WIB compatibility information

This appendix describes the differences between the WIB versions released to date.

D.2 WIB 1.1.1 vs. WIB 1.1

WIB 1.1.1 has the same functionality as WIB 1.1 with the exception of one new feature:

- **SMS Default in Get Input:** The change applies to the `input` element, where it is now possible to mix UCS2 text with SMS Default input.

D.1 WIB 1.1 vs. WIB 1.2

The list below shows new WIB commands added in WIB 1.2:

- **Check Terminal Profile:** Better error handling in WIB.
- **Conditional Jump:** Jump to different WIG WML cards depending on a variable value.
- **Display Text Clear After Delay:** Clears the text on the screen of the mobile phone after a time interval without interaction from the user.
- **Launch Browser:** Enables starting a WAP browser session from WIB.
- **Set Extended:** Allows value data for the `setvar` element to be a mix of static data and variable references.
- **Substring:** Copies a sub-string from one variable to another variable.

D.3 WIB 1.2.1 vs. WIB 1.2

WIB 1.2.1 has the same functionality as WIB 1.2 with the exception of one new feature:

- **SMS Default in Get Input:** The change applies to the `input` element, where it is now possible to mix UCS2 text with SMS Default input.

D.4 WIB 1.2.1 vs. WIB 1.3

The list below shows new WIB commands added in WIB 1.3:

- **Execute Local Script:** Enables “local links”, i.e. the ability to go from any WIBlet to an installed WIBlet.
- **Submit Extended:** Support for extensive progress information using both text and icons. Optional support for bookmarks.
- **Launch Browser Extended:** Fixes several issues with the “old” command which was more or less broken.
- **Add/Subtract:** Enables simple arithmetic in WIB.
- **Convert Variable:** Enables various forms of conversion between text and binary data.
- **Group/Ungroup:** Packing of many variables into one and vice versa.

Document number: 50316003
Revision: A. 2003-12-03.

- **Set Up Call Extended:** New formats for the destination address with support for variable references. New alpha identifier for call set up phase. Icon support.
- **Display Text Extended:** Support for text clearing, immediate response indicator and text priority. Icon support.
- **Set Up Idle Mode Text Extended:** Icon support.
- **Send SM Extended: N**ew formats for the destination address with support for variable references. New alpha identifier for call set up phase. Icon support.
- **Swap nibbles:** Swapping the nibbles of an individual byte.
- **BCD to GSM 7bit Default Conversion:** Conversion of binary BCD data to readable text.
- **GSM 7bit Default to UCS2 Conversion:** Conversion from SMS default character set to UCS2 and vice versa.
- **Timer Management:** Timers in WIB.

Some of the existing (WIB 1.2) WIB commands have also been changed in a backward compatible way in WIB 1.3, to add more features.

- **Get Input:** Icon support.
- **Select Item:** Variable reference may now be used in the options. Icon support for each option as well as in the title.
- **Skip:** Two byte skip-length.
- **New Context:** Enables more fine grained control over which variables that should be cleared.
- **Send USSD:** Icon support and improved variable reference support.

Appendix E WIG compatibility information

This appendix lists issues related to backward compatibility of the WIG.

E.1 WIG 3.3 vs. WIG 3.2 (DP 6.0 vs. DP 5.2.3)

WIG 3.3 is backward compatible with WIG 3.2, with these exceptions:

- **New URL decoding of Push WIG WML documents:** Push documents will not be URL decoded anymore. Any push application depending on this has to be updated.
- **Restricted variable ID range:** The syntax `WIGVARxFF` does not work anymore for variable IDs in the range (0xE0..0xFF). For all other variable IDs, the syntax works as before.
- **Variables not supported in wigLaunchBrowser:** Variables in the URL for the SAT command `wigLaunchBrowser` are no longer supported.
- **Binary data in WIG WML:** The `&#xFF;` syntax can no longer be used in WIG WML documents for binary data. Instead the `\xFF` syntax shall be used.
- **&apos; not supported as before:** The `&apos;` character entity in WIG/WIB specific commands such as `wigSendSM` can not be used anymore. Instead `%27` shall be used for indicating the `'` character.
- **Cache and cookies:** A WIG WML document will now be fetched from cache although there are cookies associated with the request. The Content Provider is still in control, since the WIG will only cache a WIG WML document if the HTTP headers indicate that caching is allowed.
- **Cache and query string:** A WIG WML document will now be cached, although the URL contains a query string. The Content Provider is still in control, since the WIG will only cache a WIG WML document if the HTTP headers indicate that caching is allowed.

E.2 WIG 4.0 vs. WIG 3.3 (DP 6.1 vs. DP 6.0)

WIG 4.0 supports a new and redesigned version of the WIG WML language called WIG WML v4. The WIG still supports earlier versions of WIG WML, but it is recommended that all new WIB services are developed using WIG WML v4. Appendix A covers frequently asked questions concerning the rationale behind WIG WML v4.

Since WIG WML v4 is a new language, it is not meaningful to talk about backward compatibility. Instead Appendix B includes a migration guide for those who consider migration of their WIB services to WIG WML v4.

Document number: 50316003
Revision: A. 2003-12-03.