WAP Wireless Telephony Application Interface

Version 08-Sep-2001

Wireless Application Protocol
WAP-268-WTAI-20010908-a

.

A list of errata and updates to this document is available from the WAP Forum™ Web site, http://www.wapforum.org/, in the form of SIN documents, which are subject to revision or removal without notice.

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

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

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

Document History	
WAP-268-WTAI-20010715-p	Proposed
WAP-268-WTAI-20010908-a	Current

Contents

1. Scope

Wireless Application Protocol (WAP) is a result of continuous work to define an industry wide specification for developing applications that operate over wireless communication networks. The scope for the WAP Forum 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 WAP defines a set of protocols in transport, session and application layers. For additional information on the WAP architecture, refer to "*Wireless Application Protocol Architecture Specification"* [WAPARCH].

This document outlines the extensions to the WAP Application Environment (WAE) to support Wireless Telephony Applications. The specifics of the Wireless Telephony Applications are introduced in the form of an interface. The acronym WTAI is used in the document to denote the Wireless Telephony Application Interface. For maximum benefit, the reader should be somewhat familiar with WML [WML2] and WMLScript [WMLScript].

2. References

2.1. Normative References

[CREQ] WAP-221, “Specification of WAP Conformance Requirements”. WAP Forum™, 25-Apr-2001 URL: http//www.wapforum.org/

[FORMAT] WAP-188, "WAP General Formats Document", WAP Forum™, 10-Jul-2001 URL: http://www.wapforum.org/

[ISO8601] “Data elements and interchange formats – Information interchange - Representation of dates and times”, International Organization For Standardization (ISO), 15-June-1988

“Data elements and interchange formats - Information interchange - Representation of dates and times, Technical Corrigendum 1”, International Organization For Standardization (ISO) – Technical Committee ISO/TC 154, 01-May-1991

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

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax." T. Berners-Lee, R. Fielding, L. Masinter, et al., August 1998 URL: http://www.ietf.org/rfc/rfc2396.txt

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

[WMLScript] WAP-193, "WMLScript Language Specification", WAP Forum™, 25-Oct-2000 URL: http://www.wapforum.org/

[WMLScriptStdLibs] WAP-194, "WMLScript Standard Libraries Specification", WAP Forum™, 25-Sep-2000 URL: http://www.wapforum.org/

[WTA] WAP-266, "Wireless Telephony Application Specification", WAP Forum™, 08-Sep-2001 URL: http://www.wapforum.org/

2.2. Informative References

[WAPARCH] WAP-210, “WAP Architecture”, WAP Forum™, 12-Jul-2001 URL: http://www.wapforum.org/

[WML2] WAP-238, "Wireless Markup Language", WAP Forum™ URL: http://www.wapforum.org/

3. Terminology and Conventions

3.1. Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2. Definitions

Blocking Function - a function that completes all its actions prior to returning.

Card - a navigable part of a WML document (deck). May contain information to present on the screen, instructions for gathering user input, etc.

Client - a device (or application) that initiates a request for connection with a server.

Content - synonym for resources.

Deck - a WML document.

Device - a device is a network entity that is capable of sending and receiving packets of information and has a unique device address. A device can act as both a client and a server within a given context or across multiple contexts. For example, a device can service a number of clients (as a server) while being a client to another server.

Non-blocking Function - a function that completes its execution without waiting for the availability of a resource or the occurrence of an event.

Server - a device (or application) that passively waits for connection requests from one or more clients. A server may accept or reject a connection request from a client.

User - a user is a person who interacts with a user-agent to view, hear, or otherwise use a rendered content.

User Agent - a user-agent (or content interpreter) is any software or device that interprets WML, WMLScript or resources. This may include textual browsers, voice browsers, search engines, etc.

WML - the Wireless Markup Language is a hypertext markup language used to represent information for delivery to a narrowband device, eg a phone.

WMLScript - a scripting language used to program the mobile device. WMLScript is an extended subset of the JavaScript™ scripting language.

3.3. Abbreviations

API	Application Programming Interface
CGI	Common Gateway Interface
DTMF	Dual Tone Multi-Frequency
GSM	Global System for Mobile Communication
PCS	Personal Communications System
RFC	Request For Comments
SMS	Short Message Service
URI	Uniform Resource Identifier [RFC2396]
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAE	Wireless Application Environment [WAE]
WAP	Wireless Application Protocol [WAPARCH]
WTA	Wireless Telephony Applications [WTA]

WTAI	Wireless Telephony Applications Interface
WWW	World Wide Web

4. Introduction

The WAP WTAI features provide the means to create Telephony Applications, using a WTA user-agent with the appropriate WTAI function libraries. A typical example is to set-up a mobile originated call using the WTAI functions accessible from either a WML deck/card or WMLScript. The application model for WTA is described in [WTA].

4.1. WTAI Libraries

The WTAI features are partitioned into a collection of WTAI function libraries. The type of function and its availability determines where the different functions are specified. The WTAI function libraries are accessible from WMLScript using the scripting function libraries. Some WTAI functions are also accessible from WML using URIs (see Appendix B).

These functions may initiate an interaction between the mobile and the network. The function then typically terminates independently from the started network procedure. So any result delivered by the function call will not reflect the outcome of this procedure, which itself may result in events.

Example: A “user busy” condition is not reported by the return value of the “Setup Call” function but is delivered by the “call cleared” event.

Network Common WTAI The most common features that are available in all networks. They are only accessible from the WTA user-agent. Examples of functions are call setup and answer incoming call.

Network Specific WTAI Features that are only available in certain types of networks.

Public WTAI Simple features that are available to third party applications executing using the standard WAE user-agent.

5. Network Common Models

This section describes the different models that each of the Network Common WTAI Function Libraries follows.

5.1. Voice Call Model

The Voice Call Model specifies the voice call states, the voice call handle, the voice call mode and the voice call information. All Network Common - Voice call functions and related events MUST follow the specified Voice Call Model.

WTA services can place, receive, and terminate voice calls and can get information about voice calls. An implementation MAY support multiple simultaneous voice calls or MAY limit the WTA service to only one voice call at a time.

5.1.1. Voice Call States

The WTA call model is depicted in Figure 1 for an incoming voice call and in Figure 2 for an outgoing voice call. The call models show the call states and the events that result in state transitions. A voice call may stay in a particular state for an indefinite amount of time. The call models represent the lifetime of one call.

WTA implementations MUST generate WTA events according to these models. WTA implementations will generally rely on the underlying network signaling layer such that:

- network events correspond to zero or more WTA events
- the WTA implementation can support these call models without maintaining call state
- no WTA events need to be generated without an underlying network event.

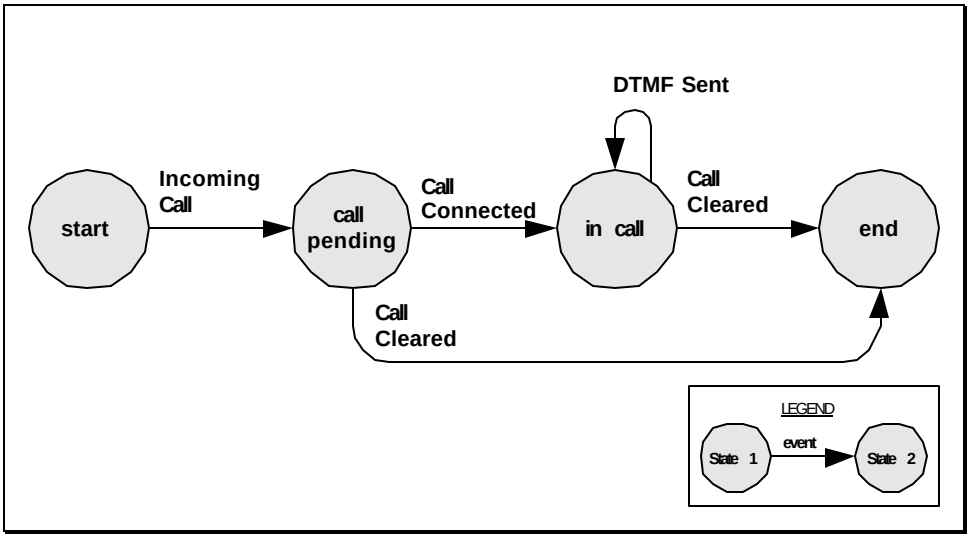

Figure 1 - Incoming Call Model

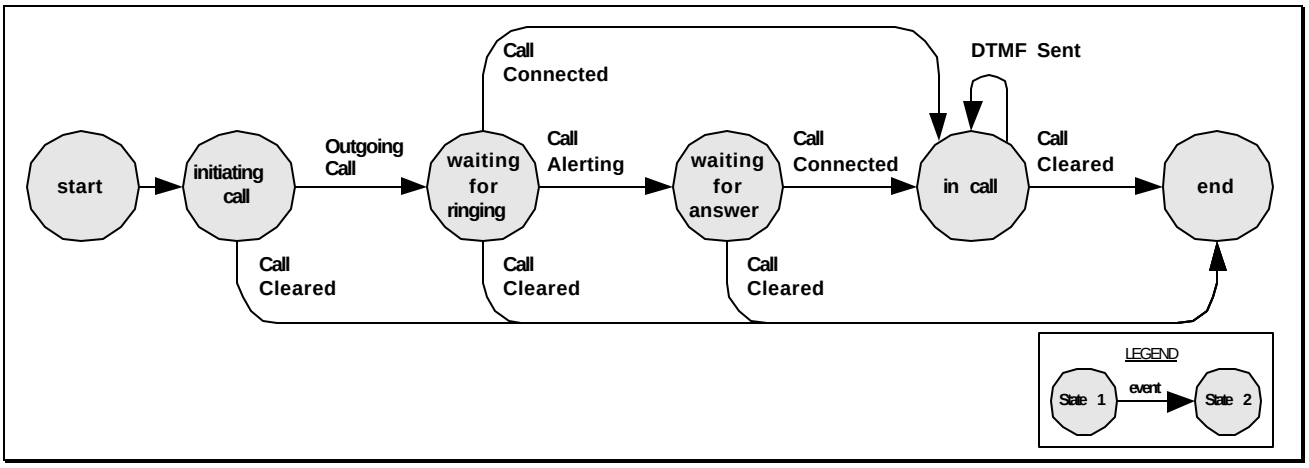

Figure 2 - Outgoing Call Model

5.1.2. Voice Call Handle

A handle is associated with each voice call. The call handle is a unique identifier for the call. WTAI functions that operate on voice calls require this call handle as an input parameter (e.g., WTAVoiceCall.accept) or return a call handle when successful (e.g., WTAVoiceCall.setup). A WTA service should keep track of the handles for its calls in order to control those calls; however, it should not inspect nor manipulate the call handle in any way.

All implementations MUST allocate call handles as follows:

- All call handles within a context MUST be unique, i.e., a specific value of a call handle must never be reused within the same context.
- The call handle and the call status information MUST be available until the end of WTA Call Cleared event processing.

The algorithm used to allocate a call handle is implementation specific and may be arbitrary. For the content author, this implies that:

- A call handle is assigned using some unspecified pattern. Authors should not rely on any specific sequence or characteristic of call handles.
- A call handle must not be compared to another call handle except to determine if they are equal, that is, if they refer to the same voice call.

5.1.3. Voice Call Modes

Authors must indicate the coupling between an established voice call and the current WTA context using the *mode* parameter of the setup() or accept() functions. (See [WTA].) The mode parameter indicates if the established voice call shold be dropped or kept when the WTA context is terminated.

5.1.4. Voice Call Information

The WTA user agent provides access to specific information about each voice call. Each information field has a name and value. A field value may be retrieved using its field name.

The following fields MUST be available for each voice call:

"number" `string` containing the number of the other party, specifically, a phone-number as defined in [FORMAT].

"status" `integer` indicating the recent state of the call in the WTA Incoming or Outgoing Call Model diagram (see Figure 1 and Figure 2). Note that this field value may not be completely accurate or up-to-date since the state of the call may change at any time. It MUST be one of the following values (all numbers are shown in decimal):

1 = the call is in the "call pending" state

2 = the call is in the "initiating call" state

3 = the call is in the "waiting for ringing" state

4 = the call is in the "waiting for answer" state

5 = the call is in the "in call" state

6 = the call is in the "end" state

"mode" `boolean` indicating whether the call was established in the "drop" or "keep" mode (see section 5.1.3):

false = drop

true = keep

The following additional fields MAY be available for each voice call:

"name" `string` containing the name of the other party or the value of the "number" field if no name exists.

"duration" `integer` containing the duration of the call in seconds. Note that this field value may not be completely accurate or up-to-date since the duration changes constantly for an ongoing call.

Other fields MAY exist within any given implementation, however, the field name MUST begin with "vnd-". Moreover, devices that implement network-specific WTAI functions MAY contain additional fields. (Refer to the WTAI addenda for GSM, PCS, etc.) Authors who wish to ensure proper operation of their content across all WTA devices should rely only on the mandatory fields listed above.

5.2. Network Message Model

The Network Message Model specifies the Network Message States, the Network Message handle and the Network Message information. All Network common WTAI - Network Message functions and related events MUST follow the specified Network Message Model.

WTA services can send and receive network messages (e.g GSM SMS) and can get information about network messages.

5.2.1. Network Message States

The WTA Network Message Model is depicted in Figure 3 for an incoming message and in Figure 4 for an outgoing message. The Network Message Models show the message states and transitions, and the events that result in state transitions. A network message may stay in a particular state for an indefinite amount of time. The Network Message Models represent the lifetime of one message.

WTA implementations MUST generate WTA events according to these models.

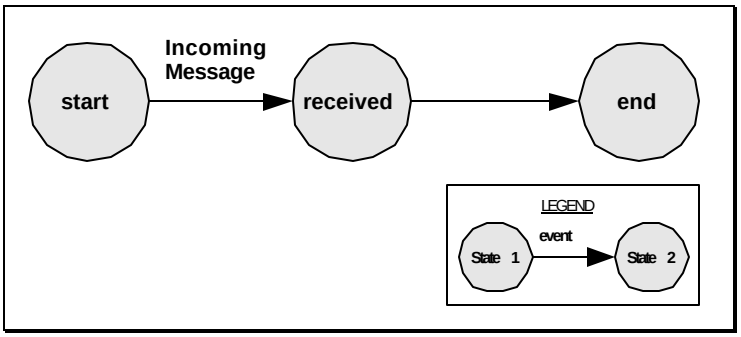

Figure 3 - Incoming Message Model

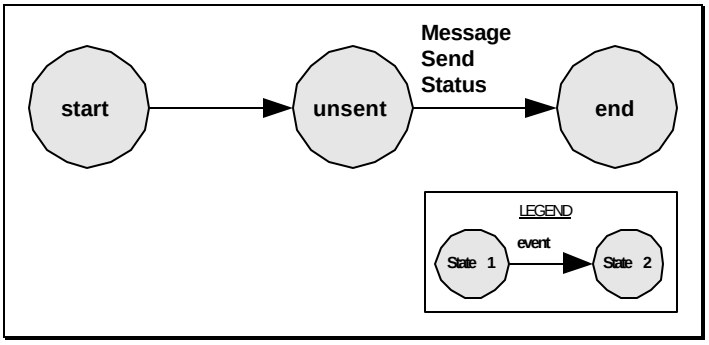

Figure 4 - Outgoing Message Model

5.2.2. Network Message Handle

A handle is associated with each network message. The message handle is a unique identifier for the message. WTAI functions that operate on network messages require this message handle as an input parameter (e.g., WTANetText.getFieldValue) or return a message handle when successful (e.g., WTANetText.send). A WTA service should keep track of the handles for its messages in order to track those messages; however, it should not inspect nor manipulate the message handle in any way.

All implementations MUST allocate message handle values as follows:

- All message handles within a context MUST be unique, i.e., a specific value of a message handle must never be reused within the same context.
- A message handle and the message status information MUST be available until the end of the MessageSendStatus event processing.

The algorithm used to allocate a message handle is implementation specific and may be arbitrary. For the content author, this implies that:

- A message handle is assigned using some unspecified pattern. Authors should not rely on any specific sequence or characteristic of message handles.
- A message handle must not be compared to another message handle except to determine if they are equal, that is, if they refer to the same network message.

5.2.3. Network Message Information

The WTA user agent provides access to specific information about each network message. Each information field has a name and value. A field value may be retrieved using its field name.

The following fields MUST be available for each message:

"text" `string` containing the body of the message.

"tstamp" `string` containing the timestamp field received with the message for an incoming message, or containing an empty `string` for an outgoing message.

"address" `string` containing the originating address for an incoming message or containing the destination address for an outgoing message.

"status" `integer` indicating the state of the message in the WTA Incoming or Outgoing Network Message Model diagram (see Figure 3 and Figure 4). Note that this field value may not be completely accurate or up-to-date since the state of the message may change at any time. It MUST be one of the following values (all numbers are shown in decimal):

1 = the message is in the "unsent" state

2 = the message is in the "received" state

3 = the message is in the "end" state

"read" `boolean` indicating whether the message is marked as read (if value is true) or unread (if value is false).

The following additional fields MAY be available for each message:

"tstamp_off" `integer` indicating the offset of the "tstamp" field from Co-ordinated Universal Time (UTC) in minutes.

"tstamp_device" `string` containing the date and time of the message as determined by the device in [ISO8601] format.

Other fields MAY exist within any given implementation, however, the field name MUST begin with "vnd-". Moreover, devices that implement network-specific WTAI functions MAY contain additional fields. (Refer to the WTAI addenda for GSM, PCS, etc.) Authors who wish to ensure proper operation of their content across all WTA devices should rely only on the mandatory fields listed above.

5.3. Phonebook Model

The Phonebook Model specifies Phonebook Storage and Phonebook Entry Information. All Network common WTAI – Phonebook functions MUST follow the specified Phonebook Model.

WTA services can access and modify the device's phone book.

5.3.1. Phonebook Storage

The WTA user agent considers the phonebook to be an ordered collection of phonebook entries, that is, an array of entries. Individual entries are accessed using an integer value that corresponds to the index of the entry within the array. Index values MUST be positive; a zero or negative index is not allowed.

5.3.2. Phonebook Entry Information

The WTA user agent provides access to specific information for each phonebook entry. Each information field has a name and value. A field value may be retrieved using its field name. All field values are of type `string`.

An entry may be empty; that is, it may contain no useful data. An entry where all the field values are the empty string is an empty entry.

The following fields MUST be available for each phonebook entry:

"number" `string` containing the phone number associated with the phonebook entry, specifically, a phone-number as defined in [FORMAT].

"name" `string` containing the name associated with the phonebook entry.

Note that there are no requirements on the format or contents of the "name" field. Authors should be careful when using this field, e.g. for searching or comparison, because:

- The "name" field may contain uppercase or lowercase characters only, or may contain a combination of uppercase and lowercase characters.
- The "name" field may present the surname and given name in either order, may contain only one of them, or may contain a nickname or mnemonic name.
- The "name" field may contain characters that might be considered equivalent for comparison purposes, for example the letters "e é ê ë e'", but that have unique character codes and hence can not be compared directly.

Other fields MAY exist within any given implementation, however, the field name MUST begin with "vnd-". Moreover, devices that implement network-specific WTAI functions MAY contain additional fields. (Refer to the WTAI addenda for GSM, PCS, etc.) Authors who wish to ensure proper operation of their content across all WTA devices should rely only on the mandatory fields listed above.

5.4. Call Log Model

The Call Log Model specifies the Call Log handle and Call Log Entry Information. All Network common WTAI – Call Log functions MUST follow the specified Call Log Model.

WTA services are provided access to a device's call history via the Call Log Model, the following Call Logs can be accessed:

Dialled Call Log history of outgoing voice calls

Missed Call Log history of incoming voice calls that were not answered

Received Call Log history of incoming voice calls that were answered

The call logs are accessed independently of each other; that is, there is no relationship between any two call logs.

5.4.1. Call Log handle

A handle is associated with each call log entry. The call log handle is a unique identifier for the call log entry. WTAI functions that operate on call log entries require this call log handle as an input parameter (e.g., WTACallLog.getFieldValue) or return a call log handle when successful (e.g., WTACallLog.dialled). A WTA service should not inspect nor manipulate the call log handle in any way.

All implementations MUST allocate call log handles as follows:

- All call log handles within a context MUST be unique, i.e., a specific value of a call log handle must never be reused within the same context.

The algorithm used to allocate a call log handle is implementation specific and may be arbitrary. For the content author, this implies that:

- A call log handle is assigned using some unspecified pattern. Authors should not rely on any specific sequence or characteristic of call log handles.
- A call log handle must not be compared to another call log handle except to determine if they are equal, that is, if they refer to the same call log entry.

5.4.2. Call Log Entry Information

The WTA user agent provides access to specific information for each call log entry. Each information field has a name and value. A field value may be retrieved using its field name.

The following field MUST be available for each call log entry:

"number" `string` containing the phone number of the far end party, specifically, a phone-number as defined in [FORMAT]. If the far end party's phone number can not be determined, this value contains an empty `string`.

The following additional fields MAY be available for each call log entry:

"tstamp" `string` containing the date and time that the entry was written to the call log in [ISO8601] format.

"explanation" `string` containing the reason why the phone number is not available in the "number" field or an empty `string` otherwise.

Other fields MAY exist within any given implementation, however, the field name MUST begin with "vnd-". Moreover, devices that implement network-specific WTAI functions MAY contain additional fields. (Refer to the WTAI addenda for GSM, PCS, etc.) Authors who wish to ensure proper operation of their content across all WTA devices should rely only on the mandatory fields listed above.

5.5. Logical Indicator Model

The Logical Indicator Model specifies Logical Inidcator Types and Logical Indicator States. The WTAI function WTAMisc.setIndicator MUST follow the specified Logical Indicator Model.

WTA services can access the device's logical indicators.

5.5.1. Logical Indicator Types

Access is provided to the following Logical Indicator Types:

Incoming Speech Call	indicates an incoming voice call is available to be answered
Incoming Data Call	indicates an incoming data call is available to be answered
Incoming Fax Call	indicates an incoming fax call is available to be answered
Call Waiting	indicates a second call is "on hold"
Text Message	indicates text messages are available
Voice Mail Message	indicates voice mail messages are available
Fax Message	indicates fax messages are available
Email Message	indicates email messages are available

A WTA implementation MUST provide access to all above listed Logical Indicator Types supported on the device.

The appearance of the logical indicators in the device is implementation dependent. For example, a logical indicator may be manifested as an LED, as an icon on the display, as a number on the display, or as a unique audible signal. Moreover, the logical indicator may not necessarily be conveyed in a unique manner within a device. For example, a device may convey two logical indicators using a single LED or icon.

Other logical indicators MAY exist within any given implementation. Moreover, devices that implement network-specific WTAI functions MAY contain additional logical indicators. (Refer to the WTAI addenda for GSM, PCS, etc.) Authors who wish to ensure proper operation of their content across all WTA devices should rely only on the logical indicators listed above.

5.5.2. Logical Indicator States

The state of a logical indicator is conveyed using an integer value as follows:

Incoming Speech Call	zero signifies the indicator is "off", positive values signify it is "on", and negative values are not allowed.
Incoming Data Call	zero signifies the indicator is "off", positive values signify it is "on", and negative values are not allowed.
Incoming Fax Call	zero signifies the indicator is "off", positive values signify it is "on", and negative values are not allowed.
Call Waiting	zero signifies the indicator is "off", positive values signify it is "on", and negative values are not allowed.
Text Message	zero indicates there are no messages, positive values indicate the number of messages, and negative values are not allowed.
Voice Mail Message	zero indicates there are no messages, positive values indicate the number of messages, and negative values are not allowed.
Fax Message	zero indicates there are no messages, positive values indicate the number of messages, and negative values are not allowed.
Email Message	zero indicates there are no messages, positive values indicate the number of messages, and negative values are not allowed.

These values are used to change or set the state of an indicator.

6. WTA Interface

6.1. WTAI Function Libraries

The WTAI functions are divided into libraries depending on type of function. A function library can also be specific to a certain type of network, and then a "well-known" network name is included in the name of the library. The WTAI specification defines the set of predefined WTAI function libraries for public and network common WTAI, listed below. Network specific WTAI function libraries are specified as addenda to the WTAI specification.

Table 1, Public WTAI Function Libraries

Function Library	*URI Name*	*Description of Library*
WTAPublic	“wp”	Public available WTAI functions.

Table 2, Network Common WTAI Function Libraries

Function Library	*URI Name*	*Description of Library*
WTAVoiceCall	-	Voice Call Control library. Handles call setup and control of device during an ongoing call.
WTANetText	-	Network Text library. Sending and retrieval of network text.
WTAPhoneBook	-	Phonebook library. Manages the entries in the device phonebook.
WTACallLog	-	Call Logs library. Used for accessing different kinds of call logs in the device.
WTAMisc	“ms”	Handling of miscellaneous features. An example is logical indications.

6.2. WTAI API Delimiters

The WTAI functions that are accessed using the WTAI URI scheme, a CGI-like invocation mechanism, pass all parameters as type `string` (see [WML2]).

Notations used for the WTAI syntax:

- < > Angle brackets denotes an enumerated parameter
- [ ] Square brackets denote an optional section.
- | Vertical bar denotes a pair of mutually exclusive options
- ( )* Repeat none or multiple times
- *( ) Repeat one or multiple times

Specification of parameters:

A general rule is to always specify all input and output parameters unless otherwise stated. The WTA user-agent SHOULD NOT fail if a result parameter is not specified. The recommended procedure in this instance is to discard the result.

6.3. WTAI URI Scheme

Access to some of the WTAI function libraries from WML can be handled through URI "calls" using the dedicated WTAI URI encoding scheme. Using a predefined reference to the specific WTAI function library together with the actual function name forms the WTAI URI. The WTAI URI library identifier can be used to identify the library. An example of a predefined library is "WTAPublic", specifying the public WTAI functions.

WTAI functions are named using URI's. URI's are defined in [RFC2396]. The character set used to specify URI's is also defined in [RFC2396]. Consequently characters such as space, used in a WTAI URI, MUST be escaped, see [RFC2396] for more details on escaping.

```
wtai://<library>/<function> (; <parameter>)* [! <result>]
```

Table 3, WTAI URI scheme

<library>	Name that identifies the type of function, i.e. WTAPublic uses the library name "wp".
<function>	Function identifier within specific library. An example is "mc" for the function "makeCall" residing in the library "WTAPublic".
<parameter>	Zero or more parameters to be sent to the function. Delimiter between subsequent parameters MUST be a semicolon ";".
<result>	Start of the result data section is indicated by an exclamation mark "!". Result an optional name of the variable that will be set in the WTA user-agent context as a result of the function invocation.

6.4. WTAI Notational Conventions

Each event is specified using the following information:

Event Name: The event mnemonic - a shorthand description of the event.

Example: CallCleared

Event ID: The event descriptor that is used in a channel definition or in a local binding for the event. Event IDs are case sensitive.

Example: wtaev-cc/cl

Parameters: A list of the event parameters.

Example: *callHandle, result*

Description: An explanation of the event behavior and of its parameters.

Each library in this document is specified using the following information:

Name: The library name. The syntax of the library name follows the syntax specified in the [WMLScript] specification. Library names are case sensitive.

Examples: WTAPublic, WTAVoiceCall

Library ID: The numeric identifier reserved for the library to be used by the WMLScript compiler.

Description: A short explanation of the library and used conventions.

Each function in a library that can be invoked using WMLScript is specified using the following information:

Function: The name of the function and a list of its input parameters. The syntax of the function name follows the syntax specified in the [WMLScript] specification. Function names are case sensitive.

Example: setup(*number*, *mode*)

Function ID: The numeric identifier reserved for the function to be used by the WMLScript compiler. The range of values reserved for this identifier is: 0..255.

Description: An explanation of the function's behavior and of its parameters.

Permission Types: A list of the user permission types that are supported for this function: "BLANKET", "CONTEXT", or "SINGLE" (see [WTA]).

Parameters: A list of the function parameter data types.

Example: *number* = `string`

Return value: A list of the possible data types returned by this function.

Example: `integer` or `invalid`

Associated Events: A list of the WTA events that may occur after (and as a direct result of) the function invocation.

Exceptions: A list of possible special exceptions and error conditions and the corresponding return values. Standard errors, common to all functions, are not described here.

Example: If *mode*!=0 and *mode*!=1 then `invalid` is returned.

Example: Sample content showing how the function could be used.

Each function in a library that can be invoked using a URI is specified using the following information:

Function: The name of the function and a list of its input parameters. The syntax of the function name follows the syntax specified in section 6.3. Function names are case sensitive.

Example: wtai://wp/mc;number!result

Description: An explanation of the function's behavior and of its parameters.

Permission Types: A list of the user permission types that are supported for this function: "BLANKET", "CONTEXT", or "SINGLE" (see [WTA]).

Parameters: A list of the function parameter data types.

Example: *number* = `string`

Result value: A list of the possible data types returned by this function.

Example: `string`

Associated Events: A list of the WTA events that may occur after (and as a direct result of) the function invocation.

Exceptions: A list of possible special exceptions and error conditions and the corresponding return values. Standard errors, common to all functions, are not described here.

Example: If *mode*!=0 and *mode*!=1 then an empty `string` is returned.

Example: Sample content showing how the function could be used.

6.5. Telephone Numbers

The format of telephone numbers and DTMF sequences valid for use in WTAI functions is specified in [FORMAT].

7. Compliance Requirements

WTAI library functions MUST follow the WMLScript standard library conventions and rules. The WMLScript standard library compliance requirements specified in [WMLScriptStdLibs] MUST also be followed for all WTAI library functions.

In addition, the WTAI library functions use the term `handle` as an alias for positive integer, and the term `error-code` as an alias for negative integer. Integer is defined in [WMLScriptStdLibs].

8. Public WTAI

8.1. WTA Events

There are no events associated with this library. No events are generated as a direct or indirect result of invoking functions in this library.

8.2. WMLScript functions

WTA implementations MUST support all the Public WTAI WMLScript functions specified in this chapter.

Name: WTAPublic

Library ID: 512

Description: This library contains functions that are available to all services, including untrusted services.

8.2.1. WTAPublic.makeCall

Function: makeCall(*number*)

Function ID: 0

Description: Initiates a mobile-originated voice call; the call must be terminated using the standard MMI. This function is blocking. No WTA events are generated by the voice call initiated using this function.

The *number* parameter specifies the destination to call and must be a phone-number as defined in [FORMAT].

This function returns an empty `string` if the call was established successfully, returns an `error-code` that indicates an error in establishing the call, or returns `invalid` if this function fails.

Permission Types: SINGLE (see [WTA]). The *number* parameter must be displayed to the user when obtaining permission.

Parameters: *number* = `string` (phone-number)

Return value: empty `string` or `invalid` or `error-code` (must be one of the following decimal values:

-105 = called party is busy

-106 = network is not available

-107 = called party did not answer

-1 = unspecified error)

Associated Events: -

Exceptions: If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns `invalid`.

Example: var flag = WTAPublic.makeCall("5554367");

8.2.2. WTAPublic.sendDTMF

Function: sendDTMF(*dtmf*)

Function ID: 1

Description: Sends a DTMF sequence through the voice call most recently created using the WTAPublic.makeCall or wtai://wp/mc function. This function is blocking. No WTA events are generated as a direct or indirect result of invoking this function.

The *dtmf* parameter specifies the DTMF sequence to be sent and must be a `tone_sequence` as defined in [FORMAT].

This function returns an empty `string` if the DTMF sequence was sent successfully, returns an `error-code` that indicates an error in sending the DTMF sequence, or returns `invalid` if the function fails.

Permission Types: CONTEXT, SINGLE (see [WTA]).

Parameters: *dtmf* = `string` (`tone_sequence`)

Return value: empty `string` or `invalid` or `error-code` (must be one of the following decimal values:

-108 = no active voice connection

-1 = unspecified error)

Associated Events: -

Exceptions: If the *dtmf* parameter is not a `tone_sequence` as defined in [FORMAT], this function returns `invalid`.

Example: var flag = WTAPublic.sendDTMF("555*1234");

8.2.3. WTAPublic.addPBEntry

Function: addPBEntry(*number*,*name*)

Function ID: 2

Description: Writes a new phonebook entry.

The *number* parameter specifies the phone number to associate with the entry and must be a phone-number as defined in [FORMAT].

The *name* parameter specifies the name to associate with the entry and may be an empty `string`.

This function returns an empty `string` if the phonebook entry was added successfully, returns an `error-code` that indicates an error in adding the phonebook entry, or returns `invalid` if the function fails.

Permission Types: SINGLE (see [WTA]). The *name* and *number* parameters must be displayed to the user when obtaining permission.

Parameters: *number* = `string` (phone-number)

name = `string`

Return value: empty `string` or `invalid` or `error-code` (must be one of the following decimal values:

-100 = *name* parameter is unacceptable or too long

-102 = *number* `parameter` is too long

-103 = phonebook entry could not be written

-104 = phonebook is full

-1 = unspecified error)

Associated Events: -

Exceptions: If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns `invalid`.

Example: var flag = WTAPublic.addPBEntry("5554367", "EINSTEIN");

8.3. URI functions

WTA implementations MUST support all the Public WTAI URI functions specified in this chapter.

8.3.1. wtai://wp/mc

Function: wtai://wp/mc;*number*!*result*

Description: Initiates a mobile-originated voice call. This function is blocking. No WTA events are generated by the voice call initiated using this function.

The *number* parameter specifies the destination to call and must be a phone-number as defined in [FORMAT].

The *result* parameter specifies the variable name that will hold the result of the function invocation, i.e., whether or not the voice call was established.

Permission Types: SINGLE (see [WTA]). The *number* parameter must be displayed to the user when obtaining permission.

Parameters: *number* = `string` (phone-number)

Result value: "" = call was established successfully

"-105" = called party is busy

"-106" = network is not available

"-107" = called party did not answer

"-200" = invocation error

"-1" = unspecified error

Associated Events: -

Exceptions: If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns an invocation error.

Example: <go href="wtai://wp/mc;5554367!resultvar"/>

8.3.2. wtai://wp/sd

Function: wtai://wp/sd;*dtmf*!*result*

Description: Sends DTMF sequence through the voice call most recently created using the WTAPublic.makeCall or wtai://wp/mc function. This function is blocking. No WTA events are generated as a direct or indirect result of invoking this function.

The *dtmf* parameter specifies the DTMF sequence to be sent and must be a `tone_sequence` as defined in [FORMAT].

The *result* parameter specifies the variable name that will hold the result of the function invocation, i.e., whether or not the DTMF was sent.

Permission Types: CONTEXT, SINGLE (see [WTA]).

Parameters: *dtmf* = `string` (`tone_sequence`)

Result value: "" = DTMF sequence was sent

"-108" = could not send DTMF because no active voice connection

"-200" = invocation error

"-1" = unspecified error

Associated Events: -

Exceptions: If the *dtmf* parameter is not a tone_sequence as defined in [FORMAT], this function returns an invocation error.

Example: <go href="wtai://wp/sd;555*1234!resultvar"/>

8.3.3. wtai://wp/ap

Function: wtai://wp/ap;*number*;*name!result*

Description: Writes a new phonebook entry.

The *number* parameter specifies the phone number to associate with the entry and must be a phone-number as defined in [FORMAT].

The *name* parameter specifies the name to associate with the entry and may be an empty string.

The *result* parameter specifies the variable name that will hold the result of the function invocation, i.e., whether or not the phonebook entry was added.

Permission Types: SINGLE (see [WTA]). The *name* and *number* parameters must be displayed to the user when obtaining permission.

Parameters: *number* = string (phone-number)

name = string

Result value: "" = phonebook entry was added

"-100" = *name* parameter is unacceptable or too long

"-102" = *number* parameter is too long

"-103" = phonebook entry could not be written

"-104" = phonebook is full

"-200" = invocation error

"-1" = unspecified error

Associated Events: -

Exceptions: If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns an invocation error.

Example: <go href="wtai://wp/ap;5554367;EINSTEIN!resultvar"/>

9. Network Common WTAI - Voice Calls

9.1. WTA Events

These events are related to voice calls. All WTA event parameters are conveyed as strings. WTA implementations MUST support all the Network Common WTAI – Voice Calls events specified in this chapter.

9.1.1. wtaev-cc/ic

Event Name: IncomingCall

Event ID: wtaev-cc/ic

Parameters: *callHandle*, *callerId*

Description: Indicates an incoming voice call has reached the WTA user agent. The voice call may be answered using WTA VoiceCall.accept or may be rejected using WTAVoiceCall.release.

The *callHandle* parameter contains the call handle for the incoming call as defined in section 5.1.2.

The *callerId* parameter contains the calling party's phone number and must be a phone-number as defined in [FORMAT]. If the calling party's phone number can not be determined, the *callerId* parameter must contain an empty `string`.

9.1.2. wtaev-cc/cl

Event Name: CallCleared

Event ID: wtaev-cc/cl

Parameters: *callHandle*, *result*

Description: Indicates a voice call has been cleared.

The *callHandle* parameter contains the call handle for the disconnected call as defined in section 5.1.2.

The *result* parameter contains a description of why the call was cleared. It must be one of the following values:

- "0" = normal termination of a voice call, e.g., the near or far end released the voice call
- "1" = unspecified, no details available
- "2" = network-specific reason
- "3" = voice call was dropped, e.g., because of a loss of signal
- "4" = called party was busy
- "5" = network is not available
- "6" = called party did not answer

9.1.3. wtaev-cc/co

Event Name: CallConnected

Event ID: wtaev-cc/co

Parameters: *callHandle*, *callerId*

Description: Indicates an incoming or outgoing voice call has been established.

The *callHandle* parameter contains the call handle for the call as defined in section 5.1.2.

The *callerId* parameter contains the far end party's phone number and must be a phone-number as defined in [FORMAT]. If the far end party's phone number can not be determined, the *callerId* parameter must contain an empty `string`.

9.1.4. wtaev-cc/oc

Event Name: OutgoingCall

Event ID: wtaev-cc/oc

Parameters: *callHandle*, *number*

Description: Indicates an outgoing voice call is being set up.

The *callHandle* parameter contains the call handle for the outgoing call as defined in section 5.1.2.

The *number* parameter contains the called party's phone number and must be a phone-number as defined in [FORMAT].

9.1.5. wtaev-cc/cc

Event Name: CallAlerting

Event ID: wtaev-cc/cc

Parameters: *callHandle*

Description: Indicates an outgoing voice call is ringing at the called party.

The *callHandle* parameter contains the call handle for the outgoing call as defined in section 5.1.2.

9.1.6. wtaev-cc/dtmf

Event Name: DTMFSent

Event ID: wtaev-cc/dtmf

Parameters: *callHandle*, *dtmf*

Description: Indicates the DTMF sequence has been sent on a voice call. A sequence of invocations of WTAVoiceCall.sendDTMF() will generate a matching sequence of DTMFSent events in order, except in cases when the DTMF can not be sent, such as if the call is cleared.

The *callHandle* parameter contains the call handle for the call as defined in section 5.1.2.

The *dtmf* parameter contains the DTMF sequence that was sent and must be a `tone_sequence` as defined in [FORMAT].

9.2. WMLScript Functions

WTA implementations MUST support all the Network Common WTAI – Voice Calls WMLScript functions specified in this chapter.

Name: WTAVoiceCall

Library ID: 513

Description: This library contains functions that are related to voice call control and that are common for all networks implementing WTA.

9.2.1. WTAVoiceCall.setup

Function: setup(*number*,*mode*)

Function ID: 0

Description: Initiates a mobile-originated call. This function is non-blocking. Subsequent WTA events signal the call progress.

The *number* parameter specifies the destination to call and must be a phone-number as defined in [FORMAT].

The *mode* parameter indicates how the call should be handled if the context in the WTA user agent terminates as defined in section 5.1.3.

This function returns the call `handle` of the new voice call if successful or returns `invalid` if the function fails. (See section 5.1.2 for a description of the call handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *number* = `string` (phone-number)

mode = `boolean` (false=drop, true=keep)

Return value: `handle` or `invalid`

Associated Events: An OutgoingCall event occurs if the call is being established.

A CallCleared event occurs if the call can not be established.

Exceptions: If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns `invalid`.

Example: var handle = WTAVoiceCall.setup("5554367",true);

9.2.2. WTAVoiceCall.accept

Function: accept(*callHandle*,*mode*)

Function ID: 1

Description: Accepts an incoming voice call that is waiting to be answered; this is the equivalent of lifting the handset. This function is non-blocking. Subsequent WTA events signal the call progress.

The *callHandle* parameter identifies which call is to be answered. (See section 5.1.2 for a description of the call handle.)

The *mode* parameter indicates how the call should be handled if the context in the WTA user agent terminates as defined in section 5.1.3.

This function returns an empty `string` if successful, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *callHandle* = `handle`

mode = `boolean` (false=drop, true=keep)

Return value: empty `string` or `invalid`

Associated Events: A CallConnected event occurs if the call is established.

A CallCleared event occurs if the call can not be established.

Exceptions: If the *callHandle* parameter does not refer to an existing voice call that is waiting to be answered, this function returns `invalid`.

Example: var flag = WTAVoiceCall.accept(handle,false);

9.2.3. WTAVoiceCall.release

Function: release(*callHandle*)

Function ID: 2

Description: Releases a voice call. This function is non-blocking. Subsequent WTA events signal that the call has ended.

The *callHandle* parameter identifies which call is to be released. (See section 5.1.2 for a description of the call handle.)

This function returns an empty `string` if successful, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *callHandle* = `handle`

Return value: empty `string` or `invalid`

Associated Events: A CallCleared event occurs when the call terminates.

Exceptions: If the *callHandle* parameter does not refer to a voice call that can be released, this function returns `invalid`.

Example: var flag = WTAVoiceCall.release(handle);

9.2.4. WTAVoiceCall.sendDTMF

Function: sendDTMF(*callHandle*,*dtmf*)

Function ID: 3

Description: Sends a DTMF sequence through a voice call. This function is non-blocking. Subsequent WTA events signal that the DTMF has been sent on the voice call.

The *callHandle* parameter identifies which call is to receive the DTMF. (See section 5.1.2 for a description of the call handle.)

The *dtmf* parameter specifies the DTMF sequence to be sent and must be a `tone_sequence` as defined in [FORMAT].

This function returns an empty `string` if successful or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *callHandle* = `handle`

dtmf = `string` (`tone_sequence`)

Return value: empty `string` or `invalid`

Associated Events: Exactly one DTMFSent event occurs when the DTMF sequence has been sent on the voice call.

Exceptions: If the *callHandle* parameter does not refer to a voice call through which DTMF may be sent, this function returns `invalid`.

If the *dtmf* parameter is not a `tone_sequence` as defined in [FORMAT], this function returns `invalid`.

Example: var flag = WTAVoiceCall.sendDTMF(handle,"555*1234");

9.2.5. WTAVoiceCall.callStatus

Function: callStatus(*callHandle*,*field*)

Function ID: 4

Description: Retrieves information about a specific voice call.

The *callHandle* parameter identifies the call of interest. (See section 5.1.2 for a description of the call handle.)

This function returns a value that depends on the *field* parameter. Any string may be given for the *field* parameter, however, if the *field* parameter matches one of the field names specified in section 5.1.4, the corresponding information must be returned as specified in that section. If the *field* parameter is unrecognized, unsupported, or otherwise unavailable, an empty `string` is returned.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *callHandle* = `handle`

field = `string` (field name)

Return value: any except `invalid` (field value), or `invalid` (failure indication)

Associated Events: -

Exceptions: If the *callHandle* parameter does not refer to a voice call for which call information is available, this function returns `invalid`.

Example: var name = WTAVoiceCall.callStatus(handle, "name");

9.2.6. WTAVoiceCall.list

Function: list(*returnFirst*)

Function ID: 5

Description: Returns the call handle of a call that can be controlled within the WTA context that invoked the function. The function is called repeatedly to retrieve information about all such calls. Note that this function may return a call that is no longer active, for example, to allow a WTA service to determine the call duration after it has ended.

The *returnFirst* parameter indicates which call handle should be returned. The call handles are returned in a list-traversal manner in the order in which the call handles were allocated, i.e., oldest call first. When the *returnFirst* parameter is true, the return value is the handle for the oldest call or invalid if there are no existing calls. When the *returnFirst* parameter is false, the return value is the handle of the next oldest call following the previously-returned one, or is invalid if there are no calls younger than the previously-returned one.

Note: It is possible that subsequent invocations can return a call handle if subsequent voice calls are established.

This function returns a call `handle` if successful or returns `invalid` if the end of the list was reached. (See section 5.1.2 for a description of the call handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *returnFirst* = `boolean` (true=return oldest call handle, false=return next-oldest call handle)

Return value: `handle` or `invalid`

Associated Events: -

Exceptions: -

Example:

```
var h1 = WTAVoiceCall.list(true);  // get call #1
var h2 = WTAVoiceCall.list(false); // get call #2
var h3 = WTAVoiceCall.list(false); // get call #3
var hA = WTAVoiceCall.list(true);  // get call #1
var hB = WTAVoiceCall.list(false); // get call #2
```

10. Network-Common WTAI - Network Messages

10.1. WTA Events

These events are related to network messages. All WTA event parameters are conveyed as strings. WTA implementations MUST support all the Network Common WTAI – Network Messages events specified in this chapter.

10.1.1. wtaev-nt/st

Event Name: MessageSendStatus

Event ID: wtaev-nt/st

Parameters: *msgHandle, result*

Description: Indicates the status of an outgoing network message has changed.

The *msgHandle* parameter contains the message handle for the outgoing message as defined in section 5.2.2.

The *result* parameter contains a description of the change in the outgoing message. It must be one of the following values:

"0" = message has been sent

"1" = message has been abandoned for unspecified reason

"2" = message has been abandoned because network is not available

"3" = message has been abandoned because of insufficient resources

10.1.2. wtaev-nt/it

Event Name: IncomingMessage

Event ID: wtaev-nt/it

Parameters: *msgHandle*, *sender*

Description: Indicates an incoming network message has been received.

The *msgHandle* parameter contains the message handle for the incoming message as defined in section 5.2.2.

The *sender* parameter contains the address of the originator of the message, that is, the value of the "address" field of the network message information as defined in 5.2.3. If the originator of the message is not known, the *sender* parameter contains an empty `string`.

10.2. WMLScript functions

WTA implementations MUST support all the Network Common WTAI – Network Messages WMLScript functions specified in this chapter.

Name: WTANetText

Library ID: 514

Description: This library contains functions that are related to network messages and that are common for all networks implementing WTA.

10.2.1. WTANetText.send

Function: send(*address*,*text*)

Function ID: 0

Description: Sends a network message. This function is non-blocking. Subsequent WTA events signal the message transmission progress.

The *address* parameter specifies the destination of the message and must be a phone-number as defined in [FORMAT]. It becomes the "address" field of the network message.

The *text* parameter specifies the text to send. It becomes the "text" field of the network message.

This function returns the message `handle` if successful, returns an `error-code` under certain conditions, or returns `invalid` if the function fails. (See section 5.2.2 for a description of the message handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *address* = `string` (phone-number)

text = `string` (message body)

Return value: `handle` or `invalid` or `error-code`

`error-code` must be one of the following decimal values:

-100 = *text* parameter is too long

-1 = unspecified error

Associated Events: A MessageSendStatus event occurs when the message enters the "end" state.

Exceptions: If the *address* parameter is not a phone-number as defined in [FORMAT], this function returns `invalid`.

If the *text* parameter contains an unacceptable character, this function returns `invalid`.

Example: var handle = WTANetText.send("5554567", "Hello!");

10.2.2. WTANetText.list

Function: list(*returnFirst*,*messageType*)

Function ID: 1

Description: Returns the message handle of an existing message. The function is called repeatedly to retrieve information about all such existing messages. This function does not change the value of the "read" field of any message.

The *returnFirst* parameter indicates which message handle should be returned. The message handles are returned in a list-traversal manner in the order in which the message handles were allocated, i.e., oldest handle first. When the *returnFirst* parameter is true, the return value is the handle for the oldest handle or invalid if there are no existing messages. When the *returnFirst* parameter is false, the return value is the handle of the next oldest handle following the previously-returned one, or is invalid if there are no handles younger than the previously-returned one.

The *messageType* parameter indicates which messages to include in the list. The *messageType* parameter is ignored if the *returnFirst* parameter is false; in that case the previous search is continued.

This function returns a message `handle` if successful or returns `invalid` if the function fails or if the end of the list was reached. (See section 5.2.2 for a description of the message handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *`returnFirst`* = `boolean` (must be one of the following values:

true=return oldest message handle

false=return "next" message handle)

`messageType` = `integer` (must be one of the following decimal values:

0 = include all read, unread and unsent messages

1 = include only messages which are unread

2 = include only messages which are read

3 = include only messages which are unsent

Return value: `handle` or `invalid`

Associated Events: -

Exceptions: If the *`messageType`* parameter is not acceptable, this function returns `invalid`.

Example:

```
var hi1 = WTANetText.list(true,0);  // get any message #1
var hi2 = WTANetText.list(false,0); // get any message #2
var hi3 = WTANetText.list(false,1); // get any message #3 (second parameter is don't care)
var ha1 = WTANetText.list(true,1); // get unread message #1
var ha2 = WTANetText.list(false,1); // get unread message #2
```

10.2.3. WTANetText.remove

Function: remove(*`msgHandle`*)

Function ID: 2

Description: Permanently removes an incoming or outgoing network message from the device.

The *`msgHandle`* parameter identifies which message is to be removed. (See section 5.2.2 for a description of the message handle.)

This function returns an empty `string` if the function is successful, returns an `error-code` under certain conditions, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *`msgHandle`* = `handle`

Return value: empty `string` or `invalid` or `error-code` (must be one of the following decimal values:

-101 = could not remove message

-1 = unspecified error)

Associated Events: -

Exceptions: If the *`msgHandle`* parameter does not refer to an existing network message, i.e., one that can be deleted, this function returns `invalid`.

Example:

```
var flag = WTANetText.remove(handle);
```

10.2.4. WTANetText.getFieldValue

Function: getFieldValue(*msgHandle*,*field*)

Function ID: 3

Description: Retrieves a field value from a specific message.

The *msgHandle* parameter identifies the message of interest. (See section 5.2.2 for a description of the message handle.)

The *field* parameter specifies the name of the field to retrieve from the network message info. (See section 5.2.3 for a description of the network message fields.)

This function returns a value that depends on the *field* parameter. Any string may be given for the *field* parameter, however, if the *field* parameter matches one of the field names specified in section 5.2.3, the corresponding information must be returned as specified in that section. If the *field* parameter is unrecognized, unsupported, or otherwise unavailable, an empty `string` is returned.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *msgHandle* = `handle`

field = `string` (field name)

Return value: any except `invalid` (field value), or `invalid` (failure indication)

Associated Events: -

Exceptions: If the *msgHandle* parameter does not refer to an existing network message, this function returns `invalid`.

Example: var addr = WTANetText.getFieldValue(handle, "address");

10.2.5. WTANetText.markAsRead

Function: markAsRead(*msgHandle*)

Function ID: 4

Description: Marks a message as read.

The *msgHandle* parameter identifies the message of interest. (See section 5.2.2 for a description of the message handle.)

This function returns an empty `string` if the function is successful or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *msgHandle* = `handle`

Return value empty `string` or `invalid`

Associated Events: -

Exceptions: If the *msgHandle* parameter does not refer to an existing network message, this function returns `invalid`.

Example: var flag = WTANetText.markAsRead(handle);

11. Network-Common WTAI - Phonebook

11.1. WTA Events

There are no events associated with this library. No events are generated as a direct or indirect result of invoking functions in this library.

11.2. WMLScript functions

WTA implementations MUST support all the Network Common WTAI – Phonebook WMLScript functions specified in this chapter.

Name: WTAPhoneBook

Library ID: 515

Description: This library contains functions that are related to the phonebook and that are common for all networks implementing WTA.

11.2.1. WTAPhoneBook.write

Function: write(*index*,*number*,*name*)

Function ID: 0

Description: Writes a phonebook entry, overwriting any existing entry.

The *index* parameter specifies the location of the entry within the phonebook. (See section 5.3.1 for a description of the phonebook index.) If the *index* parameter is zero, then the entry is stored in a location that is currently empty; the choice of location is implementation dependent.

The *number* parameter specifies the phone number to associate with the entry and must be a phone-number as defined in [FORMAT]. It becomes the "number" field of the phonebook entry.

The *name* parameter specifies the name to associate with the entry and may be an empty `string`. It becomes the "name" field of the phonebook entry.

This function returns the index where the entry was stored if successful, or returns an `error-code` under certain conditions, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *index* = `integer`

number = `string` (phone-number)

name = `string` (associated name)

Return value: `integer` or `invalid` or `error-code`.

`error-code` must be one of the following decimal values:

-100 = *name* parameter contains unacceptable characters or is too long

-102 = *number* parameter is too long

-103 = entry could not be written

-104 = phone book is full

-1 = unspecified error

Associated Events: -

Exceptions: If the *index* parameter is unacceptable, e.g., does not refer to a valid phonebook entry location, this function returns `invalid`.

If the *number* parameter is not a phone-number as defined in [FORMAT], this function returns `invalid`.

Example: var index = WTAPhoneBook.write(0,"5554367", "A. Einstein");

11.2.2. WTAPhoneBook.search

Function: search(*field*,*value*)

Function ID: 1

Description: Returns the index of a phonebook entry that matches the given search criteria. The index of an empty phonebook entry is never returned.

The *field* parameter specifies which field of the phone book entry to inspect.

The *value* parameter specifies the desired value of that field, where the empty string is a wildcard.

To start a search, this function must be invoked with a non-empty string for the *field* parameter. If the *value* parameter is a non-empty string, the search is initialized to return the index of all phone book entries containing that string anywhere within the field value, using case-insensitive comparison. If the *value* parameter is an empty string, the search is initialized to return the index of all non-empty phone book entries that have a non-empty string for the value of the specified field.

NOTE: All implementations MUST perform at least a literal string comparison between the *value* parameter and the phonebook entry field value. However, an implementation MAY use an enhanced string comparison algorithm, such as one that matches characters with diacritical marks to unadorned characters (e.g., "e" matches "e é ê ë e'").

To continue a search, this function must be invoked with an empty string for the *field* and *value* parameters.

Each invocation of this function, including the one that initializes the search, returns an index of a phonebook entry that meets the search criteria.

This function returns the next index of a phonebook entry that matches the search criteria, or returns `invalid` if there is not another search result (e.g., the end of the list was reached) or if the function fails.

Note: This function is intended for use within a loop with a non-empty string for the *field* parameter on the first invocation, with the empty string for the input parameters of each subsequent time through the loop, and with the loop exiting when `invalid` is returned.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *field* = `string` (field name or empty)

value = `string` (field value or empty)

Return value: `integer` or `invalid`

Associated Events: -

Exceptions: If the *field* parameter is the empty `string` and the *value* parameter is not (i.e., the function is invoked illegally), this function returns `invalid`.

If the *field* parameter is unrecognized, unsupported, or otherwise unavailable, this function returns `invalid`.

If this function is invoked to continue a search without the search criteria first having been initialized, then this function returns `invalid`.

Example: var index = WTAPhoneBook.search("name", "alb");

11.2.3. WTAPhoneBook.remove

Function: remove(*`index`*)

Function ID: 2

Description: Removes a phonebook entry, that is, makes the phonebook entry an empty entry.

The *`index`* parameter specifies the location of the entry within the phonebook. (See section 5.3.1 for a description of the phonebook index.)

This function returns an empty `string` if the function is successful, returns an `error-code` under certain conditions, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *`index`* = `integer` (index of location to remove)

Return value: empty `string` or `invalid` or `error-code`.

`error-code` must be one of the following decimal values:

-105 = could not remove phonebook entry

-1 = unspecified error

Associated Events: -

Exceptions: If the *`index`* parameter is unacceptable, e.g., does not refer to a valid, non-empty phonebook entry, this function returns `invalid`.

Example: var flag = WTAPhoneBook.remove(index);

11.2.4. WTAPhoneBook.getFieldValue

Function: getFieldValue(*`index`*,*`field`*)

Function ID: 3

Description: Retrieves a field value from a specific phonebook entry.

The *`index`* parameter specifies the location of the entry within the phonebook. (See section 5.3.1 for a description of the phonebook index.)

The *`field`* parameter specifies the name of the field to retrieve from the phonebook entry info. (See section 5.3.2 for a description of the phonebook entry fields.)

This function returns a value that depends on the *`field`* parameter. Any string may be given for the *`field`* parameter, however, if the *`field`* parameter matches one of the field names specified in section 5.3.2, the corresponding information must be returned as specified in that section. If the *`field`* parameter is unrecognized, unsupported, or otherwise unavailable, an empty `string` is returned.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *`index`* = `integer` (index of entry of interest)

`field` = `string` (name of field to retrieve)

Return value: any except `invalid` (field value), or `invalid` (failure indication)

Associated Events: -

Exceptions: If the *index* parameter is unacceptable, e.g., does not refer to a valid, non-empty phonebook entry, this function returns `invalid`.

Example: var number = WTAPhoneBook.getFieldValue(index, "number");

11.2.5. WTAPhoneBook.change

Function: change(*index*,*field*,*newValue*)

Function ID: 4

Description: Store the given value in the specified field in a phonebook entry, overwriting the existing value if any.

The *index* parameter specifies the location of the entry within the phonebook. (See section 5.3.1 for a description of the phonebook index.)

The *field* parameter specifies the name of the field to change within the phonebook entry. (See section 5.3.2 for a description of the phonebook entry fields.)

The *newValue* parameter specifies the new value for the specified field within the phonebook entry. (See section 5.3.2 for a description of the phonebook entry fields.)

This function returns an empty `string` if the function is successful, returns an `error-code` under certain conditions, or returns `invalid` if the function fails.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *index* = `integer` (location of entry)

field = `string` (name of field to change)

newValue = `string` (new value for the field)

Return value: empty `string` or `invalid` or `error-code`

`error-code` must be one of the following decimal values:

- -100 = *field* parameter is "name" and *newValue* contains an unacceptable character or is too long
- -101 = *field* parameter is "number" and *newValue* is not a phone-number
- -102 = *field* parameter is "number" and *newValue* is too long
- -103 = entry could not be changed
- -104 = phonebook is full
- -106 = *field* parameter is not supported
- -107 = *field* parameter is not "name" or "number" and *newValue* is unacceptable or is too long
- -1 unspecified error

Associated Events: -

Exceptions: If the *index* parameter is unacceptable, e.g., does not refer to a valid, non-empty phonebook entry location, this function returns `invalid`.

Example: var flag = WTAPhoneBook.change(index, "name", "ALBERT");

12. Network-Common WTAI - Call Logs

12.1. WTA Events

There are no events associated with this library. No events are generated as a direct or indirect result of invoking functions in this library.

12.2. WMLScript functions

WTA implementations MUST support all the Network Common WTAI – Call Logs WMLScript functions specified in this chapter.

Name: WTACallLog

Library ID: 519

Description: This library contains functions that are related to the history of voice calls and that are common for all networks implementing WTA.

12.2.1. WTACallLog.dialled

Function: dialled(*returnFirst*)

Function ID: 0

Description: Returns the call log handle of an entry from the dialled call log. The function is called repeatedly to retrieve information about all such call log entries.

The *returnFirst* parameter indicates which call log handle should be returned. The call log handles are returned in a list-traversal manner in the reverse order in which the call log entries were created, i.e., most recent call log entry first. When the *returnFirst* parameter is true, the return value is the handle for the most recent call log entry or invalid if there are no call log entries. When the *returnFirst* parameter is false, the return value is the handle of the next most recent call log entry following the previously-returned one, or is invalid if there are no call log entries older than the previously-returned one.

Note: This function is intended for use within a loop with the *returnFirst* parameter initialized to true for the first invocation and set to false for all subsequent invocations. The loop should exit once invalid is returned, indicating the end of the list.

This function returns a call log `handle` if successful or returns `invalid` if the end of the list was reached. (See section 5.4.1 for a description of the call log handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *returnFirst* = `boolean` (true=return most recent call log handle, false=return next-newest call log handle)

Return value: `handle` or `invalid`

Associated Events: -

Exceptions: -

Example: var h1 = WTACallLog.dialled(true); // read call log entry #1

var h2 = WTACallLog.dialled(false); // read call log entry #2

12.2.2. WTACallLog.missed

Function: missed(*returnFirst*)

Function ID: 1

Description: Returns the call log handle of an entry from the missed call log. The function is called repeatedly to retrieve information about all such call log entries.

The *returnFirst* parameter indicates which call log handle should be returned. The call log handles are returned in a list-traversal manner in the reverse order in which the call log entries were created, i.e., most recent call log entry first. When the *returnFirst* parameter is true, the return value is the handle for the most recent call log entry or `invalid` if there are no call log entries. When the *returnFirst* parameter is false, the return value is the handle of the next most recent call log entry following the previously-returned one, or is invalid if there are no call log entries older than the previously-returned one.

Note: This function is intended for use within a loop with the *returnFirst* parameter initialized to true for the first invocation and set to false for all subsequent invocations. The loop should exit once invalid is returned, indicating the end of the list.

This function returns a call log `handle` if successful or returns `invalid` if the end of the list was reached. (See section 5.4.1 for a description of the call log handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *returnFirst* = `boolean` (true=return most recent call log handle, false=return next-newest call log handle)

Return value: `handle` or `invalid`

Associated Events: -

Exceptions: -

Example: var h1 = WTACallLog.missed(true); // read call log entry #1

var h2 = WTACallLog.missed(false); // read call log entry #2

12.2.3. WTACallLog.received

Function: received(*returnFirst*)

Function ID: 2

Description: Returns the call log handle of an entry from the received call log. The function is called repeatedly to retrieve information about all such call log entries.

The *returnFirst* parameter indicates which call log handle should be returned. The call log handles are returned in a list-traversal manner in the reverse order in which the call log entries were created, i.e., most recent call log entry first. When the *returnFirst* parameter is true, the return value is the handle for the most recent call log entry or invalid if there are no call log entries. When the *returnFirst* parameter is false, the return value is the handle of the next most recent call log entry following the previously-returned one, or is invalid if there are no call log entries older than the previously-returned one.

Note: This function is intended for use within a loop with the *returnFirst* parameter initialized to true for the first invocation and set to false for all subsequent invocations. The loop should exit once invalid is returned, indicating the end of the list.

This function returns a call log `handle` if successful or returns `invalid` if the end of the list was reached. (See section 5.4.1 for a description of the call log handle.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *returnFirst* = `boolean` (true=return most recent call log handle, false=return next-newest call log handle)

Return value: `handle` or `invalid`

Associated Events: -

Exceptions: -

Example: var h1 = WTACallLog.received(true); // read call log entry #1

var h2 = WTACallLog.received(false); // read call log entry #2

12.2.4. WTACallLog.getFieldValue

Function: getFieldValue(*logHandle*,*field*)

Function ID: 3

Description: Retrieves a field value from a specific call log entry.

The *logHandle* parameter identifies the call log entry of interest. (See section 5.4.1 for a description of the call log handle.)

The *field* parameter specifies the name of the field to retrieve from the call log. (See section 5.4.2 for a description of the call log entry info.)

This function returns a value that depends on the *field* parameter. Any string may be given for the *field* parameter, however, if the *field* parameter matches one of the field names specified in section 5.4.2, the corresponding information must be returned as specified in that section. If the *field* parameter is unrecognized, unsupported, or otherwise unavailable, an empty `string` is returned.

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *logHandle* = `handle`

field = `string` (name of field to retrieve)

Return value: any except `invalid` (field value), or `invalid` (failure indication)

Associated Events: -

Exceptions: If the *logHandle* parameter does not refer to a call log entry for which information is available, this function returns `invalid`.

Example: var number = WTACallLog.getFieldValue(handle, "number");

13. Network-Common WTAI - Miscellaneous

13.1. WTA Events

These are miscellaneous events. All WTA event parameters are conveyed as strings. WTA implementations MUST support all the Network Common WTAI – Miscellaneous events WMLScript specified in this chapter.

13.1.1. wtaev-ms/ns

Event Name: NetworkStatus

Event ID: wtaev-ms/ns

Parameters: *inService*, *networkName*, *explanation*

Description: Indicates the value of one or more of the defined network status parameters has changed. For example, the device has changed networks or has been handed over to another cell.

The *inService* parameter indicates if the device is in service and can place or receive calls. It must be one of the following values:

"0" = the device is not in service

"1" = the device is in service

The *networkName* parameter contains the identifier for the network that is servicing the device if it is in service or is undefined if the device is not in service.

The *explanation* parameter contains the reason why the device is not in service or is undefined if the device is in service. It must be one of the following values:

"0" = no explanation given

"1" = no networks were found

"2" =only forbidden networks were found

13.2. WMLScript functions

WTA implementations MUST support all the Network Common WTAI – Miscellaneous WMLScript functions specified in this chapter.

Name: WTAMisc

Library ID: 516

Description: This library contains miscellaneous functions that are common for all networks implementing WTA

13.2.1. WTAMisc.setIndicator

Function: setIndicator(*type*,*newState*)

Function ID: 0

Description: Modifies the state of a logical indicator.

The *type* parameter identifies the indicator of interest. (See section 0 for a description of the logical indicators.)

The *newState* parameter specifies the desired state of the indicator. (See section 5.5.2 for a description of the logical indicator states.)

This function returns an empty `string` if successful, or returns `invalid` if the function fails. (See section 5.5.2 for a description of the logical indicator states.)

Permission Types: BLANKET, CONTEXT, SINGLE (see [WTA]).

Parameters: *`type`* = `integer` (indicator type:

0 = Incoming Speech Call

1 = Incoming Data Call

2 = Incoming Fax Call

3 = Call Waiting

4 = Text Message

5 = Voice Mail Message

6 = Fax Message

7 = Email Message)

`newState` = `integer` (desired state for the logical indicator)

Return value: empty `string` or `invalid`

Associated Events: -

Exceptions: If the *`type`* parameter does not reference a supported logical indicator, this function returns `invalid`.

If the *`newState`* parameter is invalid for the indicator specified by the *`type`* parameter, this function returns `invalid`.

Example: var flag = WTAMisc.setIndicator(0,1);

13.2.2. WTAMisc.endContext

Function: endContext()

Function ID: 1

Description: Terminates the current WTA user agent context.

This function returns an empty `string`.

Permission Types: not applicable – this function is always allowed and never requires user permission.

Parameters: -

Return value: empty `string`

Associated Events: -

Exceptions: -

Example: WTAMisc.endContext();

13.2.3. WTAMisc.getProtection

Function: getProtection()

Function ID: 2

Description: Retrieves the protection mode of the current WTA context.

This function returns a `boolean` that indicates the current protection mode.

Permission Types: not applicable – this function is always allowed and never requires user permission.

Parameters: -

Return value: `boolean` (true=context is protected, false=context is unprotected)

Associated Events: -

Exceptions: -

Example: var p = WTAMisc.getProtection();

13.2.4. WTAMisc.setProtection

Function: setProtection(*`mode`*)

Function ID: 3

Description: Sets the protection mode of the current WTA context.

The *`mode`* parameter specifies the desired protection mode.

This function returns an empty `string`.

Permission Types: not applicable – this function is always allowed and never requires user permission.

Parameters: *`mode`* = `boolean` (true=enable protection, false=disable protection)

Return value: empty `string`

Associated Events: -

Exceptions: -

Example: WTAMisc.setProtection(true); // enable context protection

13.3. URI functions

WTA implementations MUST support all the Network Common WTAI – Miscellaneous URI functions specified in this chapter.

13.3.1. wtai://ms/ec

Function: wtai://ms/ec

Description: Terminates the current WTA user agent context.

Permission Types: not applicable – this function is always allowed and never requires user permission.

Parameters: -

Result value: -

Associated Events: -

Exceptions: -

Example: <go href="wtai://ms/ec"/>

Appendix A. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [CREQ].

A 1 Client features

A 1.1 Voice Call Model

Item	Function	Reference	Status	Requirement
WTAI-VCM-C-001	Voice Call Model	5.1	M	WTAI-CV-C-001

A 1.2 Network message Model

Item	Function	Reference	Status	Requirement
WTAI-NMM-C-001	Network Message Model	5.2	M	WTAI-CT-C-001

A 1.3 Phonebook Model

Item	Function	Reference	Status	Requirement
WTAI-PBM-C-001	Phonebook Model	5.3	M	

A1.4 Call log Model

Item	Function	Reference	Status	Requirement
WTAI-CLM-C-001	Call Log Model	5.4	M	

A 1.5 Logical Indicator Model

Item	Function	Reference	Status	Requirement
WTAI-LIM-C-001	Logical Indicator Model	5.5	M	

A 1.6 WMLScript Library Compliance

Item	Function	Reference	Status	Requirement
WTAI-CR-C-001	WTAI library functions follow the WMLScript standard library conventions and rules.	7	M	WMLSSL:MCF

A 1.7 Public WTAI

A 1.7.1 Mandatory features in Public WTAI

Item	Function	Reference	Status	Requirement
WTAI-P-C-001	All mandatory WMLScript and all mandatory URI functions in Public WTAI library.	8.2.1, 8.2.2, 8.2.3, 8.3.1, 8.3.2 and 8.3.3	M	WTAI-PS-C-001 AND WTAI-PS-C-002 AND WTAI-PS-C-003 AND WTAI-PU-C-001 AND WTAI-PU-C-002 AND WTAI-PU-C-003

A 1.7.2 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-PS-C-001	WTAPublic.makeCall	8.2.1 and 7	M	

Item	Function	Reference	Status	Requirement
WTAI-PS-C-002	WTAPublic.sendDTMF	8.2.2 and 7	M	
WTAI-PS-C-003	WTAPublic.addPBEntry	8.2.3 and 7	M	

A 1.7.3 URI Functions

Item	Function	Reference	Status	Requirement
WTAI-PU-C-001	wtai://wp/mc	8.3.1, 6.3 and 7	M	
WTAI-PU-C-002	wtai://wp/sd	8.3.2, 6.3 and 7	M	
WTAI-PU-C-003	wtai://wp/ap	8.3.3, 6.3 and 7	M	

A 1.8 Network Common WTAI - Voice Calls

A 1.8.1 Mandatory features in Network Common WTAI – Voice Calls

Item	Function	Reference	Status	Requirement
WTAI-CV-C-001	All mandatory WTA events and all mandatory WMLScript functions in the Network Common WTAI Voice Calls library.	9.1.1, 9.1.2, 9.1.3, 9.1.4, 9.1.5, 9.1.6, 9.2.1, 9.2.2, 9.2.3, 9.2.4, 9.2.5 and 9.2.6	M	WTAI-CVE-C-001 AND WTAI-CVE-C-002 AND WTAI-CVE-C-003 AND WTAI-CVE-C-004 AND WTAI-CVE-C-005 AND WTAI-CVE-C-006 AND WTAI-CVS-C-001 AND WTAI-CVS-C-002 AND WTAI-CVS-C-003 AND WTAI-CVS-C-004 AND WTAI-CVS-C-005 AND WTAI-CVS-C-006

A 1.8.2 WTA Events

Item	Function	Reference	Status	Requirement
WTAI-CVE-C-001	Incoming Call (wtaev-cc/ic)	9.1.1	M	
WTAI-CVE-C-002	Call Cleared (wtaev-cc/cl)	9.1.2	M	
WTAI-CVE-C-003	Call Connected (wtaev-cc/co)	9.1.3	M	
WTAI-CVE-C-004	Outgoing Call(wtaev-cc/oc)	9.1.4	M	
WTAI-CVE-C-005	Call Alerting(wtaev-cc/cc)	9.1.5	M	
WTAI-CVE-C-006	DTMF sent (wtaev-cc/dtmf)	9.1.6	M	

A 1.8.3 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-CVS-C-001	WTAVoiceCall.setup	9.2.1 and 7	M	
WTAI-CVS-C-002	WTAVoiceCall.accept	9.2.2 and 7	M	
WTAI-CVS-C-003	WTAVoiceCall.release	9.2.3 and 7	M	
WTAI-CVS-C-004	WTAVoiceCall.sendDTMF	9.2.4 and 7	M	
WTAI-CVS-C-005	WTAVoiceCall.callStatus	9.2.5 and 7	M	
WTAI-CVS-C-006	WTAVoiceCall.list	9.2.6 and 7	M	

A 1.9 Network Common WTAI - Network Messages

A.1.9.1 Mandatory features in Network Common WTAI – Network Messages

Item	Function	Reference	Status	Requirement
WTAI-CT-C-001	All mandatory WTA events and all mandatory WML Script functions in the Network Common WTAI Network Messages library.	10.1.1, 10.1.2, 10.2.1, 10.2.2, 10.2.3, 10.2.4 and 10.2.5	M	WTAI-CTE-C-001 AND WTAI-CTE-C-002 AND WTAI-CTS-C-001 AND WTAI-CTS-C-002 AND WTAI-CTS-C-003 AND WTAI-CTS-C-004 AND WTAI-CTS-C-005

A 1.9.2 WTA Events

Item	Function	Reference	Status	Requirement
WTAI-CTE-C-001	Message Send Status (wtaev-nt/st)	10.1.1	M	
WTAI-CTE-C-002	Incoming Message (wtaev-nt/it)	10.1.2	M	

A 1.9.3 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-CTS-C-001	WTANetText.send	10.2.1 and 7	M	
WTAI-CTS-C-002	WTANetText.list	10.2.2 and 7	M	
WTAI-CTS-C-003	WTANetText.remove	10.2.3 and 7	M	
WTAI-CTS-C-004	WTANetText.getFieldValue	10.2.4 and 7	M	
WTAI-CTS-C-005	WTANetText.markAsRead	10.2.5 and 7	M	

A 1.10 Network Common WTAI – Phonebook

A 1.10.1 Mandatory features in Network Common WTAI – Phonebook

Item	Function	Reference	Status	Requirement
WTAI-CP-C-001	All mandatory WMLScript functions in	11.2.1, 11.2.2, 11.2.3, 11.2.4	M	WTAI-CPS-C-001 AND WTAI-CPS-C-002 AND WTAI-CPS-C-

Item	Function	Reference	Status	Requirement
	the Network Common WTAI Phonebook library.	and 11.2.5		003 AND WTAI-CPS-C-004 AND WTAI-CPS-C-005

A 1.10.2 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-CPS-C-001	WTAPhoneBook.write	11.2.1 and 7	M	
WTAI-CPS-C-002	WTAPhoneBook.search	11.2.2 and 7	M	
WTAI-CPS-C-003	WTAPhoneBook.remove	11.2.3 and 7	M	
WTAI-CPS-C-004	WTAPhoneBook.getFieldValue	11.2.4 and 7	M	
WTAI-CPS-C-005	WTAPhoneBook.change	11.2.5 and 7	M	

A 1.11 Network Common WTAI - Call Logs

A 1.11.1 Mandatory features in Network Common WTAI – Call Logs

Item	Function	Reference	Status	Requirement
WTAI-CL-C-001	All mandatory WMLScript functions in the Network Common WTAI Call Logs library.	12.2.1, 12.2.2, 12.2.3 and 12.2.4	M	WTAI-CLS-C-001 AND WTAI-CLS-C-002 AND WTAI-CLS-C-003 AND WTAI-CLS-C-004

A 1.11.2 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-CLS-C-001	WTACallLog.dialled	12.2.1 and 7	M	
WTAI-CLS-C-002	WTACallLog.missed	12.2.2 and 7	M	
WTAI-CLS-C-003	WTACallLog.received	12.2.3 and 7	M	
WTAI-CLS-C-004	WTACallLog.getFieldValue	12.2.4 and 7	M	

A 1.12 Network Common WTAI – Miscellaneous

A 1.12.1 Mandatory features in Network Common WTAI – Miscellaneous

Item	Function	Reference	Status	Requirement
WTAI-CM-C-001	All mandatory WTA events, all mandatory WMLScript functions and all mandatory URI functions in the Network Common WTAI Miscellaneous library.	13.1.1, 13.2.1, 13.2.2, 13.2.3, 13.2.4 and 13.3.1	M	WTAI-CME-C-001 AND WTAI-CMS-C-001 AND WTAI-CMS-C-002 AND WTAI-CMS-C-003 AND WTAI-CMS-C-004 AND WTAI-CMU-C-001

A 1.12.2 WTA Events

Item	Function	Reference	Status	Requirement
WTAI-CME-C-001	Network Status (wtaev-ms/ns)	13.1.1	M	

A 1.12.3 WMLScript Functions

Item	Function	Reference	Status	Requirement
WTAI-CMS-C-001	WTAMisc.setIndicator	13.2.1 and 7	M	
WTAI-CMS-C-002	WTAMisc.endContext	13.2.2 and 7	M	
WTAI-CMS-C-003	WTAMisc.getProtection	13.2.3 and 7	M	
WTAI-CMS-C-004	WTAMisc.setProtection	13.2.4 and 7	M	

A 1.12.4 URI Functions

Item	Function	Reference	Status	Requirement
WTAI-CMU-C-001	wtai://ms/ec	13.3.1, 6.3 and 7	M	

A 1.13 WMLScript Bytecode Interpreter Capabilities

A 1.13.1 General

Item	Function	Reference	Status	Requirement
WTAI-INTP-C-001	Supports Public WTAI library- and function identifiers.	8	M	WTAI-INT-C-001 AND WTAI-INT-C-002
WTAI-INTN-C-002	Supports Network common WTAI library- and function identifiers.	9, 10, 11, 12 and 13	M	WTAI-INT-C-003 AND WTAI-INT-C-004 AND WTAI-INT-C-005 AND WTAI-INT-C-006 AND WTAI-INT-C-007 AND WTAI-INT-C-008 AND WTAI-INT-C-009 AND WTAI-INT-C-010 AND WTAI-INT-C-011 AND WTAI-INT-C-0012

A 1.13.2 WTAI library and function identifiers

Item	Function	Reference	Status	Requirement
WTAI-INT-C-001	Supports Public WTAI library identifier	8	M	WMLS:MCF
WTAI-INT-C-002	Supports Public WTAI functions identifiers	8	M	WMLS:MCF
WTAI-INT-C-003	Supports Voice Call Control library identifier	9	M	WMLS:MCF
WTAI-INT-C-004	Supports Voice Call Control function identifiers	9	M	WMLS:MCF
WTAI-INT-C-005	Supports Network Text library identifier	10	M	WMLS:MCF

Item	Function	Reference	Status	Requirement
WTAI-INT-C-006	Supports Network Text function identifiers	10	M	WMLS:MCF
WTAI-INT-C-007	Supports Phonebook library identifier	11	M	WMLS:MCF
WTAI-INT-C-008	Supports Phonebook function identifiers	11	M	WMLS:MCF
WTAI-INT-C-009	Supports Call Logs library	12	M	WMLS:MCF
WTAI-INT-C-010	Supports Call Logs function identifiers	12	M	WMLS:MCF
WTAI-INT-C-011	Supports Miscellaneous library identifier	13	M	WMLS:MCF
WTAI-INT-C-012	Supports Miscellaneous function identifiers	13	M	WMLS:MCF

A 2 Server features

A 2.1 WMLScript Encoder Capabilities

Item	Function	Reference	Status	Requirement
WTAI-ENC-S-001	Supports Public WTAI library identifier	8	M	WMLS:MSF
WTAI-ENC-S-002	Supports Public WTAI functions identifiers	8	M	WMLS:MSF
WTAI-ENC-S-003	Supports Voice Call Control library identifier	9	M	WMLS:MSF
WTAI-ENC-S-004	Supports Voice Call Control function identifiers	9	M	WMLS:MSF
WTAI-ENC-S-005	Supports Network Text library identifier	10	M	WMLS:MSF
WTAI-ENC-S-006	Supports Network Text function identifiers	10	M	WMLS:MSF
WTAI-ENC-S-007	Supports Phonebook library identifier	11	M	WMLS:MSF
WTAI-ENC-S-008	Supports Phonebook function identifiers	11	M	WMLS:MSF
WTAI-ENC-S-009	Supports Call Logs library identifier	12	M	WMLS:MSF
WTAI-ENC-S-010	Supports Call Logs function identifiers	12	M	WMLS:MSF
WTAI-ENC-S-011	Supports Miscellaneous library identifier	13	M	WMLS:MSF
WTAI-ENC-S-012	Supports Miscellaneous function identifiers	13	M	WMLS:MSF

Appendix B. WTAI URI and WMLScript Function Libraries (Informative)

In the tables below, the URI and WMLScript Function Libraries Calls are summarised. The arguments have been left out in order to increase readability. The values in the column named "Lib/Func ID" denote the *Library* and *Function ID*s.

A.1 Public WTAI

Lib/Func ID	*URI*	*WMLScript call*	*Description*
512.0	wtai://wp/mc	WTAPublic.makeCall	Make a call
512.1	wtai://wp/sd	WTAPublic.sendDTMF	Send DTMF Tones
512.2	wtai://wp/ap	WTAPublic.addPBEntry	Add a new phonebook entry

A.2 Network Common WTAI

A.2.1 Voice Call Control

Lib/Func ID	*URI*	*WMLScript call*	*Description*
513.0	-	WTAVoiceCall.setup	Setup a new call
513.1	-	WTAVoiceCall.accept	Accept an incoming call
513.2	-	WTAVoiceCall.release	Release a call
513.3	-	WTAVoiceCall.sendDTMF	Send DTMF sequence
513.4	-	WTAVoiceCall.callStatus	Retrieve parameters for a specific call
513.5	-	WTAVoiceCall.list	Retrieve call handles for calls that can be controlled within the WTA context that invoked the function

A.2.2 Network Messages

Lib/Func ID	*URI*	*WMLScript call*	*Description*
514.0	-	WTANetText.send	Send network text
514.1	-	WTANetText.list	List network messages
514.2	-	WTANetText.remove	Remove network message
514.3	-	WTANetText.getFieldValue	Get Field Value
514.4	-	WTANetText.markAsRead	Mark a message as read

A 2.3 Phonebook

Lib/Func ID	*URI*	*WMLScript call*	*Description*
515.0	-	WTAPhoneBook.write	Write phonebook entry
515.1	-	WTAPhoneBook.search	Search phonebook entries
515.2	-	WTAPhoneBook.remove	Remove phonebook entry
515.3	-	WTAPhoneBook.getFieldValue	Get Field Value
515.4	-	WTAPhoneBook.change	Change an existing phonebook entry

A 2.4 Call Logs

Lib/Func ID	*URI*	*WMLScript call*	*Description*
519.0	-	WTACallLog.dialled	Read "last dialled numbers" log
519.1	-	WTACallLog.missed	Read "missed calls" log
519.2	-	WTACallLog.received	Read "received calls" log
519.3	-	WTACallLog.getFieldValue	Get Field Value

A 2.5 Miscellaneous

Lib/Func ID	*URI*	*WMLScript call*	*Description*
516.0	-	WTAMisc.setIndicator	Change logical indicator
516.1	wtai://ms/ec	WTAMisc.endContext	Terminates user-agent context
516.2	-	WTAMisc.getProtection	Reads context protection mode
516.3	-	WTAMisc.setProtection	Sets context protection mode

Appendix C. Examples using WTAI (Informative)

WTAI functions can be called in either of the following two ways: using a URI or using WMLScript. These examples show how simple problems could be solved using WTAI functions invoked from WML or WMLScript.

C 1 Simple WTAI example using URI invocation

This simple example shows the invocation of a WTAI function using the URI mechanism. This particular example places an outgoing call from a WML card.

WML:

```
<!--
This card lets the user dial a phone number.
Since this deck/card is "untrusted", the user will be prompted before the voice
call is actually initiated.
-->
<wml:card>
    <p>
        <!-- give some info about us -->
        We have the best products!

        <!-- use a hyperlink to allow user to call us -->
        <a href="wtai://wp/mc;5551212">Call Now</a>
    </p>
</wml:card>
```

C 2 Simple WTAI example using WMLScript invocation

This simple example shows the invocation of a WTAI function using the WMLScript mechanism. Notice the capability for error checking and reporting in the WMLScript example as compared to the previous (i.e., URI) example.

WMLScript (source code for "myScript.wmls"):

```
/*
This function dials the given number and does some error handling.
If the WTAI function succeeds, the browser is navigated to the #Dialing card.
If the WTAI function fails, the browser is navigated to the #Error card.
*/
extern function CallFood(number)
    {
    // try to dial the given number
    var handle = WTAVoiceCall.setup(number,true);  // true = keep mode

    // check for success/fail
    // NOTE: failure is indicated by invalid
    if (isvalid handle)
        {
        // WTAI function succeeded, show voice call is being made
        WMLBrowser.setVar("msg", "Phone number is '" + number + "'");
        WMLBrowser.go("#Dialing");
        }
    else
        {
        // WTAI function failed, warn user of error
        WMLBrowser.setVar("msg", "Could not place call to '" + number + "'");
        WMLBrowser.go("#Error");
        }
    }
```

WML:

```
<!--
This card lets the user order the desired type of food.
-->
<wml:card>
     <wml:do role="positive" label="Order Food">
                  <wml:widget type=”default”>
          <go href="myScript.wmls#CallFood('$(foodNumber)')"/>
     </wml:do>
     <p>
          Choose Food:
          <select wml:name="foodNumber">
              <option value="5556789">Pizza</option>
              <option value="5551234">Chinese</option>
              <option value="5553344">Sandwich</option>
              <option value="5551122">Burger</option>
          </select>
     </p>
</wml:card>


<!--
This card informs the user that food is being ordered.
-->
<wml:card id="Dialing">
     <p>
          Calling for food...<br/>
          <wml:getvar name=”msg”>
              </p>
</wml:card>


<!--
This card informs the user that an error was detected.
-->
<wml:card id="Error">
     <p>
          ERROR<br/>
          $(msg)
     </p>
</wml:card>
```

Appendix D. Change History (Informative)

Type of Change	Date	Section	Description
Class 0	08-Sep-2001		The initial version of this document.