Morph Protocol

# 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.

If you are new to the Morph protocol, then please start with reading the section .

Contents

[Documentation 1](#_Toc371347281)

[Introduction 4](#_Toc371347282)

[History 4](#_Toc371347283)

[Morph concept 5](#_Toc371347284)

[Message is a chain of links 5](#_Toc371347285)

[Introduction to Morph topology 5](#_Toc371347286)

[Servlets, apartments and services 6](#_Toc371347287)

[Servlets 6](#_Toc371347288)

[Apartments 6](#_Toc371347289)

[Services 7](#_Toc371347290)

[Methods and properties 7](#_Toc371347291)

[Parameters 7](#_Toc371347292)

[Extended functionality 7](#_Toc371347293)

[Sequencing 8](#_Toc371347294)

[Streaming 8](#_Toc371347295)

[Application level routing 8](#_Toc371347296)

[Security 8](#_Toc371347297)

[Morph Daemon 10](#_Toc371347298)

[How to read Morph Protocol.xlsx 11](#_Toc371347299)

[General rules 11](#_Toc371347300)

[Morph data types 11](#_Toc371347301)

[Sign bit 11](#_Toc371347302)

[Numerics 11](#_Toc371347303)

[Strings 11](#_Toc371347304)

[Link types 12](#_Toc371347305)

[Example: Reading Message Link 12](#_Toc371347306)

[Example: Reading Service Link 12](#_Toc371347307)

[Example: Reading Internet Link 13](#_Toc371347308)

[ValueType 13](#_Toc371347309)

[Simple type 14](#_Toc371347310)

[General rules 14](#_Toc371347311)

[Encoding 14](#_Toc371347312)

[Arrays 14](#_Toc371347313)

[Strings 15](#_Toc371347314)

[Transport layer 16](#_Toc371347315)

[Over networks 16](#_Toc371347316)

[Socket connections 16](#_Toc371347317)

[Handshake 16](#_Toc371347318)

[Socket errors 16](#_Toc371347319)

[Closing sockets 16](#_Toc371347320)

[Morph Links 17](#_Toc371347321)

[End link 17](#_Toc371347322)

[Message link 17](#_Toc371347323)

[Flags 17](#_Toc371347324)

[Values 17](#_Toc371347325)

[Replying to messages 17](#_Toc371347326)

[Data link 18](#_Toc371347327)

[Structure 18](#_Toc371347328)

[Flags 18](#_Toc371347329)

[Values 18](#_Toc371347330)

[Service link 19](#_Toc371347331)

[Values 19](#_Toc371347332)

[Servlet link 19](#_Toc371347333)

[Values 19](#_Toc371347334)

[Member link 19](#_Toc371347335)

[Flags 20](#_Toc371347336)

[Values 20](#_Toc371347337)

[Process link 20](#_Toc371347338)

[Values 20](#_Toc371347339)

[Internet 20](#_Toc371347340)

[Flags 20](#_Toc371347341)

[Values 21](#_Toc371347342)

[Sequence link 21](#_Toc371347343)

[Flags 21](#_Toc371347344)

[Values 21](#_Toc371347345)

[Encoding link 21](#_Toc371347346)

[Stream link 21](#_Toc371347347)

[Morph Topology 22](#_Toc371347348)

# 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.

Some of the functionality that Morph offers:

* Efficiency:
  + Morph is 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.
  + Multiple servlets (remotely callable objects).
  + Servlet references can be sent on to other application/devices.
  + Two-way RMI. (Both ends of a conversation can call each other’s servlets.)
* Parameters may contain:
  + basic values, objects and/or arrays sent by value
  + objects and/or arrays sent by reference
  + streams (not yet developed)
* Networking features:
  + Data flows seamlessly across IPv4 and IPv6.
  + Greater security by enabling encryption beyond sockets (not yet developed).
  + Application level routing. (Not to be confused with network routing.)

## History

When developing a mobile phone application I felt it was overkill for a mobile phone to run a web server in order to host a web service for doing specialised synching, especially if it was over Bluetooth. Something had to be done, and I so I invented the Morph protocol.

Originally, I almost named it 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. So the name Morph is much better fung shui but also refers to the main strength of the Morph protocol.

Peter Thönell

# Morph concept

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

## Message is a chain of links

Every Morph message consists of a chain of links. The links either specify different nodes (waypoints) along the journey or may contain the payload.

Links will be added, removed, replaced or moved as needed at each step along the way. Therefore all messages will change structure as they travel from source to destination. This is one namesake of the “Morph” Protocol.

The first link in the chain, acting as a header, is a Message Link. It is followed by a series of other links that make up a “To Path” and often also a “From Path”.

The “To Path” is the route that the message has yet to travel to reach its destination and usually includes a payload.

The “From Path” is the route that the message has already travelled from its source.

Figure 1. Message structure



The To Path is of course necessary, but the From Path is often optional. This design is intended to enable greater efficiency in a couple of ways. Firstly, if a message contains no From Path, then as the message reaches each node along its route, the message shrinks in size by one link at each node. Secondly, with this design a message relay could start forwarding a message on to the next IP socket after only the first few bytes have been received.

## Introduction to Morph topology

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

Figures 2 and 3 give an idea of what a remote procedure call can look like. The blue parts represent Morph topology and the green parts represent Morph messages.

A Message Link contains the path and content of the message. The path and content of the message consists of links that refer to different nodes along the journey from source to destination. The Data Links contain parameter data or return data for the function call.

At first glance this system may look superfluous if the goal is only remote method calling. But remember that this is just the basic concept.

Figure 2. Remote method call



Figure 3. Remote method reply



## Servlets, apartments and services

### Servlets

A “servlet” is a Morph handle to an application business object. This way, business objects can be oblivious to Morph, but Morph can call their methods and properties.

### Apartments

An “apartment” is a collection of servlets. It can be thought of as a logical memory space.

There are two kinds of apartments:

* A “Sessioned Apartment” is dedicated to one single apartment proxy. This is how Morph does sessions.
* A “Shared Apartment” can be accessed by more than one client.

A server application can contain multiple shared apartments. It’s quite possible for a server to share an apartment with a select few of its clients but not to others, analogous to a chat room scenario.

### Services

In Morph, a service is nothing more than a named session factory. A service may create a new apartment (shared or sessioned) every time it is accessed, or it may return the same shared apartment every time.

When a client establishes contact with a Morph service, the first message will contain a Service Link. The Service Link contains the name of the desired service so that the message can be directed to the correct service. The service then produces an apartment. From that point onwards the service, having served its purpose, is no longer used and the client communicates only with the apartment.

## Methods and properties

Morph distinguishes between methods, property getters and property setters, and enables calls to all of them. Morph can also handle array properties.

Because servlets are handles to business objects, servlets are responsible for invoking methods and properties.

## Parameters

All payloads, including parameters, return values, property values, stream data, etc. are sent using the Data Link. This is typically the last link in the To Path.

Parameters can be:

* Sent by value, in which case they are encoded into the Data Link.
* Sent by reference, in which case a path to a servlet is passed so that the receiving end can build a complete path to the servlet and call it.
* Stream sent by reference. At the time of writing this is not completed, but the idea is to pass one-way stream references as parameters.
* Any combination of the above, including structs and/or arrays.

## Extended functionality

Morph is a very flexible protocol, and may grow in the future to accommodate more functionality (another namesake of the “Morph” Protocol). This section briefly introduces some extra features.

### Sequencing

A Sequence Node can enforce that messages are passed on in the same order that they were sent. There are three levels of sequencing:

* No sequencing. Due to the deterministic behaviour of computers, messages will generally arrive in the same order they were sent, but there is no guarantee for that.
* Lossy sequencing. If an older message arrives later than a younger message, then the older message is dropped.
* Lossless sequencing. If a younger message arrives before an older message, then it will wait until all older messages have gone though, and Link requests will be sent asking for the older messages to be resent.

### Streaming

Much the same way that Morph can send servlet references as parameters, Morph will also be able to send stream references. The streams will be one way streams, but should be multiplexable.

### Application level routing

This powerful feature is a consequence of the way Morph works, rather than due to a link type.

The idea is that applications can meddle with the chain of links that comprises a message. By doing so, applications can control how messages are re-routed.

Application level routing is currently used by the Morph Daemon to route messages between applications and to other devices, but there are other situations where that functionality can be useful.

### Security

With most protocols, messages are sent unchanged over a single socket connection. So for security it makes sense to send messages over encrypted sockets using [TLS (or SSL)](http://en.wikipedia.org/wiki/Transport_Layer_Security).

With Morph, messages are expected to travel over multiple socket connections, through nodes that could be 3rd party devices. That eliminates any guarantee of encrypted sockets over the whole journey and leaves ample opportunity for “man in the middle” attacks.

That is why security must be a part of the Morph Protocol, and cannot be left to secure sockets.

The TLS system is well proven, as long as keys and hashing algorithms are kept up to date. Using Morph’s modular culture of design, Morph security ought to allow for changing security measures, allowing Morph to keep up with the times.

Also, due to the way Morph topology works, messages might be encrypted multiple times over. For example, say two applications are communication over the internet using Morph. The applications may encrypt their messages, plus their Morph Daemons may enforce encryption to any data sent through the internet, plus relays along the way may wish to encrypt all they send, etc. This may seem overkill, but it is just a consequence of the modularity and playing it safe at different points along the way.

That said, there is no harm in using secure socket connections in addition to Morph security.

However, because network security is not my forté, I would like to leave the detailed design of Morph security to experts in the field so that there is less risk of holes.

# Morph Daemon

Multi-process OS’s can run a Morph Daemon. This is the preferred way to run Morph as it allows for a single Morph listener to listen for incoming connections, from both remote and local applications, and then routs messages between applications. Single process OS’s will make this difficult.

A Morph Daemon serves a similar purpose to a web server application in that it listens for incoming connections on a standardised port and directs incoming messages to the appropriate application.

The difference between a Morph Daemon and a web server application is that a Morph Daemon does not host Morph services (aside from its own management services). Rather, applications connect and register with their local Morph Daemon which then acts as a hub, routing messages between applications and remote devices. All Morph communication should occur only via the Morph Daemon (if possible on that OS). The Morph Daemon uses application level routing to redirect messages toward their destinations.

For example, a local application may connect to its local Morph Daemon and register a Morph service. From then on, both local and remote applications wanting to connect to that service will connect to the Morph Daemon, which redirects messages to that local application hosting the service.

A Morph Daemon has several advantages over a web server:

* When applications die, whether gracefully or not, the Morph Daemon can detect that and tidy up all its resources relating to those applications. This allows other applications to be unaffected by application crashes and enables the Morph Daemon to keep the system clean and in a stable state.
* Applications can spawn child processes and have full two-way OO conversations between the different processes using multiple objects and callbacks.
* A Morph Daemon could enforce a certain level of encryption for all messages to and from the device. This kind of behaviour could be controlled centrally by a system administrator.
* Efficiency. Sockets are reduced to a minimum, which may be beneficial to small systems. Also, the Morph Daemon can have a small footprint. At the time of writing, the full proof of concept [.Net Morph implementation written in C#](http://morphcsharp.codeplex.com/) comes to grand total of 159 KB, including a Morph Daemon for Windows, Daemon manager UI and all the Morph DLL’s.

Default Morph sockets:

* Plain socket port: $E000
* Secure socket port: $E055

# How to read Morph Protocol.xlsx

Although *Morph Protocol.xlsx* is fairly easy to read once you know how to read it, the “how to read” is not intuitive.

## General rules

Because the protocol is modular, tables often contain multiple sections. For example; the Link Type page which contains 16 link types (of which some are not defined). Each section is independent of the others.

1. Read each section in a top-down order, evaluating each row in turn.
2. Iff a bit is set, then its corresponding flag (ex. HasFromPath) name is set to true.
3. If the flag has two names separated with a “/” (ex. IsMethod/IsProperty) then the first flag name is set when the bit is 0 and the latter is set when the bit is 1.
4. If a flag is associated with a value, then that value will only be present iff the flag is set.
5. If a row contains a condition, then the whole row is ignored when the condition evaluates to false.

## Morph data types

These are data types inherent to the Morph protocol.

### Sign bit

When dealing with integers as counters (regardless of size) we ignore the most significant bit in case some platforms insist on using signed integers.

### Numerics

Most numerics in Morph are 32 bit unsigned integers.

64 bit values are overkill for a protocol and 16 bit integers may, in some cases, not be sufficient. In a later version of Morph, this rule may be revisited.

### Strings

All strings within the Morph protocol begin with a byte count (a 16 bit unsigned integer) followed by Unicode characters making up that byte length.

No reasonable identifier should ever approach anywhere near 16K characters. However, there may be a case for identifiers being longer than 127 characters. So to play it safe the byte length counter has been set to 16 bits in size.

## Link types

In *Morph Protocol.xlsx* go to the tab “Link types” and expand the link type called “Message”.

A Morph link always begins with a single byte called a “link byte”. The lower nibble determines the type of link and the upper nibble contains flags. The link byte is followed by data specific to that link type and in many cases that data conditional on flags.

The most significant bit is the MSB flag which species whether the rest of the link is written in MSB (Most Significant Byte order) or in LSB (Least Significant Byte order). Bits 4-6 are flags that have different meanings depending on the link type and sometimes also depend on each other.

### Example: Reading Message Link

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB** | **Flags** | | | **Link type** | | | | **Value** | | **Link type** | **Condition** | **Flag** | **Size** | | **Value** |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **Dec** | **Hex** |  |  |  | **Min** | **Max** |  |
| **m** | **z** | **y** | **x** | **1** | **0** | **0** | **0** | **8** | **8** | **Message** |  |  | **5** | **13** |  |
|  |  |  | x |  |  |  |  |  |  |  |  | HasCallNumber |  | 4 | CallNumber |
|  |  | x |  |  |  |  |  |  |  |  |  | IsForceful |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 4 | 4 | ToPathSize |
|  | x |  |  |  |  |  |  |  |  |  |  | HasFromPath |  | 4 | FromPathSize |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 0 | ToPathSize | ToPath |
|  |  |  |  |  |  |  |  |  |  |  | HasPathFrom |  | 0 | FromPathSize | FromPath |

The first byte in any link is the link byte that describes that link. The lower nibble (bits 0-3) specify the link type (in this case a Message link, with value 8).

Bit 7 specifies whether the link is composed with Most Significant Byte’s first. This applies to all multi-byte values throughout the link.

But now let’s look at the remaining three flag bits (bits 4-6).

Starting from the first row, bit 4 is checked, meaning it represents a flag called HasCallNumber. If HasCallNumber is set (ie. bit 4 is set) then we read in the 4 byte value called CallNumber.

On the second row, bit 5 is checked, meaning is represents a flag called IsForceful.

On the third row, no bit is checked, meaning we simply read in a 4 byte value called ToPathSize.

On the fifth row, bit 6 is checked, meaning it represents a flag called HasPathFrom. If HasPathFrom is set (ie. bit 6 is set) then we read in a four byte value called FromPathSize.

On the sixth row we read in a value called ToPath that ranges from 0 to <ToPathSize> bytes.

On the seventh row, if flag HasPathFrom is set, then we read in a value called FromPath that ranges from 0 to <FromPathSize> bytes.

### Example: Reading Service Link

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB** | **Flags** | | | **Link type** | | | | **Value** | | **Link type** | **Condition** | **Flag** | **Size** | | **Value** |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **Dec** | **Hex** |  |  |  | **Min** | **Max** |  |
| **m** | **z** | **y** | **x** | **0** | **0** | **1** | **0** | **2** | **2** | **Service** |  |  | **5** | **1 + str** |  |
|  |  | 0 | 0 |  |  |  |  |  |  |  |  | IsService | str | str | ServiceName |
|  |  | 0 | 1 |  |  |  |  |  |  |  |  | IsApartment | 4 | 4 | ApartmentID |
|  | x |  |  |  |  |  |  |  |  |  | IsApartment | IsApartmentNotShared | 8 | 8 | SessionID+P60 |
|  |  | 1 | 1 |  |  |  |  |  |  |  |  | IsApartmentProxy | 4 | 4 | ApartmentProxyID |

Iff bit 4 is unset and bit 5 is unset, then the flag IsService is set. Iff IsService is set then we read in a string value called ServiceName. ServiceName consists first of two bytes that specify how many more bytes to read in, being the bytes that consist the Unicode string.

Iff bit 4 is set and bit 5 is unset, then flag IsApartment is set. Iff IsApartment is set, then we read in a four byte value called ApartmentID.

Iff flag IsApartment is set, then bit 6 represents flag IsApartmentNoShared. Iff IsApartmentNoShared is set, then we read in an eight byte value called SessionID.

Iff bit 4 is set and bit 5 is set, then flag IsApartmentProxy is set. Iff IsApartmentProxy is set, then we read in a four byte value called ApartmentProxyID.

### Example: Reading Internet Link

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MSB** | **Flags** | | | **Link type** | | | | **Value** | | **Link type** | **Condition** | **Flag** | **Size** | | **Value** |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **Dec** | **Hex** |  |  |  | **Min** | **Max** |  |
| **m** | **z** | **y** | **x** | **1** | **0** | **0** | **1** | **9** | **9** | **Internet** |  |  | **5** | **3 + str** |  |
|  |  |  | x |  |  |  |  |  |  |  |  | IPv4/IPv6 |  |  |  |
|  |  | x |  |  |  |  |  |  |  |  |  | IsString | str | str | HostURI |
|  |  |  |  |  |  |  |  |  |  |  | IPv4 and not IsString |  | 4 | 4 | HostIPv4 |
|  |  |  |  |  |  |  |  |  |  |  | IPv6 and not IsString |  | 16 | 16 | HostIPv6 |
|  | x |  |  |  |  |  |  |  |  |  |  | HasPort |  | 2 | Port |

On the first row, iff bit 4 is unset, then flag IPv4 is set, xor iff bit 4 is set, then flag IPv6 is set.

On the second row, bit 5 is checked, meaning is represents a flag called IsString. Iff IsString is set, then read in a string value called HostURI.

If flag IPv4 is set and IsString is unset, then read in a four byte value called HostIPv4.

If flag IPv6 is set and IsString is unset, then read in a sixteen byte value called HostIPv6.

Iff bit 5 is set, then is represents. Iff bit 4 is set, then flag IPv6 is set.

On the last row, bit 6 is checked, meaning is represents a flag called HasPort. Iff HasPort is set, then read in a two byte value called Port.

## ValueType

Data Links hold the data payloads of messages, meaning the data that makes up parameters in RMI calls and replies. This data can be:

* Basic values, such as: integers and strings
* Of predefined data types, such as: dates, date-times, currency, etc.
* Complex types, meaning: struct and/or array
* Reference, meaning: servlet or stream

Each value must begin with a ValueType that will specify what sort of value is coming, such as a basic value, complex type or a refrences.

Lookup the spreadsheet page called “ValueType”.

* Bit 0 represents flag HasValueName. Iff HasValueName is set, then read in the string value called ValueName.

ValueName be a parameter name or the value’s field name within a struct.

* Bit 1 represents flag IsNull.

IsNull means that the value is null, pure and simple. All remaining rows have the condition that IsNull is False, so if IsNull is true, then there is nothing more to this value.

* Bit 2 represents flag HasTypeName. Iff HasTypeName is set, then read in the string value called TypeName. Ignore this row unless IsNull is unset.

If so, then read in the TypeName of the value.

TypeName could be a class name, the name of an enumerated type, etc.

There are predefined types such as data/time types and currency. These are standard types that should always be recognised. Applications can also specify their own specialised types.

And so on...

## Simple type

Read exponent values (e, s and u) as small numbers. Then use these use these as powers of 2. Look at the samples in the spreadsheet page called “SimpleType”.

# General rules

## Encoding

1. Morph MUST be encoded in UTF-8.
2. There is no padding. All data is packed.
3. Every LinkType byte contains a flag that specifies whether it is encoded as MSB or LSB.

Implementations ought to be able to read both MSB and 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 and SimpleType’s.

# 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

## End link

This requests the context it is sent to to stop, close, free itself, that sort of thing. However, such as request may be ignored at the contexts discretion.

Example: A resource expensive servlet is no longer needed by a client, so the client might send an End link to that servlet, thereby notifying that the client is happy for the servlet to be tidied up. However, the servlet might be in use by other clients, so the End request may be ignored.

## Message link

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

### Flags

* IsForceful

If the next step along a message path is not readily available, then an IsForceful message will try to establish a path, whereas a non-IsForceful message will be dropped.

**Example:** A message tries to go to another device, but no socket is to that device is available. An IsForceful message tells the system to try to establish a new socket connection to that device. A non-forceful message will allow itself to be dropped.

### Values

* CallNumber

Used for matching a call response (reply) to its waiting caller.

Must be unique within the context of an apartment proxy.

* ToPathSize

The size in bytes of the “To Path” of a message.

* FromPathSize

The size in bytes of the “From Path” of a message.

* ToPath

The “To Path” of a message. This consists of a series of links.

* FromPath

The “From Path” of a message. This consists of a series of links.

### 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 the message will need to contain enough information for a reply to be constructed that can find its way back to the caller.

## Data link

The data link contains the “payload” of messages, whether that might be parameters or return values of a method call, or a chunk of data in a stream, or an application exception.

If a method/property call or reply does not require data (such as a void method call), then the data link should be omitted altogether.

### Structure

#### Special Value

The first ValueType of a data link is called the “Special Value”. It is kept apart from any other parameters. The Special Value is used for the following purposes:

* Index value when calling a getter or setter.
* Return value of a method or getter call.
* Application exception.
* The stream data in a stream message.

When there is no need for a Special Value, then the special Value is encoded as a Null. (ie. a ValueType byte with value $02, meaning flag IsNull is set, and all other flags are unset.)

#### Parameters

Then follows a 4 byte counter called ParamCount that specifies the number of parameters that can be expected in the within the rest of the Data.

Then follows as many ValueType’s as ParamCount specifies (often none).

Parameter names may be omitted, but then parameter order ought to be kept intact, as some languages use parameter order to distinguish between overloaded methods of the same name.

### Flags

* IsException

Indicates that this Data link contains exception information. Exception information is structured in a specific way. See Exceptions XXX.

### Values

* ErrorCode

This is a Morph error code. See Error codes XXX.

* Size

This is the size, in bytes, of the following data.

* Data

Contains the data. XXX

## Service link

A Service link may specify a service, a shared apartment, a sessioned apartment or an apartment proxy. A service is a named apartment factory (though it may produce the same shared apartment every time). An apartment is holds a collection of servlets.

### Values

* ServiceName

The name of a service to which a connection is being established.

Once a service has been reached, it produces an apartment and replaces the Service link (ie. this link) with an apartment link (a Service link with flag IsApartment set). This causes messages to bypass the service, and instead go directly to the apartment.

Must be unique within the context of a device.

* ApartmentID

The ID of an Apartment. An Apartment contains a set of servlets.

Must be unique within the context of a device.

* SessionID

To help ensure that a session does not receive stray messages from other clients, a sessioned apartment expects calls to provide the correct SessionID.

**Note:** This is **not** a measure of security.

* ApartmentProxyID

The ID of an Apartment Proxy. An Apartment Proxy is the client side representation of an Apartment and contains a set of servlets proxies.

Must be unique within the context of a device. Does not need to match ApartmentID.

## Servlet link

A servlet is a handle to a business object on the server side. Servlet proxies are responsible for putting together RMI calls to their respective servlets.

### Values

* ServletID

This is the ID of a given servlet.

Must be unique within the context of an apartment. Each Servlet Proxy will have the same ID as its respective servlet.

## Member link

A member is a method or property of an object. This link specifies what member is to be called.

### Flags

* IsMethod/IsProperty

These flags specify whether to call a method or a property.

* IsGet/IsSet

In the case of a property, this specifies whether to call the property getter or setter.

* HasIndex

In the case of a property, this specifies that the property is an array property. If so, then the Special Value contains the array index value.

### Values

* Name

This is, rather unsurprisingly, the name of the method or property to be called.

## Process link

This refers to application processes on a device. This link type exists in an effort to make routing faster between processes on the same device by eliminating some of the overhead of TCP/IP.

### Values

* ProcessID

This is, rather unsurprisingly, the ID of the process on a device.

Must be unique within the context of a device.

## Internet

This generally refers to other devices. This link type helps rout messages between devices. Because process ID are essentially random, an internet link to localhost on port $E000 can be used to establish a connection to the local Morph daemon.

A good Morph implementation will optimize chains of internet links. If a chain can be optimized away completely, then that will be because the client and server are on the same machine, which will greatly improve efficiency.

### Flags

* IPv4/IPv6

Specifies whether the message should go over an IPv4 or IPv6 socket connection.

* IsString

Specfies that a domain name will be provided (as compared to an IP address).

* HasPort

Used when trying to connect over non-standard ports.

### Values

* HostURI

Either a domain name or an IP address in string form.

* HostIPv4

An IPv4 address. This is MSB because, strictly speaking, an IP address consists of 4 bytes.

* HostIPv6

An IPv6 address. This is MSB because, strictly speaking, an IP address consists of 16 bytes.

* Port

A TCP port number.

## Sequence link

Sequence links aim to ensure that messages don’t arrive out of order.

### Flags

* IsStart
* ToSender
* IsLast
* Resend
* IsLossless

### Values

* SequenceID
* SenderID
* Index

## Encoding link

Not yet defined.

Intended to enable Encryption, Validation and Encryption.

## Stream link

Not yet defined.

Intended to enable streaming of data from one source to multiple destinations.

# Morph Topology