WAP Persistent Storage Interface

Version 30-May-2001

Wireless Application Protocol
WAP-227-PSTOR-20010530-a

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-227-PSTOR-20010530-a	Current

Contents

TABLE OF FIGURES

TABLES OF TABLES

1. Scope

The Wireless Application Protocol (WAP) is a result of continuous work to define an industry wide standard for developing applications that operate over wireless communication networks. The scope of the WAP Forum^{TM} is to define a set of specifications to be used by service applications. The wireless market is growing very quickly, and reaching new customers and services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation, the WAP Forum^{TM} defines a set of protocols in transport, security, transaction, session and application layers. For additional information on the WAP architecture, please refer to "*Wireless Application Protocol Architecture Specification*" [WAPARCH].

This specification describes the WAP Persistent Storage Base Level Storage Interface. This base level interface specifies a means to access, store and retrieve persistent data. The base persistent store is not aware of the structure or format of the stored data.

2. References

2.1. Normative References

[CREQ]	“Specification of WAP Conformance Requirements”. WAP Forum™. WAP-221-CREQ-20000915-a. 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
[RFC2046]	“Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”. N. Freed & N. Borenstein. November 1996. URL: http://www.ietf.org/rfc/rfc2046.txt
[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. August 1998. URL: http://www.ietf.org/rfc/rfc2396.txt
[RFC2616]	“Hypertext Transfer Protocol -- HTTP/1.1”. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt
[UAPROF]	“Wireless Application Group User Agent Profile Specification”. WAP Forum™. WAP-174-UAProf-19991110-a. URL: http://www.wapforum.org/
[WAE]	“Wireless Application Environment Specification”. WAP Forum™. WAP-236-WAESpec. URL: http://www.wapforum.org/
[WMLS]	“WMLScript Language Specification”. WAP Forum™. WAP-193-WMLScript-20000324-a. URL: http://www.wapforum.org/

2.2. Informative References

[RFC2234]	“Augmented BNF for Syntax Specifications: ABNF”. D. Crocker, Ed., P. Overell. November 1997. URL:http://www.ietf.org/rfc/rfc2234.txt
[WAPARCH]	“WAP Architecture”. WAP Forum™. WAP-210-WAPArch-20001130-p. 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

Access Controls
Software or hardware methods used in controlling user access of storage elements through passwords or other security mechanisms.

Application
An application is the executable or interpretable code that is running within the application environment (as described in [WAE]).

Attribute
Attributes are used to describe the characteristics of a storage element.

Byte
A byte is a binary storage unit that is eight bits in size (also called an octet).

Directory
Directories are the storage elements used to group together and hierarchically organise storage objects and other directories.

Null value
A special value, or mark, that is used to indicate the absence of any data value.

Persistent Storage
Persistent storage is a repository for statically stored data.

Static
Static describes a data storage device that retains information while the power is turned off.

Storage Device
A storage device is a physical unit that offers the capability to store data in an organised manner that can be subsequently retrieved. A storage device may be either removable or permanently attached.

Storage Element
A storage element is the basic storage primitive that can be individually selected and manipulated.

Storage Medium
The storage medium is the storage element that represents the physical storage device that is capable of storing persistent data.

Storage Object
A storage object is the storage element used to store persistent data.

Storage System
An integrated collection of services that are used to manage data storage that offer the capability to create, store, retrieve, delete and organise persistently stored data.

WAP client
The WAP client is the physical unit where WAE executes.

WMLScript
WMLScript is a lightweight language specified in WAP for programming a WAP client device [WMLS].

3.3. Abbreviations

API	Application Programming Interface
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electrically-Erasable Programmable Read Only Memory
RAM	Random Access Memory
RO	Read Only
ROM	Read Only Memory
R/W	Read/Write
SIM	Subscriber Identity Module
SRAM	Static Random Access Memory
TBD	To Be Defined
URI	Universal Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WPS	WAP Persistent Storage

4. Introduction

This document specifies a standard set of storage services that are coupled with a well-defined interface, used for organising, accessing, storing and retrieving data that is stored persistently. A brief list of the benefits to WAP applications resulting from the persistent storage system includes: retain information between sessions, share information with other applications and personalise an application using initialisation information.

The goal of this document is to describe a “base level” persistent storage system that consists of a set of storage primitives and a collection of services used to manipulate the storage primitives.

4.1. Design Goals

The following design goals were used in guiding the design of the persistent storage system.

- **General Purpose**: The API must provide features enabling a WAE application to store and retrieve persistent data.
- **Simplicity**: The API should be easy to understand and use. To that extent, the number of function calls will be minimised, and the structure of the stored data will be intentionally ignored.
- **Portable**: The concepts presented in this API must be portable across various environments.
- **Consistent**: This API must be internally consistent in philosophy to other libraries previously defined by the WAP Forum™. Thus, the library is defined, first and foremost, as a WMLScript library in naming, usage, and error model. Additionally, the storage system will be designed to allow a function to operate consistently on every storage element that is supported by the storage system.
- **Familiar**: Current practice suggests a level of conformity with other similar systems and thus defines obvious usage.
- **Flexible**: The system must not impose rigorous “standard” types of data to be stored, nor should it attempt the understanding of such data. To this end, data is not referenced internally or parsed for content. Understanding of structure and content are left to the application.
- **Efficient**: The API must not mandate functionality that typically requires initialisation, dictates CPU usage, or expects large memory overhead.
- **Suitable for near term implementation**: This API should be operable on a wide variety of WAP devices in a short time frame. It will not mandate functionality, programming language, or dependency upon subsystems that are not present in current specifications.
- **Extensible**: WAP is targeted to extend over a wide range of devices. The storage system must serve a wide range of WAP compatible devices, from simple to advanced systems, the prescribed set of storage services must have the capability to be implemented in a restricted runtime environment, recognising that the resources for code, stack and heap are limited.

4.2. Functionality

The Base Level persistence API is very simple in concept. It allows for retrieving data storage objects, but does not enforce structure – this is left to the application. Thus, whether a storage object that is being stored is a simple data type or a large, structured entity, the mechanism used to achieve persistence is the same. All persisted elements in the base level specification are treated as opaque resources.

5. Storage Elements

Persistent storage is a repository for statically stored data that resides locally on the client device. The persistent storage system provides three primitive storage elements. The **storage object** is used to store data and is the smallest storage element that may be independently managed as a unit by the storage system. A **directory** is used to group together storage objects and for organising other directories and storage objects into a hierarchical structure. The **storage medium** is the storage element that represents a physical storage device that is capable of storing persistent data.

The storage elements that are described in this storage system could be loosely mapped into conventional storage systems. The following diagram provides an example of how the storage elements of a typical file system and a typical relational database system correlate to the storage system elements as they are described in this base level persistent storage system. (Note that in the following diagram, the term "Disk Drive" represents one partition of a physical disk that has been partitioned into one or more logical drives.)

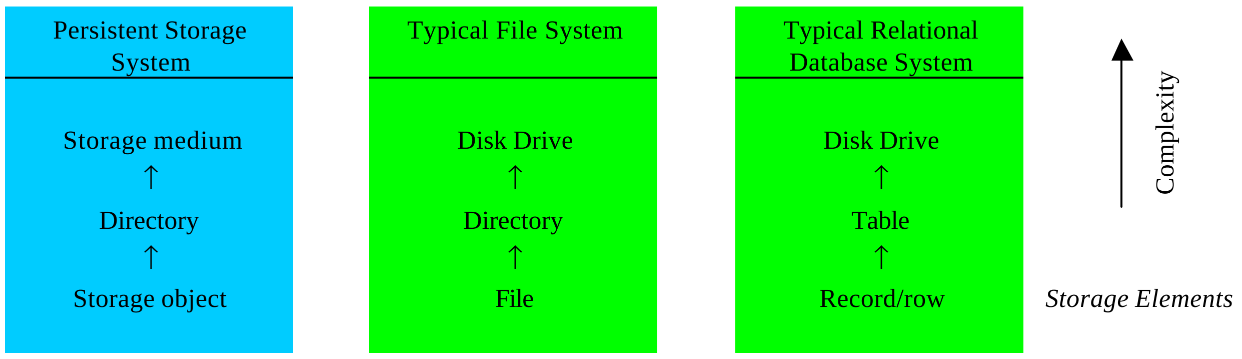

Figure 5-1. The Abstract Correlation of Storage System Elements to Conventional Storage Systems

5.1. Attributes

The characteristics of each storage system element are described using attributes. A storage element's attributes exist throughout the life of the storage element. The storage system provides a set of attributes that are used to distinguish the storage element's type, to describe the characteristics of the storage element and to characterise the contents of the data that is stored in the storage element.

Attributes will be designated as either mandatory or optional. The storage system will return an error code when an optional attribute is requested that is not supported by the particular media or implementation.

A sample of the storage system attributes includes: the format of the storage element's data, the date that the storage element was created and the date that the storage element was last modified.

5.2. Storage Medium

The storage medium element is used to represent a single instance of any device that is capable of storing data persistently, including both read/write devices and read only devices.

The persistent storage system MUST only provide access to storage media that the client device has made available to the persistent storage system. The names of all of the local storage media, that the client device has chosen to make available to the storage system, are recorded by the storage system. A WAE application can then discover the currently accessible client device's storage media by using the storage system's enumeration function.

The mapping of the storage system structure onto the storage medium (e.g. installation, initialisation and formatting) is outside the scope of the persistent storage system.

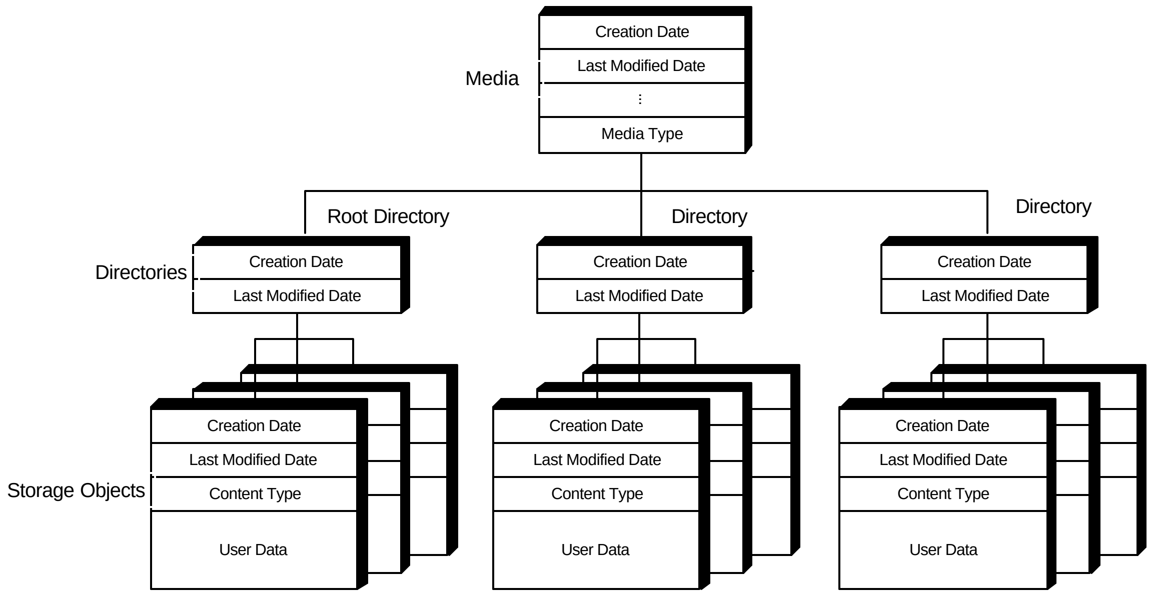

Figure 5-2. The Hierarchy of Primitive Storage Elements – Media, Directories and Storage Objects

5.2.1. Storage Medium Device Types

The persistent storage system uses a set of well-known names for storage medium that may be potentially used for persistent storage in mobile devices. The following table contains a list of device types that are currently recognised by the storage system as valid storage medium types. The names under the "Media Name" column are the names that are currently accepted by the storage system for the "stora geType" attribute that is used in describing storage medium attributes.

Table 5-1. Well-known Storage Medium Types

Media Type	Media Name	Capability	Lifetime
Battery powered RAM	RAM	R/W	Static
EEPROM	EEPROM	R/W	Static
EPROM/ROM	ROM	RO	Static
FLASH	FLASH	R/W	Static
Magnetic disk	DISK	R/W	Static
Memory Stick	FLASH	R/W	Static
ROM	ROM	RO	Static
Solid state disk	DISK	R/W	Static
SRAM	RAM	R/W	Static
SIM	SIM	R/W	Static
SmartCard	SC	R/W	Static

The base level persistent storage system does support the removal of a removable media device from an individual client device that is subsequently reinserted into the same client device. Under these circumstances, the removable media device is managed by the same storage system.

A removable media device that is removed from one client device and is subsequently modified in another system is considered a portable removable media device. The capability to map the storage system structure and naming scheme onto a portable removable device, such that the information stored onto the removable device by one client device that is expected to be subsequently recognised by another client device, is not supported by the base level persistent storage system.

5.3. Directories

Directories are used by WAE applications to logically group or collect together storage objects and provide WAE applications a means to logically separate their data from other WAE applications. A storage object MUST be a member of only one directory.

A directory is simply a list of storage objects and additional directories (sub-directories). The hierarchical directory structure is typically represented as branches of a tree. The directory tree diagrammed below depicts an example of this hierarchical structure.

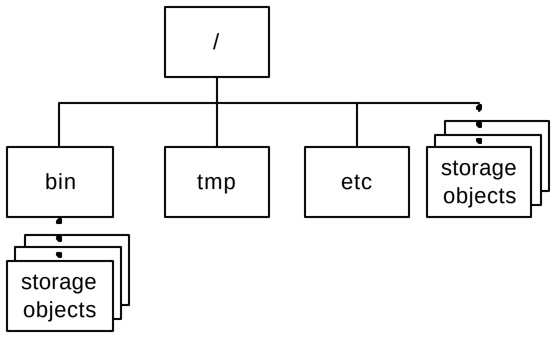

Figure 5-3. Directory Tree Structure

The main or top-level directory is referred to as the “root directory". Sub-directories represent the tree's branches and the storage objects represent the tree's leaves. Each directory may contain additional sub-directories and storage objects. The maximum number of branches a particular media device can support is specified by a storage media attribute.

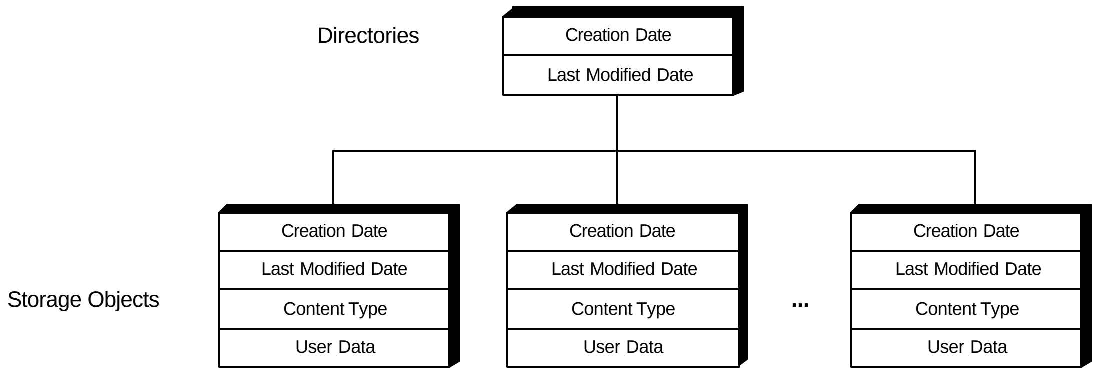

Figure 5-4. Directories are Used to Group Together Storage Objects

While the root directory is referenced by “/”, all of the other directories are referenced by names. A WAE application names the directory when the directory is created. Either the storage medium or the storage system MAY place a limit on the maximum number of directories.

A WAE application can discover the directories present on the storage medium by using the storage system’s enumeration function.

5.4. Storage Objects

A storage object is the storage system element that is used for storing WAE application data. The attributes of a storage object are used to describe the characteristics of the data that is stored in the storage object.

Each storage object has a local name that is specified by the application when the storage object is created. This local name typically carries some significant or recognisable meaning.

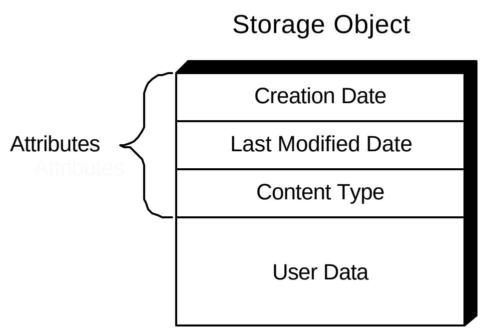

Figure 5-5. Characteristics of a Storage Object

6. Naming

Persistent storage elements are identified using the persistent storage URI syntax. This URI syntax is based on [RFC2396]. Applications MUST use the following URI syntax when accessing storage elements through the persistent storage system. Furthermore, applications MUST use the absolute path to identify storage elements; the storage system does not support relative naming.

```
wpsURI          = scheme ":" abs_path
```

In the persistent storage URI, the scheme always consists of “wps” to identify resources that are provided by the storage system.

```
scheme          = "wps"
```

The absolute path that identifies a storage element includes: the name of the media in which the storage element is located; a sequence of hierarchical directory names; and the name of the storage object. If no directory names are provided, the storage object resides in the root directory of the specified media.

```
abs_path        = "/" media ["/" | ("/" directory)* ["/" storage_object]]

media           = 1*pchar

directory       = 1*pchar

storage_object  = 1*pchar

pchar           = unreserved | escaped

unreserved      = alphanum | mark

mark            = "-" | "_" | "."

escaped         = "%" hex hex

hex             = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"

alphanum        = alpha | digit

alpha           = lowalpha | upalpha

lowalpha        = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
                  "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

upalpha         = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
                  "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

digit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

7. Management

7.1. Delete

The base level persistent storage system offers the *remove* function to delete individual storage objects and individual directories (excluding the root directory). The *remove* function MUST NOT be capable of deleting a storage medium.

When a directory is specified, *remove* MUST delete all of the storage objects and sub-directories that are contained in the directory, as well as the respective directory. Storage objects that are stored in the root directory MAY be deleted, but the root directory itself MUST NOT be deleted.

The *remove* function requires an explicit directory or storage object name and MUST NOT accept wildcards. It operates on a single storage element, except when deleting a directory, which then would also include the contents of the directory.

7.2. Transaction Semantics

The base level persistent storage system does not specify, nor require, any sophisticated transaction semantics. In particular, support for traditional transaction concepts such as concurrency, recovery and isolation are beyond the scope of this base level specification.

An implementation of the base level persistent storage system MUST ensure that when a storage object is updated, the specified update MUST entirely succeed. Otherwise, in the event when an update to a storage object fails, the storage system MUST restore the storage object to its original contents and return an appropriate error.

8. Security

The base level storage system does not offer any specific security features or mechanisms. Developers are warned that this specification is absent any protocols or mechanisms (such as library functions or function arguments) that could be used to provide security features such as identity, access controls and data integrity.

Security has been omitted from this specification because security mechanisms are dependent on a scheme that should be consistent across other application services and therefore cannot be defined within the scope of this specification.

When access to a storage element is denied due to any of the security mechanisms that are implemented by the underlying storage device, access to that storage element MUST be denied and an “Unauthorised” (401) error MUST be returned.

9. Error Handling

The persistent storage system follows the error handling scheme that is defined in the WMLScript language (see [WMLS] for more details):

- An invalid function argument MUST result in the return value of *invalid* with no side effects (unless explicitly stated otherwise).
- A function argument that cannot be converted to the required parameter type MUST result in the return value of *invalid* with no side effects.
- Storage system library functions MUST indicate function dependent errors by returning *invalid* and making the applicable status code that is specified in each function definition available to the getStatus() function.
- The getStatus() function is used to return a text string that contains the status code and a brief description of the status code.
- The status codes for each function are documented below as part of the function's specification.
- Function status codes MUST follow the status code conventions described in the HTTP1.1 RFC [RFC2616] where appropriate. When necessary, extended status codes are provided that adhere to the format of [RFC2616] and are specific to this library.
- Failures resulting when a store fills to capacity, MUST terminate the current storage operation, return *invalid*, and make the status code available through the getStatus() function.

10. Library Functions

10.1. Library

NAME:	Pstor
LIBRARY ID:	9
DESCRIPTION:	This library contains a basic set of functions for managing persistent storage primitives.

WAP client devices that provide support for the Pstor library and user agent profiling [UAPROF], MUST indicate support for the Pstor library through the device’s capability profile.

10.2. Miscellaneous Services

10.2.1. enum

Function: enum(*parentName, lastElement*)

Function ID: 0

Description: Individually enumerates each specified storage element type under the specified parent.

This function is used to discover the indicated storage element types that are children of the specified parent.

Parameters: *parentName* = String

lastElement = String

The *parentName* parameter is a URI string that forms a fully qualified absolute name string that identifies the parent of the storage elements that are to be enumerated. Possible values include:

- the storage medium and directory name for storage objects (e.g. wps:/FLASH/myObjects)
- the storage medium name for directories (e.g. wps:/BRAM)
- the empty string for storage medium (“”)

Pass an empty string to *lastElement* to request the first storage element’s name. On successive calls, pass the last element name that was retrieved in the previous call.

Return value: Type: String

Returns the name of the next storage element of the specified storage element type. Returns an empty string when the storage element list is exhausted.

If no storage elements are present, then this function returns an empty string as if it reached the end of the storage element list.

invalid is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes: 200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

Notes:

1. The name returned by *enum* can be used to query attributes of the storage element.
2. The results of the enumeration function will not be deterministic when the series is interrupted by calls to other persistent storage operations. (e.g. Calls to *create* or *remove* may disrupt the process of enumerating all of the storage objects in a directory, resulting in the possibility that some of the storage elements may be missed.)

Example:

```
// iterate through the list of available storage media
var name = Pstor.enum(“”, “”);
             while (!isNull(name) && isvalid name)
             {
                          ...          // do something with the name
                          name = Pstor.enum(“”, name);
             }
```

10.2.2. getStatus

Function:	getStatus()
Function ID:	1
Description:	Retrieves the status code and error message, as defined in [RFC2616], from the last function called (excluding this function).
Parameters:	None
Return value:	Type: String Error messages are formatted as specified in HTTP 1.1 [RFC2616].
Status Codes:	None; this function MUST NOT fail.

Notes:

1. If no library function has been called prior to calling getStatus(), then the text string that is returned MUST be "200 OK".
2. Once any other library function has been called, getStatus() MUST return the status code that was generated by the last library function called.
3. Successive, uninterrupted calls to getStatus() MUST return the same result because getStatus() does not modify the status code.

Example:

```
// Intentionally create an error and then retrieve the resulting status
code.
var val = enum(invalid, invalid);

var error = getStatus();
```
// error is now "450 Invalid Input"

10.3. Storage Primitive Manipulation

10.3.1. create

Function: create(*name, objType)*

Function ID: 2

Description: Creates a named storage element in the persistent store.

Parameters: *name* = String

objType = String

The *name* parameter is a URI string that forms a fully qualified absolute name to identify the new storage element.

objType is defined as one of the following:

- "Directory" – creates a directory
- "StorageObject" – creates a storage object

Return value: Type: String

Returns the status of the operation. *invalid* is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes: 200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

550 Media Full

Notes:

1. When a new storage object is created, the default value for ContentType is "text/plain".

Example:

```
// Create a storage object named: "wps:/SRAM/personal/entry1"

status = Pstor.create("wps:/SRAM/personal/entry1", "StorageObject");
```

10.3.2. remove

Function: remove(*name*)

Function ID: 3

Description: Removes the named directory or storage object from the persistent store.

Parameters: *name* = String

The *name* parameter is a URI string that forms a fully qualified absolute name to identify the storage element that is to be permanently removed from the persistent store.

Return value: Type: Boolean

Upon success the value true is returned. false is never returned by this function because all errors force the return of *invalid*.

invalid is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes: 200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

Notes:

1. This function only removes the single storage element that is named in the *name* parameter.
2. When a directory is removed, all of the storage objects and sub-directories that are contained in that directory will also be removed.
3. The contents of the root directory may be deleted but it is not possible to delete the root directory.
4. A “Bad Request” exception code is returned when the storage element that is named in the *name* parameter references a storage medium.

Example:

```
//remove the storage object named “wps:/EEPROM/ab/personal.obj”

result = Pstor.remove(“wps:/EEPROM/ab/personal.obj”);
```

10.4. Attribute Manipulation

The characteristics of each storage element are described using attributes.

Attributes MUST adhere to the following rules:

1. The basic calendar format “CCYYMMDDThhmmssZ”, which is specified in [ISO8601], MUST be used when specifying date attributes.
2. The names of all attributes MUST be case insensitive.

Table 10-1 contains the attributes that are used to describe storage media.

Table 10-1. Storage System Defined Storage Medium Attributes

Attribute Name	Description	Format	Modification Property	Mandatory/Optional
AvailableSize	Identifies the size in bytes of the available storage space currently remaining on the storage medium.	Integer	Read Only	Optional
CreationDate	The date that the storage medium was created (this value is the date that the root directory was created).	String	Read Only	Optional
LastModifiedDate	The date that the storage medium was last modified.	String	Read Only	Optional
MaximumDirectoryLevels	Specifies the maximum number of directory levels that the storage media is capable of supporting.	Integer	Read Only	Mandatory
StorageType	Identifies the type of storage medium of the device. This value comes from WAP’s list of well-known storage medium types.	String	Read Only	Mandatory
SupportsCreateDirectory	Indicates whether the storage medium supports the creation of new directories.	Boolean	Read Only	Mandatory
WriteLocked	Identifies whether writes to the device are permitted or prohibited. The operation of locking a storage medium is performed outside the storage system and no mechanism is provided to allow WAE applications to lock or unlock a storage medium.	Boolean	Read Only	Mandatory

Table 10-2 contains the attributes that are used to describe directories.

Table 10-2. Storage System Defined Directory Attributes

Attribute Name	Description	Format	Modification Property	Mandatory/ Optional
CreationDate	The date that the directory was created.	String	Read Only	Optional
LastModifiedDate	The date that the directory was last modified.	String	Read Only	Optional

Table 10-3 contains the attributes that are used to describe a storage object.

Table 10-3. Storage System Defined Storage Object Attributes

Attribute Name	Description	Format	Modification Property	Mandatory/Optional
ContentType	Uses the MIME format to describe the general type and specific format of the data contained in the storage object [RFC2046]. **Note:** When a new storage object is created, the default value for ContentType is “text/plain”.	String	Read/Write	Mandatory
CreationDate	The date that the storage object was created.	String	Read Only	Optional
LastModifiedDate	The date that the storage object was last modified.	String	Read Only	Optional

10.4.1. getAttribute

Function:	getAttribute(*name*, *attributeName*)
Function ID:	4
Description:	Gets the named attribute from the specified storage element.

Parameters:

name = String

attributeName = String

The *name* parameter is a URI string that forms a fully qualified absolute name that identifies the target storage element.

The *attributeName* parameter is the attribute name that is specified in:

- Table 10-1 for storage media
- Table 10-2 for directories
- Table 10-3 for storage objects

Return value:

Type: Any

Returns the value of the attribute in the form that it was stored.

invalid is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes:

200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

451 Input Not Supported

Notes:

1. Attribute names are case insensitive.
2. An error results when an optional attribute is requested that is not supported by the specified media or particular implementation.

Example:

```
// get the date that the storage object named
//   “wps:/BRAM/ab/personal.obj” was last modified

var lastModifiedDate = Pstor.getAttribute(“wps:/BRAM/ab/personal.obj”,
                                         “LastModifiedDate”);
```

10.4.2. setAttribute

Function:	setAttribute(*name, attributeName, value*)
Function ID:	5
Description:	Sets the named attribute for the specified storage element.

Parameters:

name = String

attributeName = String

value = String

The *name* parameter is a URI string that forms a fully qualified absolute name that identifies the target storage element.

The *attributeName* parameter is the attribute name that is specified in:

- Table 10-1 for storage media
- Table 10-2 for directories
- Table 10-3 for storage objects

The *value* parameter is the actual value to store into the attribute.

Return value: Type: Boolean

Upon success the value true is returned. false is never returned by this function because all errors force the return of *invalid*.

invalid is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes:

200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

451 Input Not Supported

Notes:

1. Attribute names are case insensitive.
2. An error results when an optional attribute is requested that is not supported by the specified media or particular implementation

Example:

```
// set the content type of the storage object named
//   “wps:/BRAM/ab/my.obj” to “image/jpeg”
result = Pstor.setAttribute(“wps:/BRAM/ab/my.obj”, “ContentType”,
                                                    “image/jpeg”);
```

10.5.2. getData

Function: getData(*name, offset, size*)

Function ID: 7

Description: Retrieves a portion or all of the data that is stored in the specified storage object.

Parameters: *name* = String

offset = Integer

size = Integer

The *name* parameter is a URI string that forms a fully qualified absolute name that identifies the target storage element.

offset determines the position within the attribute to begin retrieving information from, in bytes. *offset* is zero-based (the first data byte is numbered 0, the second is numbered 1, etc.)

If offset is beyond the size of the stored data, the result is a zero length string.

size is the maximum number of bytes to retrieve:

Use 0 to return a zero-length string.

Use -1 to return all of the data.

Return value: Type: String

A string containing the data that is stored in the specified storage object.

invalid is returned on error, and a call to the getStatus() function will return the appropriate status code and message.

Status Codes: 200 Ok

400 Bad Request

401 Unauthorised

404 Not Found

450 Invalid Input

Notes: None

Example:

```
// Read 3 bytes from the beginning of the data that is stored in the
// storage object named “wps:/EEPROM/ab/personal.obj” that contains the
// string “214-555-1212”

areaCode = Pstor.getData(“wps:/EEPROM/ab/personal.obj”, 0, 3);
  // the variable areaCode now contains the string “214”
```

10.5.3. setData

Function:	setData(*name, value, offset*)
Function ID:	8
Description:	Sets a portion or all of the data that is stored in the storage object.
Parameters:	*name* = String *value* = String *offset* = Integer The *name* parameter is a URI string that forms a fully qualified absolute name that identifies the target storage element. The *value* parameter is the actual data to store into the storage object. The *offset* parameter determines the position within the attribute to start modifying information, in bytes. *offset* is zero-based (the first data byte is numbered 0, the second is numbered 1, etc.)
Return value:	Type: Boolean Upon success the value true is returned. false is never returned by this function because all errors force the return of *invalid*. *invalid* is returned on error, and a call to the getStatus() function will return the appropriate status code and message.
Status Codes:	200 Ok 400 Bad Request 401 Unauthorised 404 Not Found 450 Invalid Input 550 Media Full
Notes:	1. If *offset* specifies a location that is beyond the size of the storage object, an error occurs and *invalid* is returned. 2. The setData function does not buffer writes. All changes are made to the storage object before the function returns.

Example:

```
//set the contents of the storage object named
//  “wps:/BRAM/ab/personal.obj” to “John Jones”

result = Pstor.setData(“wps:/BRAM/ab/personal.obj”, “John Jones”, 0);
```

10.5.4. setDataSize

Function:	setDataSize(*name, size*)
Function ID:	9
Description:	Truncates or extends the total size (measured in bytes) of the data space of the specified storage object.
Parameters:	*name* = String *size* = Integer The *name* parameter is a URI string that forms a fully qualified absolute name that identifies the target storage element. The *size* parameter specifies the size in bytes of data stored.
Return value:	Type: Boolean Upon success the value true is returned. false is never returned by this function because all errors force the return of *invalid*. *invalid* is returned on error, and a call to the getStatus() function will return the appropriate status code and message.
Status Codes:	200 Ok 400 Bad Request 401 Unauthorised 404 Not Found 450 Invalid Input 550 Media Full
Notes:	1. When the size of the storage object's data is extended, the content of the added data space is undefined.

Example:

```
// Reserve 50 bytes of storage space in a new storage object named
//  “wps:/EEPROM/ab/entry”

result = Pstor.setDataSize(“wps:/EEPROM/ab/entry”, 50); // the data size
                                                       // is now 50 bytes
```

Appendix A. Static Conformance Requirements (Normative)

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

WMLScript Compiler and Encoders

Item	Function	Reference	Status	Requirement
PSTOR-S-001	Support for enum() function	10.2.1 enum	M	
PSTOR-S-002	Support for getStatus() function	10.2.2 getStatus	M	
PSTOR-S-003	Support for create() function	10.3.1 create	M	
PSTOR-S-004	Support for remove() function	10.3.2 remove	M	
PSTOR-S-005	Support for getAttribute() function	10.4.1 getAttribute	M	
PSTOR-S-006	Support for setAttribute() function	10.4.2 setAttribute	M	
PSTOR-S-007	Support for getDataSize() function	10.5.1 getDataSize	M	
PSTOR-S-008	Support for getData() function	10.5.2 getData	M	
PSTOR-S-009	Support for setData() function	10.5.3 setData	M	
PSTOR-S-010	Support for setDataSize() function	10.5.4 setDataSize	M	
PSTOR-S-011	Support for storage media	5.2 Storage Medium	M	
PSTOR-S-012	Support for directories	5.3 Directories	M	
PSTOR-S-013	Support for storage objects	5.4 Storage Objects	M	
PSTOR-S-014	Use persistent storage URI syntax	6 Naming	M	
PSTOR-S-015	Use absolute path	6 Naming	M	
PSTOR-S-016	Support for delete	7.1 Delete	M	
PSTOR-S-017	Support for transaction semantics	7.2 Transaction Semantics	M	
PSTOR-S-018	Support for security	8 Security	M	
PSTOR-S-019	Support for error handling	9 Error Handling	M	

Item	Function	Reference	Status	Requirement
	handling			
PSTOR-S-020	Support for attribute manipulation	10.4 Attribute Manipulation	M	

WMLScript Bytecode Interpreter

Item	Function	Reference	Status	Requirement
PSTOR-C-001	Support for enum() function	10.2.1 enum	M	
PSTOR-C-002	Support for getStatus() function	10.2.2 getStatus	M	
PSTOR-C-003	Support for create() function	10.3.1 create	M	
PSTOR-C-004	Support for remove() function	10.3.2 remove	M	
PSTOR-C-005	Support for getAttribute() function	10.4.1 getAttribute	M	
PSTOR-C-006	Support for setAttribute() function	10.4.2 setAttribute	M	
PSTOR-C-007	Support for getDataSize() function	10.5.1 getDataSize	M	
PSTOR-C-008	Support for getData() function	10.5.2 getData	M	
PSTOR-C-009	Support for setData() function	10.5.3 setData	M	
PSTOR-C-010	Support for setDataSize() function	10.5.4 setDataSize	M	
PSTOR-C-011	Support for storage media	5.2 Storage Medium	M	
PSTOR-C-012	Support for directories	5.3 Directories	M	
PSTOR-C-013	Support for storage objects	5.4 Storage Objects	M	
PSTOR-C-014	Use persistent storage URI syntax	6 Naming	M	
PSTOR-C-015	Use absolute path	6 Naming	M	
PSTOR-C-016	Support for delete	7.1 Delete	M	
PSTOR-C-017	Support for transaction semantics	7.2 Transaction Semantics	M	
PSTOR-C-018	Support for security	8 Security	M	
PSTOR-C-019	Support for error handling	9 Error Handling	M	
PSTOR-C-020	Support for attribute	10.4 Attribute	M	

Item	Function	Reference	Status	Requirement
	manipulation	Manipulation		

Appendix B. Change History (Informative)

Type of Change	Date	Section	Description
Class 0	16-May-2001		The initial version of this document.

Appendix C. Implementation Notes (Informative)

The following implementation notes are intended to assist the implementer in the design of this library.

1. Access control security may be implemented as a layer above the base level persistent store.
2. It is strongly recommended that special considerations be given in the design of the storage system to optimise the use of storage resident on the client device.

Appendix D. Application Developer Notes (Informative)

The following notes are intended to assist developers in writing applications that use the persistent storage system.

1. It is suggested that applications use distinctive names for directories to avoid name collisions.

2. It is suggested that applications check the result of each library function call for the return value *invalid*. When *invalid* is returned, the function getStatus() should be used to determine the cause of the error.

Appendix E. Code Example (Informative)

An application that displays a persistently stored list of user specified stocks:

```
// Assume that there are stocks in the store already.
// Functionality would need to be added to allow the user to modify the
//  list of stocks.

//
// Get the stock names and prices to display to the user.
//
var stock = Pstor.enum( "wps:/FLASH/stocks", "" );
var price = Pstor.getData( "wps:/FLASH/stocks/" + stock,
                           "0", "-1" );
var prices = stock + ":" + price;
var stocks = stock;
var done = false;       // have we gotten all the stocks?
var count = 1;          // count of stocks retrieved

while ( !done && isvalid stock)
{
    stock = Pstor.enum( "wps:/FLASH/stocks", stock );
    if ( isvalid stock && !String.isEmpty( stock ) )
    {
        var price = Pstor.getData( "wps:/FLASH/stocks/" + stock,
                                   "0", "-1" );
        prices += "," + stock + ":" + price;
        stocks += "," + stock;
        count++;
    }
    else
    {
        done = true;
    }
}
```

```
// See if the user wants to get updated stock prices.
//
update = Dialogs.confirm( prices, "Get Update", "Cancel" );
if ( update == true )
{
    // Retrieve the stock prices from the stock service
    // and display the prices to the user, e.g.
    // http://quotes.com?stocks=ACME,LJRT,XSRD
    //
    stocks = URL.loadString( "http://quotes.com?stocks=" + stocks,
                             "text/plain" );
    Dialogs.alert( stocks );

    // Put the updated stock prices in the persistent storage
    //
    for ( i = 0; i < count; i++ )
    {
        // Format of the quotes provided by quote.com is
        // <name>:<price>,<name>:<price>, ...
        //
        var stock_price = String.elementAt( stocks, i, "," );
        var stock = String.elementAt( stock_price, 0, ":" );
        var price = String.elementAt( stock_price, 1, ":" );
        var length = String.length( price );

        // Update the storage object data and trim to
        // remove any leftover stock price data.
        //
        Pstor.setData( "wps:/FLASH/stocks/" + stock,
                       price, "0" );
        Pstor.setDataSize( "wps:/FLASH/stocks/" + stock,
                           length );
    }
}
```