Morph Protocol

# Introduction

The Morph Protocol was developed to provide desired functionality not provided by [SOAP](http://www.w3.org/TR/soap/). However, although SOAP has the advantage of being completely language independent, Morph is intended specifically for object oriented languages. Both protocols have their own uses.

Some of the functionality that Morph offers:

* Efficiency:
  + Morph is a carefully designed to be a frugal binary protocol.
  + Optimises resource use.
  + Designed for mobile devices.
* Remote procedure calling of methods and properties, including:
  + Session handling.
  + Each service can offer multiple servlets (server objects).
  + Two-way RMI. (Both ends of a conversation can act as both client and server.)
  + Passing object references on to other devices.
* Parameters may contain:
  + basic values, objects and/or arrays by value
  + objects and/or arrays by reference
  + streams (not yet developed)
* Networking features:
  + Addressing works seamlessly for both IPv4 and IPv6.
  + Greater security by enabling encryption beyond sockets (not yet developed).

## History

## Documentation

Morph documentation consists of two documents:

* [*Morph Protocol.docx*](Morph%20Protocol.docx) Describes Morph. (This document.)
* [*Morph Protocol.xlsx*](Morph%20Protocol.xlsx) Defines Morph.

## How to read *Morph Protocol.xlsx*

Although *Morph Protocol.xlsx* is fairly easy to read, it might help to have an introduction.

Let’s take a look at an example.

Lookup the spreadsheet page called “ValueType”.

* The columns named 0-7 refer to the bit locations of the byte that is being defined. So, the flag called HasValueName is located on bit 0.
* Read downwards, from the top of each section.
* Looking at the category called Basic, we see that there can be four flags: HasValueName, IsNull, HasTypeName and IsReference.
* A value MUST exist iff a flag is set that is associated with that value. In this example, if HasValueName then the byte must be followed by a string value called ValueName. The next flag called IsNull has no value associated with it, so there is no value to follow.
* Flags only apply if their conditions are met. The next flag, called HasTypeName, has a condition “IsNull=False” and so it only applies if IsNull was false. So, if IsNull=False then we can read the value of HasTypeName. If HasTypeName is true, then we MUST read/write the value TypeName.
* The conditional reading means that the same bit can be used for different flags, based on the values of prior flags. So, in this example, if IsNull=False and IsReference=True then bit 4 represents the flag IsServlet. But if IsNull=False and IsReference=False then bit 4 represents the flag IsStruct.

So a ValueType byte with the value 0x09 (0000 1001) MUST be followed by a string (ValueName) and then by a 4 byte integer (ReferenceID), in that order. (Don’t worry about the size of the string for this example.)

# Morph concept

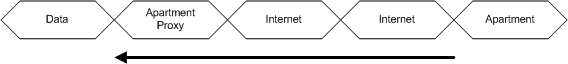
The Morph protocol works in quite a different way from most other protocols, therefore this section is very important to understand. Please make sure you understand the general idea of the Morph concept before doing any other work.

## Message is a chain of links

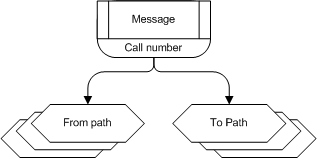
Messages can and will change structure as they travel from source to destination. This is the namesake of “Morph”.

Each message consists of a header called a “Message link”. The message link is followed by a chain of links that specify different nodes (waypoints) along the trip. Links will be added, removed or replaced as needed.





Actually, the chain is split into a From path (the nodes that the message has already visited) and the To path (the nodes that the message intends to visit). The To path is of course necessary, but the From path is often optional. This design is intended to enable greater efficiency.



(Originally, the protocol was almost named the “Chain protocol” after the chain like structure of messages, but the name “Chain” made it feel like a bind whereas the name “Morph” sounds more flexible and adaptable.)

## Morph topology

What are these Morph “nodes”? Well, there are different types. Each node represents an object or a function in the Morph topology.

This figure describes the topology for servlets.



Each message contains the path to its destination.

# General rules

## Encoding

Morph MUST be encoded in UTF-8.

There is no padding. All data is packed.

Every Link type byte contains a flag that specifies whether it is encoded as MSB or LSB.

## Arrays

All arrays consist of a counter, followed by the data. The counter specifies the number of array elements.

Where the size of the counter is not specified, it is 4 bytes long.

Where the size of the counter is specified, it is specified using a bit pair. The size of the array counter, in bytes, is = 2bit pair. (ex. For a bit pair value of b11 = 3, the array counter is 23 bytes = 8 bytes long.)

**Note:** The ONLY exception are FromPathSize and the ToPathSize in Message links. They specify the length of each path in bytes (not links).

## Strings

Strings are arrays of Unicode characters.

There is no ASCII in Morph, except for the connection string.

# Transport layer

## Over networks

Morph is designed to run over any transport layer that supports streaming.

Over the internet, the ideal protocols are IPv6 and IPv4, in that order. Ideally all communication should go over port 0xE000 for plain sockets, or 0xE055 for secure sockets. (These are the Morph equivalents of port 80 and 443 for web services.)

A Bluetooth transport has not yet been designed. Any volunteers?

## Socket connections

### Handshake

When a socket connection is established, BOTH ends MUST send the following bytes as the very first bytes. When a socket connection is established, BOTH ends MUST verify that the first bytes received are the following bytes, and BOTH ends MUST check to see that they are compatible with the version numbers received from the other end.

|  |  |  |
| --- | --- | --- |
| Value | Size in bytes | Description |
| "Morph"#0 | 6 | ASCII characters: 0x4D, 0x6F, 0x72, 0x70, 0x68, 0x00 |
| Major version | 1 | Clearly incompatible versions must have different Major version numbers. First version is 1. |
| Minor version | 1 | Possibly compatible differences must have different Minor version numbers. First version is 1. |

After that, all data sent is Morph messages.

### Socket errors

If, for any reason, the socket appears corrupted, then close the connection.

If another connection to the same device can be re-established, then the local device SHOULD try to send the waiting messages over the new socket connection.

If another connection cannot be re-established, then the local device MAY try to find another route over some other device.

### Closing sockets

There are no niceties for closing socket connections. Just close ‘em.

# Morph Links

## Message link

Every message consists of a Message link followed by non-Message links. (Message links are not nested.)

### Replying to messages

If you send a message that is intended as a request that you want a response to (ex. calling a remote procedure) then it will need to contain enough information for a reply to be constructed that can find its way back to the caller.

For a reply to find its way back to the calling thread on the calling machine, the information that will need to be included in the request message is:

|  |  |
| --- | --- |
| **Value** | **Used for** |
| CallNumber | Required for matching the response to the waiting thread on the calling machine. |
| FromPath | Required so that the reply message knows how to find its way back.  **Note:** If the call is sent to a sessioned apartment, then this will not be necessary. |