WMLScript Standard Libraries Specification

25-SEP-2000

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.
You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. You may not use this document in any other manner without the prior written permission of the WAP Forum™. The WAP Forum authorises you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services offered by you.

The WAP Forum™ assumes no responsibility for errors or omissions in this document. In no event shall the WAP Forum be liable for any special, indirect or consequential damages or any damages whatsoever arising out of or in connection with the use of this information.

WAP Forum™ members have agreed to use reasonable endeavors to disclose in a timely manner to the WAP Forum the existence of all intellectual property rights (IPR’s) essential to the present document. The members do not have an obligation to conduct IPR searches. This information is publicly available to members and non-members of the WAP Forum and may be found on the "WAP IPR Declarations" list at http://www.wapforum.org/what/ipr.htm. Essential IPR is available for license on the basis set out in the schedule to the WAP Forum Application Form.

No representations or warranties (whether express or implied) are made by the WAP Forum™ or any WAP Forum member or its affiliates regarding any of the IPR’s represented on this list, including but not limited to the accuracy, completeness, validity or relevance of the information or whether or not such rights are essential or non-essential.
Contents

1. Scope ........................................................................................................................................... 5
2. Document Status .............................................................................................................................. 6
   2.1 Copyright Notice .......................................................................................................................... 6
   2.2 Errata .......................................................................................................................................... 6
   2.3 Comments ................................................................................................................................. 6
   2.4 Document Changes ..................................................................................................................... 6
       2.4.1 WAP-194.100 15-May-2000 ............................................................................................... 6
       2.4.2 WAP-194.101 25-Sept-2000 ............................................................................................. 7
   2.5 Document History ...................................................................................................................... 7
3. References ........................................................................................................................................ 8
   3.1 Normative references .................................................................................................................. 8
   3.2 Informative References .............................................................................................................. 8
4. Definitions and abbreviations .............................................................................................................. 9
   4.1 Definitions .................................................................................................................................. 9
   4.2 Abbreviations ............................................................................................................................ 10
5. Notational Conventions .................................................................................................................... 11
6. WMLScript Compliance .................................................................................................................... 13
   6.1 Supported Data Type .................................................................................................................. 13
   6.2 Data Type Conversions ............................................................................................................... 13
   6.3 Error Handling ............................................................................................................................ 13
   6.4 Support for Integer-Only Devices .............................................................................................. 13
7. Lang ................................................................................................................................................ 15
   7.1 abs ............................................................................................................................................. 15
   7.2 min .......................................................................................................................................... 15
   7.3 max .......................................................................................................................................... 16
   7.4 parseInt .................................................................................................................................... 16
   7.5 parseFloat ................................................................................................................................ 17
   7.6 parseInt .................................................................................................................................... 17
   7.7 isFloat ..................................................................................................................................... 18
   7.8 maxInt .................................................................................................................................... 18
   7.9 minInt ..................................................................................................................................... 18
   7.10 float ....................................................................................................................................... 19
   7.11 exit ....................................................................................................................................... 19
   7.12 abort ..................................................................................................................................... 19
   7.13 random .................................................................................................................................. 20
   7.14 seed ...................................................................................................................................... 20
   7.15 characterSet ............................................................................................................................ 21
8. Float ............................................................................................................................................... 22
   8.1 int ............................................................................................................................................ 22
   8.2 floor ........................................................................................................................................ 22
   8.3 ceil .......................................................................................................................................... 23
   8.4 pow .......................................................................................................................................... 23
   8.5 round ....................................................................................................................................... 23
   8.6 sqrt .......................................................................................................................................... 24
   8.7 maxFloat ................................................................................................................................. 24
   8.8 minFloat ................................................................................................................................ 24
9. String ............................................................................................................................................. 26
   9.1 length ...................................................................................................................................... 26
   9.2 isEmpty .................................................................................................................................. 26

All rights reserved
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3</td>
<td><code>charAt</code></td>
<td>27</td>
</tr>
<tr>
<td>9.4</td>
<td><code>subString</code></td>
<td>27</td>
</tr>
<tr>
<td>9.5</td>
<td><code>find</code></td>
<td>28</td>
</tr>
<tr>
<td>9.6</td>
<td><code>replace</code></td>
<td>28</td>
</tr>
<tr>
<td>9.7</td>
<td><code>elements</code></td>
<td>29</td>
</tr>
<tr>
<td>9.8</td>
<td><code>elementAt</code></td>
<td>29</td>
</tr>
<tr>
<td>9.9</td>
<td><code>removeAt</code></td>
<td>30</td>
</tr>
<tr>
<td>9.10</td>
<td><code>replaceAll</code></td>
<td>31</td>
</tr>
<tr>
<td>9.11</td>
<td><code>insertAt</code></td>
<td>31</td>
</tr>
<tr>
<td>9.12</td>
<td><code>squeeze</code></td>
<td>32</td>
</tr>
<tr>
<td>9.13</td>
<td><code>trim</code></td>
<td>32</td>
</tr>
<tr>
<td>9.14</td>
<td><code>compare</code></td>
<td>33</td>
</tr>
<tr>
<td>9.15</td>
<td><code>toString</code></td>
<td>33</td>
</tr>
<tr>
<td>9.16</td>
<td><code>format</code></td>
<td>34</td>
</tr>
<tr>
<td>10.1</td>
<td><code>isValid</code></td>
<td>36</td>
</tr>
<tr>
<td>10.2</td>
<td><code>getScheme</code></td>
<td>36</td>
</tr>
<tr>
<td>10.3</td>
<td><code>getHost</code></td>
<td>37</td>
</tr>
<tr>
<td>10.4</td>
<td><code>getPort</code></td>
<td>37</td>
</tr>
<tr>
<td>10.5</td>
<td><code>getPath</code></td>
<td>38</td>
</tr>
<tr>
<td>10.6</td>
<td><code>getParameters</code></td>
<td>38</td>
</tr>
<tr>
<td>10.7</td>
<td><code>getQuery</code></td>
<td>39</td>
</tr>
<tr>
<td>10.8</td>
<td><code>getFragment</code></td>
<td>39</td>
</tr>
<tr>
<td>10.9</td>
<td><code>getBase</code></td>
<td>39</td>
</tr>
<tr>
<td>10.10</td>
<td><code>getReferer</code></td>
<td>40</td>
</tr>
<tr>
<td>10.11</td>
<td><code>resolve</code></td>
<td>40</td>
</tr>
<tr>
<td>10.12</td>
<td><code>escapeString</code></td>
<td>41</td>
</tr>
<tr>
<td>10.13</td>
<td><code>unescapeString</code></td>
<td>42</td>
</tr>
<tr>
<td>10.14</td>
<td><code>loadString</code></td>
<td>43</td>
</tr>
<tr>
<td>11.1</td>
<td><code>getVar</code></td>
<td>44</td>
</tr>
<tr>
<td>11.2</td>
<td><code>setVar</code></td>
<td>44</td>
</tr>
<tr>
<td>11.3</td>
<td><code>go</code></td>
<td>45</td>
</tr>
<tr>
<td>11.4</td>
<td><code>prev</code></td>
<td>46</td>
</tr>
<tr>
<td>11.5</td>
<td><code>newContext</code></td>
<td>47</td>
</tr>
<tr>
<td>11.6</td>
<td><code>getCurrentCard</code></td>
<td>47</td>
</tr>
<tr>
<td>11.7</td>
<td><code>refresh</code></td>
<td>47</td>
</tr>
<tr>
<td>12.1</td>
<td><code>prompt</code></td>
<td>49</td>
</tr>
<tr>
<td>12.2</td>
<td><code>confirm</code></td>
<td>49</td>
</tr>
<tr>
<td>12.3</td>
<td><code>alert</code></td>
<td>50</td>
</tr>
<tr>
<td>Appendix A. Library Summary</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Appendix B. Static Conformance Requirements</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>12.4</td>
<td><code>WMLScript Encoder Capabilities</code></td>
<td>53</td>
</tr>
<tr>
<td>12.5</td>
<td><code>WMLScript Bytecode Interpreter Capabilities</code></td>
<td>55</td>
</tr>
</tbody>
</table>
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 standards 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* [WAP].

This document specifies the library interfaces for the standard set of libraries supported by WMLScript [WMLScript] to provide access to the core functionality of a WAP client. WMLScript is a language that can be used to provide programmed functionality to WAP based applications. It is part of the WAP platform and it can be used to add script support also to the client.

One of the main differences between ECMAScript [ECMA262] and WMLScript is the fact that WMLScript is compiled into bytecode before it is being sent to the client. This way the narrowband communication channels available today can be optimally utilized and the memory requirements for the client kept to the minimum. For the same reasons, many of the advanced features of the JavaScript language have been removed to make the language both optimal, easier to compile into bytecode and easier to learn.

Library support has been added to the WMLScript to replace some of the functionality that has been removed from ECMAScript in accordance to make the WMLScript more efficient. This feature provides access to built-in functionality and a means for future expansion without unnecessary overhead.

The following chapters describe the set of libraries defined to provide access to core functionality of a WAP client. This means that all libraries, except *Float*, are present in the client's scripting environment. Float library is optional and only supported with clients that can support floating-point arithmetic operations.
2. DOCUMENT STATUS

This document is available online in the following formats:

- PDF format at http://www.wapforum.org/.

2.1 Copyright Notice


2.2 Errata

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

2.3 Comments

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

2.4 Document Changes

2.4.1 WAP-194.100 15-May-2000

<table>
<thead>
<tr>
<th>Change Request</th>
<th>Title</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSL-IBM-20000320-SeedSequence</td>
<td>Lang.seed Example</td>
<td>Section 7.14</td>
</tr>
<tr>
<td>WMLSL-IBM-20000315-FormatConversion</td>
<td>String Format Conversions</td>
<td>Section9.16</td>
</tr>
<tr>
<td>WMLSL-IBM-20000315-CancelNav</td>
<td>WMLBrowser – Canceling Navigation</td>
<td>Sections11.3and11.4</td>
</tr>
<tr>
<td>WMLSL-IBM-20000308-PowExample</td>
<td>Correct Floating Point Example</td>
<td>Section 8.4</td>
</tr>
<tr>
<td>WMLSL-IBM-20000308-AbortText</td>
<td>Description of Lang.abort()</td>
<td>Section 7.12</td>
</tr>
<tr>
<td>WMLSL-IBM-20000218-v2-seed</td>
<td>Lang.seed() – Non-numeric Input</td>
<td>Section 7.14</td>
</tr>
</tbody>
</table>
2.4.2 WAP-194.101 25-Sept-2000

<table>
<thead>
<tr>
<th>Change Request</th>
<th>Title</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSL-IBM-20000803-LoadString</td>
<td>Clarification on the return from loadString(). It returns the content type &quot;text&quot;.</td>
<td>Section 10.14</td>
</tr>
<tr>
<td>WMLSL-IBM-20000810-NewContext</td>
<td>Clarification of effect of newContext() call on go() and prev().</td>
<td>Section 11.5</td>
</tr>
<tr>
<td>New Static Conformance Format</td>
<td>Added Inter-specification dependency Column</td>
<td>Section 12.4</td>
</tr>
</tbody>
</table>

2.5 Document History

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Date of Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAP-194-WMLSL-20000925</td>
<td>Approved Re-release (rolled up)</td>
</tr>
<tr>
<td>WAP-194.101-WMLSL-20000925</td>
<td>SCD 09-Sept-2000</td>
</tr>
<tr>
<td>SPEC-WMLScriptLibs-v1_2</td>
<td>24-March-2000</td>
</tr>
<tr>
<td>SPEC-WMLScriptLibs-19990815</td>
<td>15-August-1999</td>
</tr>
</tbody>
</table>
3. REFERENCES

3.1 Normative references


3.2 Informative References

4. DEFINITIONS AND ABBREVIATIONS

4.1 Definitions

The following are terms and conventions used throughout this specification.

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]. In the absence of any such terms, the specification should be interpreted as "MUST".

**Bytecode** - content encoding where the content is typically a set of low-level opcodes (i.e., instructions) and operands for a targeted hardware (or virtual) machine.

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

**Content** - subject matter (data) stored or generated at an origin server. Content is typically displayed or interpreted by a user agent in response to a user request.

**Content Encoding** - when used as a verb, content encoding indicates the act of converting a data object from one format to another. Typically the resulting format requires less physical space than the original, is easier to process or store and/or is encrypted. When used as a noun, content encoding specifies a particular format or encoding standard or process.

**Content Format** – actual representation of content.

**Device** - a network entity that is capable of sending and receiving packets of information and has a unique device address. A device can act as either a client or 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.

**JavaScript** - a de facto standard language that can be used to add dynamic behaviour to HTML documents. JavaScript is one of the originating technologies of ECMAScript.

**Origin Server** - the server on which a given resource resides or is to be created. Often referred to as a web server or an HTTP server.

**Resource** - a network data object or service that can be identified by a URL. Resources may be available in multiple representations (e.g. multiple languages, data formats, size and resolutions) or vary in other ways.

**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.

Web Server - a network host that acts as an HTTP server.

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

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

4.2 Abbreviations

For the purposes of this specification, the following abbreviations apply:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ECMA</td>
<td>European Computer Manufacturer Association</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol [RFC2068]</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bits</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bits</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator [RFC2396]</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>WSP</td>
<td>Wireless Session Protocol</td>
</tr>
<tr>
<td>WTP</td>
<td>Wireless Transport Protocol</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Application Protocol</td>
</tr>
<tr>
<td>WAE</td>
<td>Wireless Application Environment</td>
</tr>
<tr>
<td>WTA</td>
<td>Wireless Telephony Applications</td>
</tr>
<tr>
<td>WTAI</td>
<td>Wireless Telephony Applications Interface</td>
</tr>
<tr>
<td>WBMP</td>
<td>Wireless BitMaP</td>
</tr>
</tbody>
</table>
5. NOTATIONAL CONVENTIONS

The libraries in this document are represented by providing the following information:

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

Examples: Lang, String

LIBRARY ID: The numeric identifier reserved for the library to be used by the WMLScript Compiler. The range of values reserved for this identifier is divided into the following two categories:

0 .. 32767 Reserved for standard libraries.
32768 .. 65535 Reserved for future use.

DESCRIPTION: A short description of the library and used conventions.

Each function in the library is represented by providing the following information:

FUNCTION: Specifies the function name and the number of function parameters. The syntax of the function name follows the syntax specified in the [WMLScript] specification. Function names are case sensitive.

Example: abs(value)
Usage: var a = 3*Lang.abs(length);

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: Describes the function behaviour and its parameters.
PARAMETERS: Specifies the function parameter types.

Example: value = Number

RETURN VALUE: Specifies the type(s) of the return value.

Example: String or invalid.

EXCEPTIONS: Describes the possible special exceptions and error codes and the corresponding return values. Standard errors, common to all functions, are not described here (see 6.3 for more information about error handling).

Example: If the value1 <= 0 and value2 < 0 and not an integer then invalid is returned.

EXAMPLE: Gives a few examples of how the function could be used.

var a = -3;
var b = Lang.abs(a); // b = 3
6. WMLSCRIPT COMPLIANCE

WMLScript standard library functions provide a mechanism to extend the WMLScript language. Thus, the specified library functions must follow the WMLScript conventions and rules.

6.1 Supported Data Type

The following WMLScript types [WMLScript] are used in the function definitions to denote the type of both the function parameters and return values:

- Boolean, Integer, Float, String and Invalid

In addition to these, number can be used to denote a parameter type when both integer and floating-point parameter value types are accepted. Any can be used when the type can be any of the supported types.

6.2 Data Type Conversions

Since WMLScript is a weakly typed language, the conversions between the data types are done automatically if necessary (see [WMLScript] for more details about data type conversion rules). The library functions follow WMLScript operator data type conversion rules except where explicitly stated otherwise.

6.3 Error Handling

Error cases are handled in the same way as in the WMLScript language (see [WMLScript] for more details):

- An invalid function argument results in an invalid return value with no other side effects unless explicitly stated otherwise.
- A function argument that cannot be converted to the required parameter type results in an invalid return value with no side effects. See 6.2 for more information about data type conversions.
- Function dependent error cases are handled by returning a suitable error code specified in each function definition. These errors are documented as part of the function specification (exceptions).

6.4 Support for Integer-Only Devices

The WMLScript language has been designed to run also on devices that do not support floating-point operations. The WMLScript standard libraries have operations that require floating-point support. Thus, the following rules apply when the libraries are implemented for an integer-only device:

- Library functions accept arguments of the following type only: boolean, integer, string and invalid.
- All conversion rules related to floating-point data are ignored.
- Lang.float() function returns false.
- Lang.parseFloat() function returns invalid.
• `String.format()` function returns `invalid` when type `f` is specified in the format.
• All `Float` (see chapter 8) library functions return `invalid`. 
7. **LANG**

   NAME: Lang
   LIBRARY ID: 0
   DESCRIPTION: This library contains a set of functions that are closely related to the WMLScript language core.

7.1 **abs**

   FUNCTION: abs(value)
   FUNCTION ID: 0
   DESCRIPTION: Returns the absolute value of the given number. If the given number is of type integer then an integer value is returned. If the given number is of type floating-point then a floating-point value is returned.
   PARAMETER S: value = Number
   RETURN VALUE: Number or invalid.
   EXCEPTIONS: -
   EXAMPLE: var a = -3;
             var b = Lang.abs(a);  // b = 3

7.2 **min**

   FUNCTION: min(value1, value2)
   FUNCTION ID: 1
   DESCRIPTION: Returns the minimum value of the given two numbers. The value and type returned is the same as the value and type of the selected number. The selection is done in the following way:
   - WMLScript operator data type conversion rules for *integers and floating-points* (see [WMLScript]) must be used to specify the data type (integer or floating-point) for comparison.
   - Compare the numbers to select the smaller one.
   - If the values are equal then the first value is selected.
   PARAMETER S: value1 = Number
                value2 = Number
   RETURN VALUE: Number or invalid.
   EXCEPTIONS: -
EXAMPLE:

```javascript
var a = -3;
var b = Lang.abs(a);
var c = Lang.min(a,b); // c = -3
var d = Lang.min(45, 76.3); // d = 45 (integer)
var e = Lang.min(45, 45.0); // e = 45 (integer)
```

### 7.3 max

**FUNCTION:** `max(value1, value2)`

**FUNCTION ID:** 2

**DESCRIPTION:** Returns the maximum value of the given two numbers. The value and type returned is the same as the value and type of the selected number. The selection is done in the following way:

- WMLScript operator data type conversion rules for integers and floating-points (see [WMLScript]) must be used to specify the data type (integer or floating-point) for comparison.
- Compare the numbers to select the larger one.
- If the values are equal then the first value is selected.

**PARAMETERS:**

- `value1` = Number
- `value2` = Number

**RETURN VALUE:** Number or invalid.

**EXAMPLES:**

```javascript
var a = -3;
var b = Lang.abs(a);
var c = Lang.max(a,b); // c = 3
var d = Lang.max(45.5, 76); // d = 76 (integer)
var e = Lang.max(45.0, 45); // e = 45.0 (float)
```

### 7.4 parseInt

**FUNCTION:** `parseInt(value)`

**FUNCTION ID:** 3

**DESCRIPTION:** Returns an integer value defined by the string `value`. The legal integer syntax is specified by the WMLScript (see [WMLScript]) numeric string grammar for decimal integer literals with the following additional parsing rule:

- Parsing ends when the first character is encountered that is not a leading '+' or '-' or a decimal digit.

The result is the parsed string converted to an integer value.

**PARAMETERS:**

- `value` = String
RETURN VALUE: Integer or invalid.

EXCEPTIONS: In case of a parsing error an invalid value is returned.

EXAMPLE: var i = Lang.parseInt("1234"); // i = 1234
         var j = Lang.parseInt(" 100 m/s"); // j = 100

7.5 parseFloat

FUNCTION: parseFloat(value)  
FUNCTION ID: 4

DESCRIPTION: Returns a floating-point value defined by the string value. The legal floating-point syntax is specified by the WMLScript (see [WMLScript]) numeric string grammar for decimal floating-point literals with the following additional parsing rule:

- Parsing ends when the first character is encountered that cannot be parsed as being part of the floating-point representation.

The result is the parsed string converted to a floating-point value.

PARAMETERS: value = String

RETURN VALUE: Floating-point or invalid.

EXCEPTIONS: In case of a parsing error an invalid value is returned.

If the system does not support floating-point operations then an invalid value is returned.

EXAMPLE: var a = Lang.parseFloat("123.7"); // a = 123.7
         var b = Lang.parseFloat(" +7.34e2 Hz"); // b = 7.34e2
         var c = Lang.parseFloat(" 70e-2 F"); // c = 70.0e-2
         var d = Lang.parseFloat(".1 C"); // d = -0.1
         var e = Lang.parseFloat(" 100 "); // e = 100.0
         var f = Lang.parseFloat("Number: 5.5"); // f = invalid
         var g = Lang.parseFloat("7.3e meters"); // g = invalid
         var h = Lang.parseFloat("7.3e- m/s"); // h = invalid

7.6 isInt

FUNCTION: isInt(value)  
FUNCTION ID: 5

DESCRIPTION: Returns a boolean value that is true if the given value can be converted into an integer number by using parseInt(value). Otherwise false is returned.

PARAMETERS: value = Any

RETURN VALUE: Boolean or invalid.
7.7 **isFloat**

FUNCTION: `isFloat(value)`

FUNCTION ID: 6

DESCRIPTION: Returns a boolean value that is true if the given value can be converted into a floating-point number using `parseFloat(value)`. Otherwise false is returned.

PARAMETERS: `value` = Any

RETURN VALUE: Boolean or invalid.

EXCEPTIONS: If the system does not support floating-point operations then an invalid value is returned.

EXAMPLE:
```javascript
var a = Lang.isFloat(" -123");   // true
var b = Lang.isFloat(" 123.33"); // true
var c = Lang.isFloat("string");  // false
var d = Lang.isFloat("#123.33"); // false
var e = Lang.isFloat(invalid);   // invalid
```

7.8 **maxInt**

FUNCTION: `maxInt()`

FUNCTION ID: 7

DESCRIPTION: Returns the maximum integer value.

PARAMETERS: -

RETURN VALUE: Integer 2147483647.

EXCEPTIONS: -

EXAMPLE:
```
var a = Lang.maxInt();
```

7.9 **minInt**

FUNCTION: `minInt()`

FUNCTION ID: 8
DESCRIPTION: Returns the minimum integer value.

PARAMETER: -

RETURN VALUE: Integer –2147483648.

EXAMPLE: var a = Lang.minInt();

7.10 float

FUNCTION: float()

FUNCTION ID: 9

DESCRIPTION: Returns true if floating-points are supported and false if not.

PARAMETER: -

RETURN VALUE: Boolean.

EXAMPLE: var floatsSupported = Lang.float();

7.11 exit

FUNCTION: exit(value)

FUNCTION ID: 10

DESCRIPTION: Ends the interpretation of the WMLScript bytecode and returns the control back to the caller of the WMLScript interpreter with the given return value. This function can be used to perform a normal exit from a function in cases where the execution of the WMLScript bytecode should be discontinued.

PARAMETERS: value = Any

RETURN VALUE: None (this function ends the interpretation).

EXAMPLE: Lang.exit("Value: " + myVal); // Returns a string
Lang.exit(invalid); // Returns invalid

7.12 abort

FUNCTION: abort(errorDescription)

FUNCTION ID: 11
DESCRIPTION: Aborts the interpretation of the WMLScript bytecode and returns the control back to the caller of the WMLScript interpreter with the return errorDescription. This function can be used to perform an abnormal exit in cases where the execution of the WMLScript should be discontinued due to serious errors detected by the program. If the type of the errorDescription is invalid, string "invalid" is used as the errorDescription instead.

PARAMETERS: errorDescription = String

RETURN VALUE: None (this function aborts the interpretation).

EXAMPLE: Lang.abort("Error: " + errVal); // Error value is a string

7.13 random

FUNCTION: random(value)

FUNCTION ID: 12

DESCRIPTION: Returns an integer value with positive sign that is greater than or equal to 0 but less than or equal to the given value. The return value is chosen randomly or pseudo-randomly with approximately uniform distribution over that range, using an implementation-dependent algorithm or strategy.

If the value is of type floating-point, Float.int() is first used to calculate the actual integer value.

PARAMETERS: value = Number

RETURN VALUE: Integer or invalid.

EXAMPLE: var a = 10;
var b = Lang.random(5.1)*a; // b = 0..50
var c = Lang.random("string"); // c = invalid

7.14 seed

FUNCTION: seed(value)

FUNCTION ID: 13

© 2000,2001, Wireless Application Protocol Forum, Ltd. All rights reserved
DESCRIPTIO
N: Initialises the pseudo-random number sequence and returns an empty string. If the value is zero or a positive integer then the given value is used for initialisation, otherwise a random, system dependent initialisation value is used. A seed value of greater than or equal to zero results in a repeatable sequence of pseudo-random numbers. A seed value of less than zero results in a non-repeatable sequence of random numbers.

If the value is of type floating-point, Float.int() is first used to calculate the actual integer value. If the value is non-numeric, invalid is returned and the current seed is unchanged.

PARAMETER S: value = Number

RETURN VALUE: String or invalid.

EXAMPLE:
```javascript
var a = Lang.seed(123); // a = ""
var b = Lang.random(20); // b = 0..20
var c = Lang.seed("seed"); // c = invalid (random seed left unchanged)
Lang.seed(7);
var a = Lang.rand(10); // a = 4(perhaps);
Lang.seed(7);
var b = Lang.rand(10); // b = 4(perhaps, but same as a)
Lang.seed(-1);
var c = Lang.rand(10); // c = 6(perhaps)
Lang.seed(-1);
var d = Lang.rand(10); // d = 1(perhaps, but not necessarily the same as c)
```

7.15 CHARACTERSET

FUNCTION: characterSet()

FUNCTION ID: 14

DESCRIPTIO
N: Returns the character set supported by the WMLScript Interpreter. The return value is an integer that denotes a MIBEnum value assigned by the IANA for all character sets (see [WSP] for more information).

PARAMETER S: -

RETURN VALUE: Integer.

EXAMPLE:
```
var charset = Lang.characterSet(); // charset = 4 for latin1
```
8. FLOAT

NAME: Float
LIBRARY ID: 1
DESCRIPTION: This library contains a set of typical arithmetic floating-point functions that are frequently used by applications.

The implementation of these library functions is optional and implemented only by devices that can support floating-point operations (see 6.4). If floating-point operations are not supported, all functions in this library must return invalid.

8.1 int

FUNCTION: int(value)
FUNCTION ID: 0
DESCRIPTION: Returns the integer part of the given value. If the value is already an integer, the result is the value itself.
PARAMETERS: value = Number
RETURN VALUE: Integer or invalid.
EXAMPLE:
var a = 3.14;
var b = Float.int(a); // b = 3
var c = Float.int(-2.8); // c = -2

8.2 floor

FUNCTION: floor(value)
FUNCTION ID: 1
DESCRIPTION: Returns the greatest integer value that is not greater than the given value. If the value is already an integer, the result is the value itself.
PARAMETERS: value = Number
RETURN VALUE: Integer or invalid.
EXAMPLE:
var a = 3.14;
var b = Float.floor(a); // b = 3
var c = Float.floor(-2.8); // c = -2
8.3 ceil

FUNCTION: ceil(value)

FUNCTION ID: 2

DESCRIPTION: Returns the smallest integer value that is not less than the given value. If the value is already an integer, the result is the value itself.

PARAMETERS:

- value = Number

RETURN VALUE: Integer or invalid.

EXAMPLE:

```javascript
var a = 3.14;
var b = Float.ceil(a); // b = 4
var c = Float.ceil(-2.8); // c = -2
```

8.4 pow

FUNCTION: pow(value1, value2)

FUNCTION ID: 3

DESCRIPTION: Returns an implementation-dependent approximation to the result of raising value1 to the power of value2. If value1 is a negative number then value2 must be an integer.

PARAMETERS:

- value1 = Number
- value2 = Number

RETURN VALUE: Floating-point or invalid.

EXAMPLE:

```javascript
var a = 3;
var b = Float.pow(a, 2); // b = 9.0
```

8.5 round

FUNCTION: round(value)

FUNCTION ID: 4

DESCRIPTION: Returns the number value that is closest to the given value and is equal to a mathematical integer. If two integer number values are equally close to the value, the result is the larger number value. If the value is already an integer, the result is the value itself.

PARAMETERS:

- value = Number
RETURN: Integer or invalid.

EXAMPLE:

var a = Float.round(3.5);  // a = 4
var b = Float.round(-3.5); // b = -3
var c = Float.round(0.5);  // c = 1
var d = Float.round(-0.5); // b = 0

8.6 sqrt

FUNCTION: sqrt(value)
FUNCTION ID: 5

DESCRIPTION: Returns an implementation-dependent approximation to the square root of the given value.

PARAMETERS: value = Floating-point

RETURN VALUE: Floating-point or invalid.

EXAMPLE:

var a = 4;
var b = Float.sqrt(a); // b = 2.0
var c = Float.sqrt(5); // c = 2.2360679775

8.7 maxFloat

FUNCTION: maxFloat()
FUNCTION ID: 6

DESCRIPTION: Returns the maximum floating-point value supported by [IEEE754] single precision floating-point format.

PARAMETERS: -

RETURN VALUE: Floating-point 3.40282347E+38.

EXAMPLE:

var a = Float.maxFloat();

8.8 minFloat

FUNCTION: minFloat()
FUNCTION ID: 7

DESCRIPTION: Returns the smallest nonzero floating-point value supported by [IEEE754] single precision floating-point format.
PARAMETER  -
S:
RETURN Floating-point. Smaller than or equal to the normalised minimum single
VALUE: precision floating-point value: 1.17549435E-38.
EXCEPTIONS -
:
EXAMPLE:  var a = Float.minFloat();
9. STRING

NAME: String
LIBRARY ID: 2
DESCRIPTION: This library contains a set of string functions. A string is an array of characters. Each of the characters has an index. The first character in a string has an index zero (0). The length of the string is the number of characters in the array.

The user of the String library can specify a special separator by which elements in a string can be separated. These elements can be accessed by specifying the separator and the element index. The first element in a string has an index zero (0). Each occurrence of the separator in the string separates two elements (no escaping of separators is allowed).

A White space character is one of the following characters:
- TAB: Horizontal Tabulation
- VT: Vertical Tabulation
- FF: Form Feed
- SP: Space
- LF: Line Feed
- CR: Carriage Return

9.1 length

FUNCTION: length(string)
FUNCTION ID: 0
DESCRIPTION: Returns the length (number of characters) of the given string.
PARAMETERS: string = String
RETURN VALUE: Integer or invalid.
EXAMPLE:
var a = "ABC";
var b = String.length(a); // b = 3
var c = String.length(" "); // c = 0
var d = String.length(342); // d = 3

9.2 isEmpty

FUNCTION: isEmpty(string)
FUNCTION ID: 1
DESCRIPTION: Returns a boolean true if the string length is zero and boolean false otherwise.
PARAMETERS: string = String
RETURN VALUE: Boolean or invalid.
EXAMPLE:
var a = "Hello";
var b = "";
var c = String.isEmpty(a); // c = false;
var d = String.isEmpty(b); // d = true
var e = String.isEmpty(true); // e = false

9.3 charAt

FUNCTION: charAt(string, index)
FUNCTION ID: 2
DESCRIPTION: Returns a new string of length one containing the character at the specified index of the given string.
If the index is of type floating-point, Float.int() is first used to calculate the actual integer index.
PARAMETERS: string = String
index = Number (the index of the character to be returned)
RETURN VALUE: String or invalid.
EXAMPLE:
var a = "My name is Joe";
var b = String.charAt(a, 0); // b = "M"
var c = String.charAt(a, 100); // c = ""
var d = String.charAt(34, 0); // d = "3"
var e = String.charAt(a, "first"); // e = invalid

9.4 subString

FUNCTION: subString(string, startIndex, length)
FUNCTION ID: 3
DESCRIPTION: Returns a new string that is a substring of the given string. The substring begins at the specified startindex and its length (number of characters) is the given length. If the startindex is less than 0 then 0 is used for the startindex. If the length is larger than the remaining number of characters in the string, the length is replaced with the number of remaining characters.

If the startindex or the length is of type floating-point, Float.int() is first used to calculate the actual integer value.

PARAMETERS:

- string = String
- startindex = Number (the beginning index, inclusive)
- length = Number (the length of the substring)

RETURN VALUE: String or invalid.

EXAMPLES:

```java
var a = "ABCD";
var b = String.subString(a, 1, 2); // b = "BC"
var c = String.subString(a, 2, 5); // c = "CD"
var d = String.subString(1234, 0, 2); // d = "12"
```

9.5 find

FUNCTION: find(string, subString)

DESCRIPTION: Returns the index of the first character in the string that matches the requested subString. If no match is found, integer value –1 is returned.

Two strings are defined to match when they are identical. Characters with multiple possible representations match only if they have the same representation in both strings. No case folding is performed.

PARAMETERS:

- string = String
- subString = String

RETURN VALUE: Integer or invalid.

EXAMPLES:

```java
var a = "abcde";
var b = String.find(a, "cd"); // b = 2
var c = String.find(34.2, "de"); // c = -1
var d = String.find(a, "qz"); // d = -1
var e = String.find(34, "3"); // e = 0
var f = String.find(a, ""); // f = invalid
```

9.6 replace

FUNCTION: replace(string, oldSubString, newSubString)
FUNCTION: 5
ID:  
DESCRIPTION: Returns a new string resulting from replacing all occurrences of
oldSubString in this string with newSubString.

Two strings are defined to match when they are identical. Characters with
multiple possible representations match only if they have the same
representation in both strings. No case folding is performed.

PARAMETERS:  
string = String
oldSubString = String
newSubString = String

RETURN VALUE: String or invalid.

EXAMPLE:  
var a = "Hello Joe. What is up Joe?";
var newName = "Don";
var oldName = "Joe";
var c = String.replace(a, oldName, newName);
  // c = "Hello Don. What is up Don?"
var d = String.replace(a, newName, oldName);
  // d = "Hello Joe. What is up Joe?"

9.7 elements

FUNCTION: elements(string, separator)

FUNCTION ID: 6
DESCRIPTION: Returns the number of elements in the given string separated by the given
separator. Empty string (""") is a valid element (thus, this function can never
return a value that is less or equal to zero).

PARAMETERS:  
string = String
separator = String (the first character of the string used as separator)

RETURN VALUE: Integer or invalid.

EXAMPLE:  
var a = "My name is Joe; Age 50;"
var b = String.elements(a, " ");
  // b = 6
var c = String.elements(a, ";");
  // c = 3
var d = String.elements(" ", ";");
  // d = 1
var e = String.elements("a", ";");
  // e = 1
var f = String.elements(";", ";");
  // f = 1
var g = String.elements(";", ";", ";");  // g = 4 separator = 

9.8 elementAt

FUNCTION: elementAt(string, index, separator)
FUNCTION ID: 7
DESCRIPTION: Search string for index'th element, elements being separated by separator and return the corresponding element. If the index is less than 0 then the first element is returned. If the index is larger than the number of elements then the last element is returned. If the string is an empty string then an empty string is returned.

If the index is of type floating-point, Float.int() is first used to calculate the actual index value.

PARAMETERS:
- string = String
- index = Number (the index of the element to be returned)
- separator = String (the first character of the string used as separator)

RETURN VALUE: String or invalid.

EXAMPLE:

var a = "My name is Joe; Age 50;"
var b = String.elementAt(a, 0, ";"); // b = "My"
var c = String.elementAt(a, 14, ";"); // c = ""
var d = String.elementAt(a, 1, ";"); // d = " Age 50"

9.9 removeAt

FUNCTION: removeAt(string, index, separator)
FUNCTION ID: 8
DESCRIPTION: Returns a new string where the element and the corresponding separator (if existing) with the given index are removed from the given string. If the index is less than 0 then the first element is removed. If the index is larger than the number of elements then the last element is removed. If the string is empty, the function returns a new empty string.

If the index is of type floating-point, Float.int() is first used to calculate the actual index value.

PARAMETERS:
- string = String
- index = Number (the index of the element to be deleted)
- separator = String (the first character of the string used as separator)

RETURN VALUE: String or invalid.

EXAMPLE:

var a = "My name is Joe; Age 50;"
var b = String.elementAt(a, 0, ";"); // b = "My"
var c = String.elementAt(a, 14, ";"); // c = ""
var d = String.elementAt(a, 1, ";"); // d = " Age 50"
EXAMPLE:

```javascript
var a = "A A; B C D";
var s = " ";
var b = String.removeAt(a, 1, s);
// b = "A B C D"
var c = String.removeAt(a, 0, ";");
// c = " B C D"
var d = String.removeAt(a, 14, ";");
// d = "A A"
```

9.10 replaceAt

FUNCTION: replaceAt(string, element, index, separator)

FUNCTION ID: 9

DESCRIPTION: Returns a string with the current element at the specified index replaced with the given element. If the index is less than 0 then the first element is replaced. If the index is larger than the number of elements then the last element is replaced. If the string is empty, the function returns a new string with the given element.

If the index is of type floating-point, Float.int() is first used to calculate the actual index value.

PARAMETERS:
- string = String
- element = String
- index = Number (the index of the element to be replaced)
- separator = String (the first character of the string used as separator)

RETURN VALUE: String or invalid.

EXAMPLE:

```javascript
var a = "B C; E";
var s = " ";
var b = String.replaceAt(a, "A", 0, s);
// b = "A C; E"
var c = String.replaceAt(a, "F", 5, ";");
// c = "B C; F"
```

9.11 insertAt

FUNCTION: insertAt(string, element, index, separator)

FUNCTION ID: 10

DESCRIPTION: Returns a string with the element and the corresponding separator (if needed) inserted at the specified element index of the original string. If the index is less than 0 then 0 is used as the index. If the index is larger than the number of elements then the element is appended at the end of the string. If the string is empty, the function returns a new string with the given element.

If the index is of type floating-point, Float.int() is first used to calculate the actual index value.
PARAMETE:  
string = String (original string)  

RS:  
element = String (element to be inserted)  

index = Number (the index of the element to be added)  

separator = String (the first character of the string used as separator)  

RETURN VALUE:  
String or invalid.  

EXAMPLE:  
var a = "B C; E";  
var s = " ";  
var b = String.insertAt(a, "A", 0, s);  
// b = "A B C; E"  
var c = String.insertAt(a, "X", 3, s);  
// c = "B C; E X"  
var d = String.insertAt(a, "D", 1, ":");  
// d = "B C;D; E"  
var e = String.insertAt(a, "F", 5, ":");  
// e = "B C; E;F"

9.12 squeeze

FUNCTION: squeeze(string)

FUNCTION ID: 11

DESCRIPTION: Returns a string where all consecutive series of white spaces within the string are reduced to single inter-word space.

PARAMETE: String = String

RS: 

RETURN VALUE: String or invalid.

EXAMPLE:  
var a = "Hello";  
var b = " Bye Jon . \r\n See you! ";  
var c = String.squeeze(a); // c = "Hello";  
var d = String.squeeze(b); // d = " Bye Jon . See you! ";

9.13 trim

FUNCTION: trim(string)

FUNCTION ID: 12

DESCRIPTION: Returns a string where all trailing and leading white spaces in the given string have been trimmed.

PARAMETE: String = String

RS: 

RETURN VALUE: String or invalid.
9.14 compare

FUNCTION: compare(string1, string2)

The return value indicates the lexicographic relation of string1 to string2. The relation is based on the relation of the character codes in the native character set. The return value is –1 if string1 is less than string2, 0 if string1 is identical to string2 or 1 if string1 is greater than string2.

PARAMETERS: String1 = String
String2 = String

RETURN VALUE: Integer or invalid.

EXAMPLE:
var a = "Hello";
var b = "Bye    Jon . See you! ";
var c = String.trim(a); // c = "Hello"
var d = String.trim(b); // d = "Bye    Jon . See you!"
var e = String.compare(a, b); // e = 0
var f = String.compare(b, a); // f = -1
var g = String.compare(12, "Hello"); // g = -1
var h = String.compare("Hello", 12); // h = 1
var i = String.compare("Hello", "Hello"); // i = 0
var j = String.compare("Hello", "World"); // j = -1

9.15 toString

FUNCTION: toString(value)

Returns a string representation of the given value. This function performs exactly the same conversions as supported by the [WMLScript] language (automatic conversion from boolean, integer and floating-point values to strings) except that invalid value returns the string "invalid".

PARAMETERS: value = Any

RETURN VALUE: String.

EXAMPLE:
var a = String.toString(12); // a = "12"
var b = String.toString(true); // b = "true"
9.16 format

FUNCTION: format(format, value)
FUNCTION ID: 15
DESCRIPTION: Converts the given value to a string by using the given formatting provided as a format string. The format string can contain only one format specifier, which can be located anywhere inside the string. If more than one is specified, only the first one (leftmost) is used and the remaining specifiers are replaced by an empty string. The format specifier has the following form:

```
% [width] [.precision] type
```

The width argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left until the minimum width is reached. The width argument never causes the value to be truncated. If the number of characters in the output value is greater than the specified width or, if width is not given, all characters of the value are printed (subject to the precision argument).

The precision argument specifies a nonnegative decimal integer, preceded by a period (.), which can be used to set the precision of the output value. The interpretation of this value depends on the given type:

- **d**: Specifies the minimum number of digits to be printed. If the number of digits in the value is less than precision, the output value is padded on the left with zeroes. The value is not truncated when the number of digits exceeds precision. Default precision is 1. If precision is specified as 0 and the value to be converted is 0, the result is an empty string.

- **f**: Specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. Default precision is 6; if precision is 0 or if the period (.) appears without a number following it, no decimal point is printed.

- **s**: Specifies the maximum number of characters to be printed. By default, all characters are printed.

Unlike the width argument, the precision argument can cause either truncation of the output value or rounding of a floating-point value.

The type argument is the only required format argument; it appears after any optional format fields. The type character determines whether the given value is interpreted as integer, floating-point or string. If the value argument is of a different type than is specified by the type argument, it is converted according to WMLScript standard automatic conversion rules, with the addition that if value is of type floating-point and type is d, Float.int() is called to convert the value. The supported type arguments are:

- **d**: Integer: The output value has the form [-]dddd, where dddd is one
or more decimal digits.

**f** Floating-point: The output value has the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number and the number of digits after the decimal point depends on the requested precision. When the number of digits after the decimal point in the value is less than the precision, letter 0 should be padded to fill columns (e.g. the result of `String.format("%2.3f", 1.2) will be "1.200"")

**s** String: Characters are printed up to the end of the string or until the precision value is reached. When the width is larger than precision, the width should be ignored.

A literal percent character (%) may be included in the format string by preceding it with another percent character (%%).

**PARAMETERS:**
- `format` = String
- `value` = Any

**RETURN VALUE:** String or invalid.

**EXCEPTIONS:**
- Illegal format specifier results in an invalid return value.
- If type f is specified in format argument and floating-point numbers are not supported, invalid is returned.

**EXAMPLE:**
```
var a = 45;
var b = -45;
var c = "now";
var d = 1.2345678;
var e = String.format("e: %6d", a);   // e = "e:     45"
var f = String.format("%6d", b);      // f = "   -45"
var g = String.format("%6.4d", a);    // g = "  0045"
var h = String.format("%6.4d", b);    // h = " –0045"
var i = String.format("Do it %s", c); // i = "Do it now"
var j = String.format("%3f", d);      // j = "1.234568"
var k = String.format("%10.2f%%", d); // k = "1.23%"
var l = String.format("%3f %2f.", d); // l = "1.234568 ."
var m = String.format("%0d", 0);      // m = ""
var n = String.format("%7d", "Int"); // n = invalid
var o = String.format("%s", true);    // o = "true"
```
10. URL

NAME: URL
LIBRARY ID: 3
DESCRIPTION: This library contains a set of functions for handling both absolute URLs and relative URLs. The URL syntax supported is defined in [RFC2396]. Currently this library supports access to only a subset of URL elements specified in [RFC2396].

10.1 isValid

FUNCTION: isValid(url)
FUNCTION ID: 0
DESCRIPTION: Returns true if the given url has the right URL syntax, otherwise returns false. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.
PARAMETERS: url = String
RETURN VALUE: Boolean or invalid.
EXAMPLE:
var a = URL.isValid("http://w.hst.com/script#func()" );
   // a = true
var b = URL.isValid("../common#test()" );
   // b = true
var c = URL.isValid("experimental/?://www.host.com/cont>");
   // c = false

10.2 getScheme

FUNCTION: getScheme(url)
FUNCTION ID: 1
DESCRIPTION: Returns the scheme used in the given url. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.
PARAMETERS: url = String
RETURN VALUE: String or invalid.
EXAMPLE:
If an invalid URL syntax is encountered while extracting the scheme an invalid value is returned.
EXAMPLE: var a = URL.getScheme("http://w.h.com/path#frag");
            // a = "http"
var b = URL.getScheme("w.h.com/path#frag");
            // b = ""

10.3 getHost

FUNCTION: getHost(url)
FUNCTION ID: 2
DESCRIPTION: Returns the host specified in the given url. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs. If the host part of the URL is not defined, the function returns an empty string.

PARAMETERS: url = String
RETURN VALUE: String or invalid.
EXAMPLES: var a = URL.getHost("http://w.h.com/path#frag");
              // a = "w.h.com"
var b = URL.getHost("path#frag");
              // b = ""
var c = URL.getHost("zyx://me@ismo.k.oksa#fab");
              // c = "ismo.k.oksa"

10.4 getPort

FUNCTION: getPort(url)
FUNCTION ID: 3
DESCRIPTION: Returns the port number specified in the given url. If no port is specified then an empty string is returned. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.

PARAMETERS: url = String
RETURN VALUE: String or invalid.
EXAMPLES: var a = URL.getPort("http://w.h.com:80/path#frag");
              // a = "80"
var b = URL.getPort("http://w.h.com/path#frag");
              // b = ""
10.5 getPath

FUNCTION: getPath(url)

FUNCTION ID: 4

DESCRIPTION: Returns the path specified in the given url. Parameters specified for each path segment, if any, are not returned. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.

PARAMETERS: url = String

RETURN VALUE: String or invalid.

EXAMPLE: a = URL.getPath("http://w.h.com/home/sub/comp#frag");
         // a = "/home/sub/comp"
   b = URL.getPath("../home/sub/comp#frag");
         // b = "/../home/sub/comp"
   c = URL.getPath("http://w.a.p/a;x/b;y=1/c;fg#a");
         // c = "/a/b/c"
   d = URL.getPath("http://w.a.p/a;x/b;y=1/c#b");
         // d = "/a/b/c"

10.6 getParameters

FUNCTION: getParameters(url)

FUNCTION ID: 5

DESCRIPTION: Returns the parameters used in the last path segment of the given url. If no parameters are specified an empty string is returned. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.

NOTE: This version of this function does not take into account the possibility for each segment to have parameters (see [RFC2396] for more information). Only the parameters specified for the last segment are returned. This may change in the future.

PARAMETERS: url = String

RETURN VALUE: String or invalid.

EXAMPLE: a = URL.getPath("http://w.h.com/home/sub/comp#frag");
         // a = "/home/sub/comp"
   b = URL.getPath("../home/sub/comp#frag");
         // b = "/../home/sub/comp"
   c = URL.getPath("http://w.a.p/a;x/b;y=1/c;fg#a");
         // c = "/a/b/c"
   d = URL.getPath("http://w.a.p/a;x/b;y=1/c#b");
         // d = "/a/b/c"
EXAMPLE: 

```java
a = URL.getParameters("http://w.h.com/script;3;2?x=1&y=3");
//  a = "3;2"
b = URL.getParameters("../script;3;2?x=1&y=3");
//  b = "3;2"
c = URL.getParameters("http://w.a.p/a;x/b;y=1/c;fg");
//  c = "fg"
d = URL.getParameters("http://w.a.p/a;x/b;y=1/c");
//  d = ""
```

10.7 getQuery

FUNCTION: getQuery(url)

FUNCTION ID: 6

DESCRIPTION: Returns the query part specified in the given url. If no query part is specified an empty string is returned. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.

PARAMETERS: 

url = String

RETURN VALUE: String or invalid.

EXCEPTIONS: If an invalid URL syntax is encountered while extracting the query part an invalid value is returned.

EXAMPLE: 

```java
a = URL.getQuery("http://w.h.com/home;3;2?x=1&y=3");
//  a = "x=1&y=3"
```

10.8 getFragment

FUNCTION: getFragment(url)

FUNCTION ID: 7

DESCRIPTION: Returns the fragment used in the given url. If no fragment is specified an empty string is returned. Both absolute and relative URLs are supported. Relative URLs are not resolved into absolute URLs.

PARAMETERS: 

url = String

RETURN VALUE: String or invalid.

EXCEPTIONS: If an invalid URL syntax is encountered while extracting the fragment an invalid value is returned.

EXAMPLE: 

```java
var a = URL.getFragment("http://w.h.com/cont#frag");
//  a = "frag"
```

10.9 getBase

FUNCTION: getBase()
FUNCTION: 8
ID:
DESCRIPTION: Returns an absolute URL (without the fragment) of the current WMLScript compilation unit.
PARAMETERS:
RETURN VALUE:
EXCEPTIONS:
EXAMPLE:

```
var a = URL.getBase();
// Result: "http://www.host.com/test.scr"
```

10.10 getReferer

FUNCTION: getReferer()
FUNCTION ID: 9
DESCRIPTION: Returns the smallest relative URL (relative to the base URL of the current compilation unit, see 10.9) to the resource that called the current compilation unit. Local function calls do not change the referer. If the current compilation unit does not have a referer then an empty string is returned.
PARAMETERS:
RETURN VALUE:
EXCEPTIONS:
EXAMPLE:

```
var base = URL.getBase();
// base = "http://www.host.com/current.scr"
var referer = URL.getReferer();
// referer = "app.wml#card2"
```

10.11 resolve

FUNCTION: resolve(baseUrl, embeddedUrl)
FUNCTION ID: 10
DESCRIPTION: Returns an absolute URL from the given baseUrl and the embeddedUrl according to the rules specified in [RFC2396]. Before executing the rules specified in [RFC2396] the baseUrl is checked. If the baseUrl's path component is an empty string, then a single slash character ("/") is assumed as the path. If the embeddedUrl is already an absolute URL, the function returns it without modification.
PARAMETERS:
baseUrl = String
embeddedUrl = String
RETURN VALUE:
String or invalid.
**exception**  If an invalid URL syntax is encountered as part of the resolution an invalid value is returned.

**example:**

```javascript
var a = URL.resolve("http://foo.com/","foo.vcf");
// a = "http://foo.com/foo.vcf"
var b = URL.resolve("http://foo.com","c");
// b = "http://foo.com/c"
var c = URL.resolve("http://foo.com","/c");
// c = "http://foo.com/c"
var d = URL.resolve("http://foo.com","?q");
// d = "http://foo.com/?q"
var e = URL.resolve("http://", "x");
// e = "http://x"
```

### 10.12 escapeString

**function:** escapeString(string)

**function id:** 11

**description:** This function computes a new version of a string value in which special characters specified by [RFC2396] have been replaced by a hexadecimal escape sequence (a two-digit escape sequence of the form %xx must be used). The characters to be escaped are:

- **Control characters:** <US-ASCII coded characters 00-1F and 7F>
- **Space:** <US-ASCII coded character 20 hexadecimal>
- **Reserved:** "*" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
- **Unwise:** "{" | "}" | "<" | ">" | ":" | ";" | "#" | "^" | "_" | "\"
- **Delims:** ";" | ";" | ":" | "=" | ";" | ";"
- **Non-US-ASCII:** <characters with hex code 8F-FF>

The given string is escaped as such; no URL parsing is performed. Non-US-ASCII characters must be converted using the character codes used in the native character set.

**parameters:**

- `string` = String

**return value:** String or invalid.

**exceptions:**

- If `string` contains characters with a character codes above hex FF an invalid value is returned.

**example:**

```javascript
var a = URL.escapeString("http://w.h.com/dck?q=x\u007f#crd");
// a = "http%3a%2f%2fw.h.com%2fdck%3fx%3d7f%23crd"
var b = URL.escapeString("http://w.h.com/dck?q=x\u007f#crd");
// b = "http%3a%2f%2fw.h.com%2fdck%3fx%3d5cu007f%23crd"
```
10.13 unescapeString

FUNCTION: UnescapeString(string)

FUNCTION ID: 12

DESCRIPTION: The unescape function computes a new version of a string value in which each escape sequences of the sort that might be introduced by the URL.escapeString() function (see 10.12) is replaced with the character that it represents. The given string is unescaped as such; no URL parsing is performed.

PARAMETERS: string = String

RETURN VALUE: String or invalid.

EXCEPTIONS: If string contains characters that are not part of the US-ASCII character set, an invalid value is returned.

EXAMPLE:

var a = "http%3a%2f%2fw.h.com%2fdck%3fx%3d12%23crd";
var b = URL.unescapeString(a);
// b = "http://w.h.com/dck?x=12#crd"
10.14 **loadString**

**FUNCTION:** loadString(url, contentType)

**FUNCTION ID:** 13

**DESCRIPTION:** Returns the content denoted by the given absolute `url` and the `content type`. Regardless of what content types the user agent supports, it MUST expect to receive "text" content as a result of executing URL.loadString. The user agent MUST preface the request with an Accept header which consists ONLY of the content type specified by `contentType`.

The given `content type` is erroneous if it does not follow the following rules:

- Only one content type can be specified. The whole string must match with only one content type and no extra leading or trailing spaces are allowed.

- The type must be `text` but the subtype can be anything. Thus, the type prefix must be "text/".

The behaviour of this function is the following:

- The content with the given `content type` and `url` is loaded. The rest of the attributes needed for the content load are specified by the default settings of the user agent.

- If the load is successful and the returned content type matches the given `content type` then the content is converted to a string and returned.

- If the load is unsuccessful or the returned content is of wrong content type then a scheme specific error code is returned.

**PARAMETERS:**

- `url` = String
- `contentType` = String

**RETURN VALUE:** String, integer or invalid.

**EXCEPTIONS:** Returns an integer `error code` that depends on the used URL scheme in case the load fails. If HTTP [RFC2068] or WSP (see [WAE]) schemes are used, HTTP error codes are returned.

If an erroneous `content type` is given, an invalid value is returned.

**EXAMPLE:**

```javascript
var myUrl = "http://www.host.com/vcards/myaddr.vcf";
myCard = URL.loadString(myUrl,"text/x-vcard");
```
11. **WMLBROWSER**

**NAME:** WMLBrowser  
**LIBRARY ID:** 4  
**DESCRIPTION:** This library contains functions by which WMLScript can access the associated WML context. These functions must not have any side effects and they must return invalid in the following cases:

If the system does not have a WML browser, or if the WMLScript interpreter was not invoked by the WML browser, these functions must always return invalid.

11.1 **getVar**

**FUNCTION:** getVar(name)  
**FUNCTION ID:** 0  
**DESCRIPTION:** Returns the value of the variable with the given name in the current browser context. Returns an empty string if the given variable does not exist.

Variable name must follow the syntax specified by [WML].

**PARAMETERS:** name = String  
**RETURN VALUE:** String or invalid.  
**EXCEPTIONS:** If the syntax of the variable name is incorrect an invalid value is returned.

**EXAMPLE:** var a = WMLBrowser.getVar("name");  
// a = "Jon" or whatever value the variable has.

11.2 **setVar**

**FUNCTION:** setVar(name, value)  
**FUNCTION ID:** 1  
**DESCRIPTION:** Returns true if the variable with the given name is successfully set to contain the given value in the current browser context, false otherwise.

Variable name and its value must follow the syntax specified by [WML]. Variable value must be legal XML CDATA.

**PARAMETERS:** name = String  
value = String  
**RETURN VALUE:** Boolean or invalid.  
**EXCEPTIONS:** If the syntax of the variable name or its value is incorrect an invalid value is returned.

**EXAMPLE:** var a = WMLBrowser.setVar("name", Mary);  
// a = true
11.3 go

FUNCTION: go(url)
FUNCTION ID: 2
DESCRIPTI ON: Specifies the content denoted by the given url to be loaded. This function has the same semantics as the GO task in WML (see [WML] for more information). The content is loaded only after the WML browser resumes the control back from the WMLScript interpreter after the WMLScript invocation is finished. When the WML browser loads the content, the referring URI is the URI of the current card. If a relative URI is given as the url, the WML browser must resolve it by using the URI of the current card. No content is loaded if the given url is an empty string ('"').

   go() and prev() (see 11.3) library functions override each other. Both of these library functions can be called multiple times before returning the control back to the WML browser. However, only the settings of the last call stay in effect. In particular, if the last call to go() sets the URL to an empty string ('"'), all previous go() and prev() requests are effectively cancelled.

   Invoking the Lang.abort() function (see 7.12) along with any other fatal errors (see [WMLScript]) cancels any pending go() request.

This function returns an empty string.

PARAMETERS: url = String
RETURN VALUE: String or invalid.
EXCEPTIONS: -
EXAMPLE:    extern function goToStart() {
        var card = "http://www.host.com/loc/app.dck#start";
        WMLBrowser.go(card);
    };

    extern function get() {
        WMLBrowser.go("#next_card");
        return;
    };

    // If the above function is invoked from the following
    // WML fragment:
    // ..
    // <card id="referring_card" >
    // ..
    // <go href="myscript#get()"/>
    // ..
    // </card>
    // ..
    // The referring URI will be the URI of the WML card that
    // invoked the function go and fulfils the script's request
    // (i.e., the card with the id "referring_card").

11.4 prev

    FUNCTION: prev()  
    FUNCTION ID: 3  
    DESCRIPTION: Signals the WML browser to go back to the previous WML card. This function has the same semantics as the PREV task in WML (see [WML] for more information). The previous card is loaded only after the WML browser resumes the control back from the WMLScript interpreter after the WMLScript invocation is finished.

    prev() and go() (see 11.3) library functions override each other. Both of these library functions can be called multiple times before returning the control back to the WML browser. However, only the settings of the last call stay in effect. In particular, if the last call to go()sets the URL to an empty string (""), all previous go() and prev() requests are effectively cancelled.

    Invoking the Lang.abort() function (see 7.12) along with any other fatal errors (see [WMLScript]) cancels any pending prev() request.

    This function returns an empty string.

    PARAMETER: -  
    RETURN VALUE: String or invalid.
11.5 newContext

FUNCTION: newContext()

FUNCTION ID: 4

DESCRIPTION: A call to this function clears all variables of the associated WML context and clears the navigation history stack (see [WML] for more information) except for the current card before returning execution to the calling entity. The function does not impact a navigation request from a previous or subsequent go() call. A previous or subsequent prev() call will have no effect. The function returns an empty string.

PARAMETERS: -

RETURN VALUE: String or invalid.

EXAMPLE: WMLBrowser.newContext();

11.6 getCurrentCard

FUNCTION: getCurrentCard()

FUNCTION ID: 5

DESCRIPTION: Returns the smallest relative URL (relative to the base of the current compilation unit, see 10.9 for information about how to access the current base) specifying the card (if any) currently being processed by the WML Browser (see [WML] for more information). The function returns an absolute URL in case the WML deck containing the current card does not have the same base as the current compilation unit.

PARAMETERS: -

RETURN VALUE: String or invalid.

EXAMPLE: var a = WMLBrowser.getCurrentCard();
// a = "deck#input"

11.7 refresh

FUNCTION: refresh()

FUNCTION ID: 6
DESCRIPTION: The WML Browser should update its user interface based on the current context. Implementations that support refresh must perform the steps defined in [WML] with the exception to restarting a suspended timer. This function must not restart a suspended timer. This function must block until all the steps have completed. The refresh actions must be applied to the current WML card. If the current WML card was not rendered prior to invoking this call (e.g., a script that was invoked by an onenterforward event binding), the refresh function must render the current card.

This function returns invalid if the current implementation does not support immediate refresh. Otherwise, this function returns an empty string if the refresh succeeds, or a non-empty string if it fails (e.g., failed to update an image). The content of the string is implementation dependent. The function should return a brief message explaining the error.

Note: if the current implementation does not support refresh, the WML user agent must still refresh the card when control returns back to the WML user agent.

PARAMETERS: -

RETURN VALUE: String or invalid.

EXAMPLE:

```javascript
WMLBrowser.setVar("name","Zorro");
var refreshOK = WMLBrowser.refresh();
```
12. DIALOGS

NAME: Dialogs
LIBRARY ID: 5
DESCRIPTION: This library contains a set of typical user interface functions.

12.1 prompt

FUNCTION: prompt(message, defaultInput)
FUNCTION ID: 0
DESCRIPTION: Displays the given message and prompts for user input. The defaultInput parameter contains the initial content for the user input. Returns the user input.
PARAMETERS: message = String
defaultInput = String
RETURN VALUE: String or invalid.
EXAMPLE: var a = "09-555 3456";
var b = Dialogs.prompt("Phone number: ", a);

12.2 confirm

FUNCTION: confirm(message, ok, cancel)
FUNCTION ID: 1
DESCRIPTION: Displays the given message and two reply alternatives: ok and cancel. Waits for the user to select one of the reply alternatives and returns true for ok and false for cancel.
PARAMETERS: message = String
ok = String (text, empty string results in the default implementation-dependent text)
cancel = String (text, empty string results in the default implementation-dependent text)
RETURN VALUE: Boolean or invalid.
EXAMPLE: function onAbort() {
    return Dialogs.confirm("Are you sure?","Yes","Well...");
}
12.3 alert

FUNCTION: alert(message)
FUNCTION ID: 2
DESCRIPTION: Displays the given message to the user, waits for the user confirmation and returns an empty string.
PARAMETERS: message = String
RETURN VALUE: String or invalid.
EXAMPLE: function testValue(textElement) {
    if (String.length(textElement) > 8) {
        Dialogs.alert("Enter name < 8 chars!");
    }
};
}
Appendix A. Library Summary

The libraries and their library identifiers:

<table>
<thead>
<tr>
<th>Library name</th>
<th>Library ID</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lang</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Float</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>String</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>URL</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>WMLBrowser</td>
<td>4</td>
<td>44</td>
</tr>
<tr>
<td>Dialogs</td>
<td>5</td>
<td>49</td>
</tr>
</tbody>
</table>

The libraries and their functions:

**Lang library**

<table>
<thead>
<tr>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>0</td>
</tr>
<tr>
<td>min</td>
<td>1</td>
</tr>
<tr>
<td>max</td>
<td>2</td>
</tr>
<tr>
<td>parseInt</td>
<td>3</td>
</tr>
<tr>
<td>parseFloat</td>
<td>4</td>
</tr>
<tr>
<td>isInt</td>
<td>5</td>
</tr>
<tr>
<td>isFloat</td>
<td>6</td>
</tr>
<tr>
<td>maxInt</td>
<td>7</td>
</tr>
<tr>
<td>minInt</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>9</td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
</tr>
<tr>
<td>abort</td>
<td>11</td>
</tr>
<tr>
<td>random</td>
<td>12</td>
</tr>
<tr>
<td>seed</td>
<td>13</td>
</tr>
<tr>
<td>characterSet</td>
<td>14</td>
</tr>
</tbody>
</table>

**Float library**

<table>
<thead>
<tr>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>floor</td>
<td>1</td>
</tr>
<tr>
<td>ceil</td>
<td>2</td>
</tr>
<tr>
<td>pow</td>
<td>3</td>
</tr>
<tr>
<td>round</td>
<td>4</td>
</tr>
<tr>
<td>sqrt</td>
<td>5</td>
</tr>
<tr>
<td>maxFloat</td>
<td>6</td>
</tr>
<tr>
<td>minFloat</td>
<td>7</td>
</tr>
</tbody>
</table>
## String library

<table>
<thead>
<tr>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>0</td>
</tr>
<tr>
<td>isEmpty</td>
<td>1</td>
</tr>
<tr>
<td>charAt</td>
<td>2</td>
</tr>
<tr>
<td>subString</td>
<td>3</td>
</tr>
<tr>
<td>find</td>
<td>4</td>
</tr>
<tr>
<td>replace</td>
<td>5</td>
</tr>
<tr>
<td>elements</td>
<td>6</td>
</tr>
<tr>
<td>elementAt</td>
<td>7</td>
</tr>
<tr>
<td>removeAt</td>
<td>8</td>
</tr>
<tr>
<td>replaceAt</td>
<td>9</td>
</tr>
<tr>
<td>insertAt</td>
<td>10</td>
</tr>
<tr>
<td>squeeze</td>
<td>11</td>
</tr>
<tr>
<td>trim</td>
<td>12</td>
</tr>
<tr>
<td>compare</td>
<td>13</td>
</tr>
<tr>
<td>toString</td>
<td>14</td>
</tr>
<tr>
<td>format</td>
<td>15</td>
</tr>
</tbody>
</table>

## URL library

<table>
<thead>
<tr>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>isValid</td>
<td>0</td>
</tr>
<tr>
<td>getScheme</td>
<td>1</td>
</tr>
<tr>
<td>getHost</td>
<td>2</td>
</tr>
<tr>
<td>getPort</td>
<td>3</td>
</tr>
<tr>
<td>getPath</td>
<td>4</td>
</tr>
<tr>
<td>getParameters</td>
<td>5</td>
</tr>
<tr>
<td>getQuery</td>
<td>6</td>
</tr>
<tr>
<td>getFragment</td>
<td>7</td>
</tr>
<tr>
<td>getBase</td>
<td>8</td>
</tr>
<tr>
<td>getReferer</td>
<td>9</td>
</tr>
<tr>
<td>resolve</td>
<td>10</td>
</tr>
<tr>
<td>escapeString</td>
<td>11</td>
</tr>
<tr>
<td>unescapeString</td>
<td>12</td>
</tr>
<tr>
<td>loadString</td>
<td>13</td>
</tr>
</tbody>
</table>

## WMLBrowser library

<table>
<thead>
<tr>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>getVar</td>
<td>0</td>
</tr>
<tr>
<td>setVar</td>
<td>1</td>
</tr>
<tr>
<td>go</td>
<td>2</td>
</tr>
<tr>
<td>prev</td>
<td>3</td>
</tr>
<tr>
<td>newContext</td>
<td>4</td>
</tr>
<tr>
<td>getCurrentCard</td>
<td>5</td>
</tr>
<tr>
<td>refresh</td>
<td>6</td>
</tr>
</tbody>
</table>
Appendix B. Static Conformance Requirements

This static conformance clause defines a minimum set of features that can be implemented to ensure that WMLScript Standard Libraries will be able to inter-operate.

12.4 WMLScript Encoder Capabilities

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-001</td>
<td>Supports Lang library and all of its functions</td>
<td>Lang</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-002</td>
<td>Supports Float library and all of its functions</td>
<td>Float</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-003</td>
<td>Supports String library and all of its functions</td>
<td>String</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-004</td>
<td>Supports URL library and all of its functions</td>
<td>URL</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-005</td>
<td>Supports WMLBrowser library and all of its functions</td>
<td>WMLBrowser</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-006</td>
<td>Supports Dialogs library and all of its functions</td>
<td>Dialogs</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-007</td>
<td>Supports all library identifiers for standard libraries</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-008</td>
<td>Supports Lang library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-009</td>
<td>Supports Float library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-010</td>
<td>Supports String library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

Dialogs library

<table>
<thead>
<tr>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompt</td>
</tr>
<tr>
<td>confirm</td>
</tr>
<tr>
<td>alert</td>
</tr>
<tr>
<td>Identifier</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>WMLSSL-011</td>
</tr>
<tr>
<td>WMLSSL-012</td>
</tr>
<tr>
<td>WMLSSL-013</td>
</tr>
</tbody>
</table>
### 12.5 WMLScript Bytecode Interpreter Capabilities

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-014</td>
<td>Supports WMLScript data types integer, boolean, string, invalid and float</td>
<td>Supported Data Type</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-015</td>
<td>Supports automatic type conversions</td>
<td>Data Type Conversions</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-016</td>
<td>Supports error handling</td>
<td>Error Handling</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-017</td>
<td>Supports floating point operations</td>
<td>Support for Integer-Only Devices</td>
<td>O</td>
<td>WMLS-111</td>
</tr>
<tr>
<td>WMLSSL-018</td>
<td>Supports Lang library</td>
<td>Lang</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-019</td>
<td>Supports Float library</td>
<td>Float</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-020</td>
<td>Supports String library</td>
<td>String</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-021</td>
<td>Supports URL library</td>
<td>URL</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-022</td>
<td>Supports WMLBrowser library</td>
<td>WMLBrowser</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-023</td>
<td>Supports Dialogs library</td>
<td>Dialogs</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-024</td>
<td>Supports all library identifiers for standard libraries</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-025</td>
<td>Supports Lang library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-026</td>
<td>Supports Float library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-027</td>
<td>Supports String library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-028</td>
<td>Supports URL library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-029</td>
<td>Supports WMLBrowser library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-030</td>
<td>Supports Dialogs library function identifiers</td>
<td>Appendix A. Library Summary</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Function</td>
<td>Reference</td>
<td>Status</td>
<td>Requirement</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>WMLSSL-031</td>
<td>abs function</td>
<td>abs</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-032</td>
<td>min function</td>
<td>min</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-033</td>
<td>max function</td>
<td>max</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-034</td>
<td>parseInt function</td>
<td>parseInt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-035</td>
<td>parseFloat function</td>
<td>parseFloat</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-036</td>
<td>isInt function</td>
<td>isInt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-037</td>
<td>isFloat function</td>
<td>isFloat</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-038</td>
<td>maxInt function</td>
<td>maxInt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-039</td>
<td>minInt function</td>
<td>minInt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-040</td>
<td>float function</td>
<td>float</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-041</td>
<td>exit function</td>
<td>exit</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-042</td>
<td>abort function</td>
<td>abort</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-043</td>
<td>random function</td>
<td>random</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-044</td>
<td>seed function</td>
<td>seed</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-045</td>
<td>characterSet function</td>
<td>characterSet</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

### Float Library

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-046</td>
<td>All functions return invalid if floating point not supported</td>
<td>Float</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-047</td>
<td>int function</td>
<td>int</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-048</td>
<td>floor function</td>
<td>floor</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-049</td>
<td>ceil function</td>
<td>ceil</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-050</td>
<td>pow function</td>
<td>pow</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-051</td>
<td>round function</td>
<td>round</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-052</td>
<td>sqrt function</td>
<td>sqrt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-053</td>
<td>maxFloat function</td>
<td>maxFloat</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-054</td>
<td>minFloat function</td>
<td>minFloat</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

### String Library

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-055</td>
<td>length function</td>
<td>length</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Function</td>
<td>Reference</td>
<td>Status</td>
<td>Requirement</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>WMLSSL-056</td>
<td>isEmpty function</td>
<td>isEmpty</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-057</td>
<td>charAt function</td>
<td>charAt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-058</td>
<td>subString function</td>
<td>subString</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-059</td>
<td>find function</td>
<td>find</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-060</td>
<td>replace function</td>
<td>replace</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-061</td>
<td>elements function</td>
<td>elements</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-062</td>
<td>elementAt function</td>
<td>elementAt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-063</td>
<td>removeAt function</td>
<td>removeAt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-064</td>
<td>replaceAt function</td>
<td>replaceAt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-065</td>
<td>insertAt function</td>
<td>insertAt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-066</td>
<td>squeeze function</td>
<td>squeeze</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-067</td>
<td>trim function</td>
<td>trim</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-068</td>
<td>compare function</td>
<td>compare</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-069</td>
<td>toString function</td>
<td>toString</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-070</td>
<td>format function</td>
<td>format</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-071</td>
<td>isValid function</td>
<td>isValid</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-072</td>
<td>getScheme function</td>
<td>getScheme</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-073</td>
<td>getHost function</td>
<td>getHost</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-074</td>
<td>getPort function</td>
<td>getPort</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-075</td>
<td>getPath function</td>
<td>getPath</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-076</td>
<td>getParameters function</td>
<td>getParameters</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-077</td>
<td>getQuery function</td>
<td>getQuery</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-078</td>
<td>getFragment function</td>
<td>getFragment</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-079</td>
<td>getBase function</td>
<td>getBase</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-080</td>
<td>getRefferer function</td>
<td>getRefferer</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-081</td>
<td>resolve function</td>
<td>resolve</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-082</td>
<td>escapeString function</td>
<td>escapeString</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-083</td>
<td>unescapeString function</td>
<td>unescapeString</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-084</td>
<td>loadString function</td>
<td>loadString</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Function</td>
<td>Reference</td>
<td>Status</td>
<td>Requirement</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>--------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>WMLSSL-085</td>
<td>getVar function</td>
<td>getVar</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-086</td>
<td>setVar function</td>
<td>setVar</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-087</td>
<td>go function</td>
<td>go</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-088</td>
<td>prev function</td>
<td>prev</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-089</td>
<td>newContext function</td>
<td>newContext</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-090</td>
<td>getCurrentCard function</td>
<td>getCurrentCard</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-091</td>
<td>refresh function</td>
<td>refresh</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

**Dialogs Library**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMLSSL-092</td>
<td>prompt function</td>
<td>prompt</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-093</td>
<td>confirm function</td>
<td>confirm</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WMLSSL-094</td>
<td>alert function</td>
<td>alert</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>