Protocol for Operations Discovery (POD)

Specification V1.1
Matthew Ford
16th December 2010

Table of Contents

Introduction

This light weight protocol is designed to allow a general purpose application, PODapp™ (often a mobile phone app) to discover the functionality provided by a PODdevice™. PODdevices are simple microcontroller devices which implement this operation discovery protocol and have some form of serial data connection (often RS232 over bluetooth).
When queried with a GetMainMenu request, **{.}**, the PODdevice typically responds with a menu/navigation of commands and associated display names.
Subsequently when one of these commands is sent to the PODdevice, the device performs that operation and replies either with an empty message, **{}**, an updated menu, a new menu or a request for user input.
The protocol is robust and flexible. Most of the fields in the messages sent by a PODdevice are optional. Commands supported by the PODdevice are can usually be single character commands. This simplifies the programming required in the microcontroller to support the protocol.

Trade Marks

PODapp™ and PODdevice™ are trade marks of Forward Computing and Control Pty. Ltd. and their use is restricted. Forward Computing and Control Pty. Ltd. will only the grant use of these trade marks to a device or software that complies with this specification.

The format of POD messages

The POD protocol has two types of messages,:- messages which are enclosed by **{ }** and streaming raw data which is unformatted.

The following notation is used to describe the format of the messages:-
<*title*> is a series of characters excluding **}** , **~** and **|** e.g. Main Menu
<*prompt*> is a series of characters excluding **}** , **~** and **|** e.g. Input the required level
<*string*> is a series of characters excluding **}** , **~** and **|** e.g. test input
<*displayString*> a **<string>** which is displayed to the user to name/describe a command.
<*number*> is an integer number e.g. 15
<*cmd*> is a series of characters that PODdevice recognises as a command name. It cannot contain any of the characters **! ~ ` @ # $ % ^ & * ( ) _ = { } , . < > ? “ : ; '** .
It cannot have leading or trailing whitespace. Typically <cmd> is a single character to simplify the receiving code in the PODdevice.

The restricted characters can be used if a text format, see setLanguage below, is set and they are escaped.

If any of the above fields are completely blank (wholly white space) then they are assumed to be not present. (The sample implementation of this the phone application supporting this protocol goes further and removes leading and trailing white space from all fields before use.)

[ ] indicates optional block
*[ ]** indicates zero or more blocks

Messages the PODapp™ can send to a PODdevice™

There three types of messages the PODapp can send
{@ setLanguage
{. getMainMenu
{<cmd> commands

PODapp Message Formats and Examples

setLanguage format

{@ *<format1> [~ <format2>] * [| <language1>] [~ <language2>] ***** **}**

If used this message is often the first message sent by the PODapp to the PODdevice on establishing a connection. It requests the PODdevice to respond to subsequent messages using the text format and language from one of those specified.

The PODdevice can reply to this message with either an empty message **{}** or a Language **{@** message.
The empty message **{}** indicates that the PODdevice does not support any of the formats or languages request and that communication in both directions will proceed using the defaults (no text formatting, English using the ASCII characters set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space)
The PODdevice Language message **{@** indicates which one of the preferred formats and languages the PODdevice can support and all future messages in both directions will use these settings.

The PODdevice is not required to support anything but the default values described below but will honour the PODapp's stated preference if it can.

The PODapp is required to support the default values but can also support additional formatting, languages and request the PODdevice to respond using these if the PODdevice supports them.

The optional fields is separated by | and within each field multiple values can be specified separated by ~ . The values are ordered by preference, with the most preferable listed first.

The <*format*> field defines the types of text formatting the PODapp can understand. It can take values from those defined here below, or any other value that is recognised by the PODdevice being connected to (i.e. device defined). If no value is specified, the default is no special text formatting and no escaped characters.

The defined **<format>** field options are :- POD , HTML

> **POD** basic text formatting consisting of
> *strong* for bold i.e. **strong**
> _italics_ for italics i.e *italics* and
> +underline+ for underline i.e. underline
> and combinations of these.
> '\' is an escape character. That is **\}** will be parsed as **}** and not as the endOfMessage

character. Similarly the following escape sequences **\{** , \| , **\~** , \, ***** _ \+ will be parsed as a single character without any special formatting or message processing. All other characters can also be escaped but that is not particularly useful. Finally \\ is used to send the character \

HTML for html text formatting. Note this implies support for HTML encoded and escaped characters. The character encoding is set by the <*language*> field as described below. NOTE: HTML formatting is only text formatting. Page layout formatting such as <OI> <UL> <TABLE> etc. are ignored.

The <*language*> field defines the preferred language the PODapp would like to receive messages in. It can take values from ISO 639-1. If no value is specified the default is English with a character set of 7bit ASCII. If <*language*> value specified then the character set defaults to UTF-8. (see the examples below)

In all cases any of the following combinations:- *carriage return*(0x0d) OR *carriage return* + *linefeed* (0x0d 0x0a) OR *linefeed* (0x0a) will represent a new line sequence

Examples

{@} requests the PODdevice communicate using the default values. Once the PODdevice responds, all subsequent messages in both directions will be in English with no special text formatting and using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space. This is the simplest form of the message.

{@|fr~en} requests the PODdevices's communicate in preferable French or if French is not supported then in English using no special formatting and UTF-8 as the character set.

If the PODdevice responds with the empty **{}** message then it means the neither French nor English in UTF-8 is not supported. Only the default language English and using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space, is supported by the PODdevice.

If the PODdevice responds with the Language message **{@|fr}** then all subsequent messages in both directions will take place in French using UTF-8 with no special text formatting or character escaping.

If the PODdevice responds with the language message **{@|en}** then all subsequent messages in both directions will take place in English using UTF-8 with no special text formatting or character escaping. Note: since UTF-8 includes 7bit ASCII characters, most PODdevices responding with this message will only actually use 7bit ASCII to communicate.

{@HTML~POD|ja} requests the PODdevices's communicate in Japanese using UTF-8 character set and either HTML text formatting, or if HTML is not supported by the PODdevice, then POD text formatting.

If the PODdevice responds with the empty **{}** message then it means Japanese is not supported and neither HTML nor POD text formatting is supported. Only the default language English and using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space, is supported by the PODdevice.

If the PODdevice responds with the Language message **{@|ja}** then all subsequent messages in both directions will take place in Japanese using UTF-8 with no special text formatting or character

escaping.

If the PODdevice responds with the Language message **{@POD|ja}** then all subsequent messages in both directions will take place in Japanese using UTF-8 with POD text formatting.

If the PODdevice responds with the language message **{@HTML|ja}** then all subsequent messages in both directions will take place in Japanese using UTF-8 with HTML text formatting.

Although not usual used, the specification allows the PODapp to send the setLanguage message at any time to request a change in text formatting and language. If the PODdevice supports the requested formatting and language, then the change to that formatting and language occurs immediately after the PODdevice sends its Language message response indicating the format and language to be used.

getMainMenu Format

{.}
This message requests the PODdevice's main menu.

If the PODapp does not send a getLanguages message on establishing a connection, this is the first message the PODapp sends to the PODdevice and all messages in both directions are assumed to be in English using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space and with no text formatting or character escaping.

The PODdevice responds to this message, **{.}** , with its main or top level message. The PODdevice's response is usually a Menu message, **{.** , or a Navigation Input message, **{^** , described below, but can be any of the 10 PODdevice messages including **{}**

The specification also allows the PODdevice to respond with an empty **{}** message which indicates there is no main menu/navigation or user input. In this case the PODapp must know all the valid commands available on the PODdevice. This type if implementation is NOT recommended.

Commands format

{ *<cmd> [| <arg1> [, <arg2>]]** **}**

These messages are sent by the PODapp to the PODdevice and contain commands that the PODdevice has advertised to the PODapp that the PODdevice can handle. The messages can have additional fields containing the users input. (see the PODdevice Input examples below).

When the PODdevice has received this Command message from the PODapp, the PODdevice always responds with a PODdevice message, perhaps just the empty message, **{}**

Messges a PODdevice™ can send to the PODapp™

There are 10 messages types that a PODdevice can send to the application. They are indicated by the first character after the message start **{** character

{} Empty
{@ Language
{. Menu
{: Update menu/navigation items
{^ Navigation input
{= Streaming raw data
{' String input
{# Numeric input
{? Single Selection input
{* Multiple Selection input

The messages are terminated by a **}** character.

Most of the PODdevice messages specify a command the PODdevice will respond to and have optional fields containing information to display to the user about this command.

PODdevice Message Formats and Examples

These commands are sent from the PODdevice. They are only send in response to a command sent by the PODapp to the PODdevice

Note: The screen displays shown below are examples only. This specification does not define how the messages are displayed to the user.

Empty Message format

{}
This message is sent back by the PODdevice when no other command is appropriate. It indicates the last command set to the PODdevice has been received and acted on. Note: there are no spaces between the **{}**

If the PODdevice sends this message in response to a menu item command or navigation direction command then the PODapp is expected to continue to display that menu or navigation input screen to allow the user to make further selections. In contrast, when the user's input is sent by the PODapp from a user input screen, then that screen is dismissed. What is displayed instead depends on what the PODdevice sends back. If the PODdevice sends back **{}** or **{:** then the PODapp can display something it thinks is appropriate, e.g. the last menu. Otherwise the PODapp displays what the PODdevice sends.

Language Message format

{@ [*<format1>] [| <language1>]* **}**
This message is sent by the PODdevice in response to a setLanguage, **{@** , message from the PODapp

The *<format>* can be either be omitted for no formatting, or can be one on the format strings

described in the PODapp setLanguage message above

The |<*language*> can be either omitted for the English language using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space), or can be one of the values from ISO 639-1 in which case the character set defaults to UTF-8. (see the examples below)

In all cases any of the following combinations:- *carriage return*(0x0d) OR *carriage return* + *linefeed* (0x0d 0x0a) OR *linefeed* (0x0a) will represent a new line sequence

Examples

If the PODapp sends {@|fr} then if the PODdevice response with:-

{} then it means French and UTF-8 is not supported. Only the default language, English using the 7bit-ASCII character set with characters outside the range *space* (0x20) to ~ (0x7e) treated as white space is supported by the PODdevice.

{@|fr} then all subsequent messages in both directions will take place in French using UTF-8 with no special text formatting or character escaping.

If the PODapp sends {@HTML~POD|ja} then if the PODdevice response with:-

{@POD|ja} then all subsequent messages in both directions will take place in Japanese using UTF-8 with POD text formatting.

{@HTML|ja} then all subsequent messages in both directions will take place in Japanese using UTF-8 with HTML text formatting.

Although not usual, the specification allows the PODapp to send the setLanguage message at any time to request a change in text formatting and language. If the PODdevice supports the requested formatting and language, then the change to that formatting and language occurs immediately after the PODdevice sends its Language message response indicating the format and language to be used.

Menu Message format

{. *[<title>] [, <re-requestTime mS>] [| <cmd> [~ <displayString>]]* * **}**
The menu command starts with **{.** and has an optional title and optional re-request time followed by zero or more commands and their associated menu display strings. Menu items with empty <*cmd*> (all white space) are ignored. Menu items with empty <*displayString*> cannot be selected but can be updated to a non-blank **<displayStrings>** by a subsequent Update message, **{:**.

Each menu item must have unique command associated within. The <cmd>s only have to be unique within a menu. They can be reused across different menus

Only the menu items that are available for selection (i.e. with non-blank <*displayStrings*>) can send commands back to the PODdevice.

The <*re-requestTime mS*> is a hint to the PODapp to re-issue the command that returned this menu in <*re-requestTime mS*> milliseconds time. This is used for menus that will change state at some time in the future due. The PODapp may ignore this hint.

If the PODdevice responds with to a menu command with an empty message **{}**, the menu screen,

like the navigation screen, remains displayed for further input. This is in contrast to other input screens which are usually dismissed once the user's selection is sent to the PODdevice.

Examples

{. } an empty menu with no title. Not very useful. Don't confuse this with **{.}** command sent to the PODdevice by the PODapp to request the main menu

{.PC Main Menu|o~on|u~up|d} displays

The **on** menu item will send the **{o}** cmd to the PODdevice when selected and Send pressed.
The **up** menu item will send the **{u}** cmd to the PODdevice when selected and Send pressed
The **d** menu item did not have a display string so the user cannot send this command back to the PODdevice. However the PODdevice can update this menu later to assign a displayString to this command, see the Update Menu examples below.

{.PC Main Menu,60000|o~on|u~up|d}
Displayed as above but with a hint to the PODapp to re-issue the command that returned this menu in 60secs (60000mS) to see if it has changed. The PODdevice can update or remove the re-request time each time it responds to the re-issued command.

Update menu/navigaion items Message format

{: *[* **,** *<re-requestTime mS>] [| <cmd> [~<string>]] [| <cmd> [~ <string>]] ** **}**
This command updates menu/navigation items on the currently displayed menu or navigation input screen. The items are matched by *<cmd>*.

Only existing menu or navigation items can be updated with this message. Additional items cannot be added. If a item which currently has a blank *<displayString>* is updated to a non-blank *<displayString>*, then that item becomes available for selection. If an item is updated to have a blank *<displayString>*, then that item becomes un-available for selection.

Only items that are available for selection (i.e. with non-blank **<displayStrings>**) can send commands back to the PODdevice.

The *<re-requestTime mS>* is a hint to the PODapp to re-issue the command that returned this update message in *<re-requestTime mS>* milliseconds time. This is used for menus/navigataions

that will change state at some time in the future due. The PODapp may ignore this hint.

Examples

{: } does nothing to the current menu/navigation
{:|d~down|s~setup} updates the menu above (see Menu Message examples) by changing the display for the **d** command to “down”. The extra menu item “setup” is not added to the menu as its associated command **s** does not match any existing menu command

{:,100|o~turning On} updates the on menu item **o** to display “turning On” and hints to the PODapp that it should re issue the command that returned this message in 1/10th sec.

At some time in the future you would expect the reply to come back as
{:|o~Off}
indicating that the PODdevice state is now fully on and sending the **o** command will turn it off.
Note: there is no re-request time field in this final message as the PODdevice state will not change until another command is sent to it by the PODapp.

{:|d} or **{:|d~ }**
Updates the menu item associated with the d command to blank its display string. This menu then can no longer send back the **{d}** message to the PODdevice.

Navigation Message Format

{^ *[<prompt>] [, <re-requestTime mS>] | <leftCmd> [~ <leftLabel>] | <rightCmd> [~ <rightLabel>] | <upCmd> [~ <upLabel>] | <downCmd> [~ <downLabel>] | <homeCmd> [~ <homeLabel>]* **}**

This command opens a navigation input screen which typically displays four arrow buttons around a home button. Note: however the exact format of the display is not part of this specification. The user can then select either left, right, up, down, or home (usually a centre button). The selection sends back to the PODdevice the associated command. For example if the user selected the down arrow the PODapp sends back **{** <*downCmd*> **}** to the PODdevice.

Direction items with empty <*cmd*> (all white space) mean that that direction cannot be selected and cannot be updated by later Update messages. Direction items with empty <*displayString*> cannot be selected but can be updated to a non-blank <displayStrings> by a subsequent Update message.

Each direction must have unique command associated within. The <cmd>s only have to be unique within a Navigation input. They can be reused across different menus/navigation inputs.

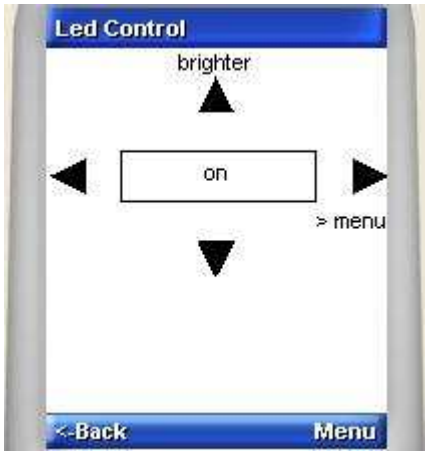
Only directions that are available for selection (i.e. with non-blank <*displayStrings*>) can send commands back to the PODdevice.

If the PODdevice responds with an empty message **{}**, the navigation input screen, like a menu screen, remains displayed for further input. This is in contrast to other input screens which are usually dismissed once the user's selection is sent to the PODdevice.

Examples

{^Led Control,200||m~menu|u~brighter|d|o~on}

opens a navigation input screen associated with a prompt of “Led Control” and a suggested refresh time of 200mS. The left button has no command and is inactive and will not be able to updated by a later Update message. The left button is the first section and in this message it is empty, || The right button is labelled **menu** and it associated command is **m**. The up button is labelled **brighter** and its associated command is **u**. The down button has the **d** command associated but is in-active at the moment because it has not <*displayString*>. The home button is labelled **on** and its associated command is **o**.



Pressing the Home button will send the {o} to the PODdevice which might respond with

{:|d~darker|o~off}

This would change the home button label to **off** and activate the down button by giving a non-blank label of **darker**.

Streaming raw data Message format

{= *[<title>]* **}**
This command turns on the streaming raw data mode. In this mode the PODdevice can send any characters to the application for display. If the text format setting allows escapes then an escaped **{** can be sent in raw data. No text formatting, other then escape sequences, is applied to raw data.

Typically this command is used to show the PODdevice' status, either once off or continually updating.

The streaming raw data mode is exited by the a PODapp sending another command message to the PODdevice. This is often initiated on the PODapp by the user pressing the back button and requesting the previous menu from the PODdevice.

Examples

{=} Set the streaming raw data mode without a title specified.

{=Light Level} Sets the raw data mode with a title of Light Level.
Typically after the PODdevice has received either of these messages, the PODdevice would then start sending the raw data messages. For example
Level 4
Light 2.6%

Data can be sent at any time the PODdevice is not sending another message, provided it does not contain an unescaped the start of command, **{** , character. The negotiated text formatting determines if escape sequences are supported.

Whether this data is captured or ignored when the PODapp is not in Streaming Raw Data mode is up to the application.

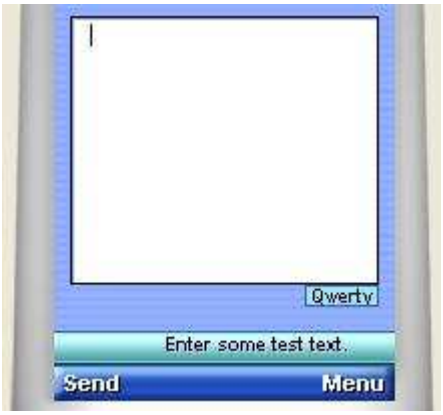
String input Message format

{' *<cmd> [~ <prompt>] [| <string>]***}**
This command opens a text entry screen with the prompt <prompt> and the initial text <string> and associates <cmd> with the response. When the user wants to send their text entry, the application sends
{ *<cmd>* **|** *<newString>* **}** back to the PODdevice

When the user's input is sent by the PODapp from a user input screen, that screen is dismissed, i.e. no longer available for user input.

Examples

{'s} Opens a text entry screen with no prompt and no initial text and associates the command s with the response.
{'s~Enter some test text.} Displays a input screen like the one below. The initial text is all banks.



Assuming the user enters the text “Hello World” and presses send, then the application would send **{s| Hello World}** back to the PODdevice. The PODdevice would then process the **s** command using the **Hello World** as the data.

{'s~Edit the message|Hello World}
when sent by the PODdevice, the PODapp would display something like

Numeric Input Message Format

{# *<cmd> [~ <prompt>] | <numberMax> [* **,** *<numberCurrent> [* **,** *<numberMin>]]* **}**
This command opens an integer entry screen with prompt <prompt>. The input is bounded between Max and Min and with a current value of Current. The numbers are separated by commas. If Current or Min are missing 0 is used. Negative numbers are supported. The PODapp is expected to swap Max and Min so that Max > Min and then limit Current to fall in the range, after which Max >= Current >= Min will always hold true.

When the user's input is sent by the PODapp from a user input screen, that screen is dismissed, i.e. no longer available for user input..

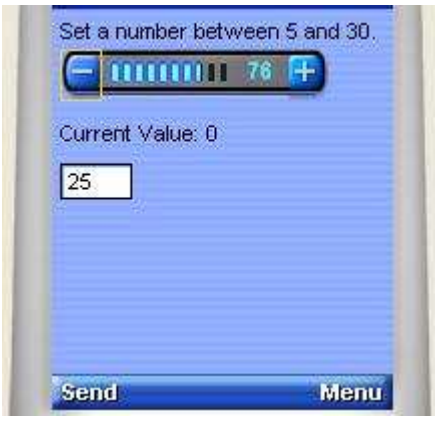
Examples

{#n|5}

Opens a numeric input screen and associates the PODdevice command **n** to it. There is no prompt specified and the user input is restricted to be between 5 and 0, inclusive. If the user enters 3 and presses send, the application will send back the users input to the PODdevice with the command **{n|3}**

{#n~Set a number between 5 and 30.|30,25,5}

Opens a numeric input screen with the prompt of “Set a number between 5 and 30.” and with an initial value of 25.



Other implementations of PODapp may display this screen differently.

When the user presses send the application sends
{n|25}
back to the PODdevice for it to process the **n** command using **25** as the data.

Hint: If you want to enter a decimal number e.g. 35.333, then use a string input screen instead.

Single Selection List Message Format

{? *<cmd> [, <initialSetting>] [~ <prompt>] | <label0> [| <label_N>]** **}**

This command opens an input screen were the user can select one of the multiple options to send back to the PODdevice. If the user selects the first option then **{** *<cmd>* **|0}** is send back. If the user selects the second option then **{** *<cmd>* **|1}** is sent back to the PODdevice and so on.

The *<initialSetting>* is the index, zero based, of the currently selected option. The PODapp is expected to normalize *<initialSetting>*s so that at least one option is always selected.

When the user's input is sent by the PODapp from a user input screen, that screen is dismissed, i.e. no longer available for user input.

Examples

{?i,0~Do you want to enable timeout?|Enable|Disable} opens a single selection input screen associated with the **i** command, with a prompt of “Do you want to enable timeout?”. There are two possible choices, “Enable” and “Disable”. The Enable option is initially selected.

If the user presses send with Enable selected, then the PODapp will send back
{i|0}
to the PODdevice for it to process the **i** command with **0** as the data

Multiple Selection List Message Format

{* <*cmd*> *[,* <initialSetting> *] * [~* <*prompt*> *|* <*label0*> *[|* <*label_N*>*]** **}**

This command opens an input screen were the user can select zero or more options to send back to the PODdevice. If the user selects no options then **{** <*cmd*> **| }** is sent back to the PODdevice. If the user selects the first and third options then **{** <*cmd*> **|0,2}** is send back. If the user selects the second option then **{** <*cmd*> **|1}** is sent back to the PODdevice.

When the user's input is sent by the PODapp from a user input screen, that screen is dismissed, i.e. no longer available for user input.

Examples

{*m,0,1~Choose the options you want|Fade on/off|3 Levels} opens a multiple selection input screen associated with the **i** command, with a prompt of "**Choose the options you want**". Both options are currently selected in the PODdevice.

If the user presses send, then the PODapp will send back
{m|0,1}
to the PODdevice for it to process the **m** command with **0** and **1** as the data.

Displaying long messages to the user

If the PODdevice has a message to display to the user, it can send it as Menu item in its response to the PODapp command message. When the use presses Send the command associated with this menu time is sent to the PODdevice which can then respond with some suitable menu or input screen.

For text that would make the PODdevice message longer then 255 characters limit, either a **more** menu item can be added or the a Streaming Raw data message can be sent and the text then sent as raw data.

Examples

The first example below shows the menu approach. The first screen shows the first block of text as a menu item. In this example the command for this menu item will display the second page of text.

When Send is pressed, the command associated with this menu item is sent to the PODdevice, which then returns the second page as a menu.

The PODdevice responds to the command associated with this menu item by returning a new menu or input screen as appropriate.

The second example uses a Streaming Raw Date screen to display long messages.

The PODdevice first sends

{=Message Example}

a streaming raw data message with the title of the screen and then sends the message as raw data.

Required POD Support

This section lists the functions that the PODapp and the PODdevice must support in order to implement this protocol.

PODapp

The PODapp (for example, the mobile phone app) is required to support:-

1. The automatic sending of the GetMainMenu request, **{.}**, on establishing a connection. The exact form of the communication and how a connection is established is not specified.
2. The display the Menu, UpdateMenu and StreamingRawData, StringInput, NumericInput , YesNoInput, SingleSelectionList, MultiSelectionList and Navigation messages from the PODdevice and to handle the Empty message. The exact display format of these messages in not specified
3. A 'debug' mode in which the user can view all the transmissions to and from PODdevice and the PODapp.
4. A configurable message response timeout per connection, and at least 1 retry before disconnecting from the PODdevice. The configuration for each PODdevice connection sets the inter character timeout. The PODdevice is expected to respond to a message sent to it within 5 times the inter character timeout. Once the application starts receiving a response, other then streaming data, the next character is expected to be received within the character timeout.
5. Termination of a message after 256 characters. The truncated message is still processed by assuming an implicit end **}** character after the 255th character. Note if the 255th character is a \ then no escape is implied and the implicit **}** is treated as true end character.
6. Limit messages sent to the PODdevice to less than or equal to 256 characters including the start **{** and end **}** characters.

Optionally the PODapp can support requesting particular languages and text formatting and act on the re-request times sent by the PODdevice

PODdevice

The PODdevice is required to:-

1. Respond to all messages, even unrecognised ones, with a reply message. This message can be the Empty message **{}**
2. Respond to the GetMainMenu request, **{.}** , with a main menu or naviagation input of top level functionality or, less commonly, with one of other messages:- UpdateMenu and StreamingRawData, StringInput, NumericInput , SingleSelectionList, MultiSelectionList or the Empty message **{}**
3. If the connection is half-duplex then the messages from the PODapp to the PODdevice take precedence over, and can interrupt, messages coming from the PODdevice.

Optionally, the PODdevice can support other Languages and text formatting and respond to a setLanguage message with non-empty message, a Language message.

Message Response Timeouts

These timeouts are designed to detect loss of connection to the PODdevice. When a PODdevice receives a message, it is expected to send the starting character of the response, **{**, within 5 times the character timeout setting for its connection. If it application does not receive a response within this time, the application can assume connection to the PODdevice has been lost. Note: this connection timeout is set in the application to match the processing time of the PODdevice. Nothing special is needed in the PODdevice other then to respond to the message

Once the starting character has been sent by the PODdevice, each following character must be sent within the character timeout of the sending the previous character, otherwise the application can assume connection to the PODdevice has been lost. Setting a character timeout of 0 means the connection never times out.

Also the closing character of the response, **}**, must be sent within 256 characters of the opening character, **{**. After 256 chars the application can truncate the response and commence processing it. Note: the length of streaming raw data is unlimited.

Usually the application will ignore user requests to send another message while it is waiting for a response from the PODdevice. However for the special case of 0 character timeout, the sample implementation of the application for a JME2 mobile phone does not block the sending of multiple messages to the PODdevice.

PODdevice coding guidelines.

These guidelines are not part of the formal specification of a PODdevice. They are based on the specification above but are guidelines only.

The PODdevice should respond to the **{.}** message by sending back a main menu or navigation input of its functions. It is also acceptable for a very simple PODdevice to send back just an input screen **{?**, **{#** or **{'** or **{^** or **{*** or a streaming raw data screen **{=** , if this is all the functionality the PODdevice provides.

The PODdevice must respond to every message sent to it. The PODdevice should send back an empty message, **{}** , if there is nothing else to send back.

The PODdevice should normally NOT send any message to the application unless responding to a message that was sent to the PODdevice. This and having the PODapp messages take precedence over responses or raw data the PODdevice allows the use of a half-duplex connection. This simplifies programming of micro-controllers that do not have a built-in RS232 interface.

The PODapp should normally wait for the PODdevice's response before sending another message. If the PODdevice does not respond within the timeout periods discussed above, the application should close the connection.

However the PODdevice can always send any streaming raw data that does not contain the character sequences
{. or **{:** or **{'** or **{?** or **{#** or **{*** or **{^** or **{=** or **{}**

After the PODdevice sends a streaming raw data message, **{=** , it can send any sequence of characters.

The commands that the PODdevice advertises are usually single characters, as in the examples above, to simplify the coding in the PODdevice. However the PODdevice can use multi-character commands if it chooses

The sample mobile phone application and sample micro-controller provided uses a Bluetooth connection to a half duplex software RS232 interface on the micro-controller. However this protocol does mandate any particular type of connection. It only deals with the format of the messages sent and received.