Control System Protocol Developers Manual

Protocol version: 1.3
Updated: 11th April, 2012.

Table of Contents

Introduction

This document has been created for developers to create backend modules for the CommandFusion line of Viewer solutions, such as the iViewer for iPhone and iPod Touch, iPad and Android devices.

The intended audience is home automation programmers and software developers.

More information on the CommandFusion software products can be found at www.commandfusion.com

Communication Methods

There are two main communication methods that can be used:

1. **Control System Protocol** - defined in Project Properties of your guiDesigner project.

2. **External Systems** – defined in System Manager of your guiDesigner project.

This document only relates to the first communication type – Control System Protocol. Systems defined in the System Manager do NOT use this protocol. Instead you create custom commands and feedback processing for those systems.

Control System Protocol

Message Format

All messages in the CommandFusion protocol take the form of:

```
[identifier]=[value][h03]
```

Each message is delimited with the End Of Message (EOM) hex value of 03. This delimeter will appear as `h03` in all message examples throughout this document.

Indentifiers

The following identifiers are valid within the protocol:

- Digital Joins - d#
- Analog Joins - a#
- Serial Joins - s#
- Password Validation - p
- Initialize - i
- Orientation Mode Change – m
- Heartbeat Message – h

- Lists – l# (lowercase L)

Digital, Analog and Serial join identifiers must be combined with a join number (starting at 1). For example `d12=0h03`, where the join number is 12.

Digital Joins

Digital joins can have a value of either 0 or 1 (low or high, off or on). A typical Digital join message looks like this:

```
d1=1h03
```

The above message sets digital join 1 to a value o f 1 (high).

The following GUI elements make use of Digital joins:

1. Buttons
    a. Changes button state depending on value (0 = inactive state, 1 = active state).
    b. On Press, sets the join value to 1 (high).
    c. On Release, sets the join value to 0 (low).
2. Sliders
    a. On Press, sets the join value to 1 (high).
    b. On Release, sets the join value to 0 (low).
3. Input Fields
    a. Button press on same join as input field forces input field to send its text value to the Server.
4. Pages
    a. Flip to a page when its join goes high.
5. Subpages
    a. Show subpage when its join goes high.
    b. Close subpage when its join goes low.
6. Webpages
    a. To refresh, stop, go forward/backward
7. Video objects
    a. To play/pause or stop the video stream.

Analog Joins

Analog joins can have a value of between 0 and 65535 inclusive. A typical Analog join message looks like this:

```
a23=54677h03
```

The above message sets Analog join 23 to a value of 54677.

The following GUI elements make use of Analog joins:

1. Gauges
    a. Sets the value of the gauge, 0 being lowest, 65535 being highest.
2. Sliders
    a. Sets the value of the slider, 0 being lowest, 65535 being highest.
    b. On Press, Drag and Release, sets the join value to the relative value based on pressed/dragged position within the slider.
    c. Slider indicator value is calculated from its analog join value, using the Slider's min and max attributes to scale the value.

Serial Joins

Serial joins can have any string value. A typical Serial join message looks like this:

```
s12=Hello World!h03
```

The above message sets Serial join 12 to a value of 'Hello World!' (without quotes).

The following GUI elements make use of Serial joins:

1. Text
    a. Sets the value of the text object to be displayed.
2. Buttons
    a. Inactive and Active states each can have a Serial join to dynamically change the button's text value.
3. Images
    a. Dynamically change the image's path to load a dynamic image via the web, or an image included in the current GUI.
4. Input Fields
    a. Set the displayed text within the Input Field.
    b. When a button is pressed on the same digital join as the input field, the Input Field's text value is sent to the processor.
5. Web objects
    a. Set the URL of the webpage to be displayed.
    b. To show the title of the page being displayed in a text field, etc.
6. Video Objects
    a. Set the URL for the video stream.

Password Validation

Connections between the Viewer and Server can be password protected to prevent someone connecting to your Server without authorization. The password is set on the Server side, and in the Viewer settings.

At the initialization of a communication session between the Viewer and the Server, the Viewer will send a message containing the connection password.

The Server will then respond with a message telling the Viewer whether or not the password has been accepted.

The message sequence goes like this:

1. Connection made
2. Viewer sends password
   `p=<PasswordValueEnteredInSettings>h03`
3. If the password matches what's in the Server settings, the Server will reply with:
   `p=okh03`
4. If the password doesn't match, the Server will reply with:
   `p=badh03`

If the password doesn't match, the Viewer will not load the GUI and will show an error message to the user informing them of the password error.

Initialize

Once the password has been authenticated, the Viewer will send the following initialization message:

```
i=1h03
```

The Server will then reply with a group of messages informing the Viewer of any join state values. Only joins with values other than 0 (for Digital and Analog joins) or blank (for Serial joins) will be included in this group of messages.

This is simply used to initialize the GUI state, updating button states, slider values, text values, etc.

A typical group of initialization messages will look like this:

```
d12=1h03a1=4555h03s43=Hello World!h03
```

As you can see, this is just a group of standard join messages (as described earlier in this document) concatenated, each message ending with the delimiter value of hex 03.

Whenever a message is received by the Server, it should always check for multiple commands in the one message, and split then parse them appropriately.

Orientation Mode Change

The iViewer supports two orientations, portrait and landscape. This is mainly utilized in the iViewer (for iPhone and iPod Touch) when rotating the device, but is also accessed via the webViewer (for cross-browser control) when resizing the browser window so that its width becomes larger than its height, or vice-versa.

When the orientation mode of the Viewer changes, it sends the following command, depending on the mode being changed to:

```
m=portraith03
```

or

```
m=landscapeh03
```

Again, the value is the orientation mode being change *to*.

Heartbeat Message

The iViewer application utilizes a heartbeat message to ensure the connection between the iPhone and the Server remains stable.

iViewer sends a heartbeat command every 3 seconds, and expects to receive a reply heartbeat command within 5 seconds or it will show a message warning the user that the connection is experiencing difficulties.

iViewer sends the following heartbeat command:

```
h=0h03
```

and expects the heartbeat reply message of:

```
h=1h03
```

Dynamic, scrolling Lists

The iViewer application has a very flexible list generation system that allows creation of custom lists with a large emphasis on customizability.

Due to this flexibility, the protocol for sending list data and parsing list feedback is a lot more complex than the rest of our protocol.

Lists can be created in vertical and horizontal orientations, allowing you to create vertical lists on portrait pages, and horizontal lists on landscape pages. Although you are also able to do the inverse if you wish (vertical lists on landscape pages, etc).

Lists are separated into 4 main 'items' (which are defined as subpages in guiDesigner). Header, Title, Content and Footer.

Each item can contain any number of GUI objects, such as text, buttons, images, gauges, etc. The only objects that cannot appear on an item's subpage are input fields and other subpages.

The header item appears only once in the list at the very top (or left if using horizontal lists). The header is optional and only appears if defined in guiDesigner.

Title items can appear multiple times in a list. These should be designed in a way to separate content meaningfully, such as separating a list of artists alphabetically, grouping artists starting with 'A', 'B', etc, together.

Content items are the main part of lists where your information should be displayed.

Footer items appear only once at the very bottom (or right) of the list. Footer element will be visible only if the list object defines the Footer subpage within guiDesigner.

List generation is quite an intensive process, so we recommend keeping list items as light as possible. Assigning a background color to the list object itself will in turn disable transparency for all other list subpages, drastically increasing performance. So this is a route we recommend for all heavy list usage.

List Generation Protocol

```
  REC_SEP = \x0D
FIELD_SEP = \x1E

l<list join>=<row command>REC_SEP<row command>REC_SEP ...
REC_SEP\x03
```

Row commands:

```
<row index><command><FIELD_SEP>[fields definition]
```

Available Commands (# = <row index>, 0 based):

```
 #t   title element at row #
 #c   content element at row #
 #it  insert title element at row #
 #ic  insert content element at row #
 #d   delete element at row #
 #-#d delete range of elements from rows # to #
(inclusive)
 0x   clear the entire list contents
```

Fields definition:

```
 <join in subpage>=<join value><FIELD_SEP>
```

List Examples

Below are some examples of how to generate list data. Please note that any line wrapping in the below commands is to be ignored. All commands should be a single line of text.

Create a list on analog join 10, with one title row and two contents rows:

```
l10=0t\x1Es1=title\x1Es2=subtitle\x1Ea1=50\x0D1c\x1Es1=conte
nt element 1\x0D2c\x1E s1=content element 2\x0D\x03
```

In the above example, the 'title item subpage' in guiDesigner has a single text object on join 1. The 'content item subpage' has a single text element on join 1 and a single gauge on analog join 1.

Clear the entire list on join 5:
```
l5=0x\x03
```

Update element 2 in the list (on join 10):
```
l10=1c\x1Es1=new value for content element 2\x0D\x03
```

Insert element between 1 and 2 in the list (on join 10):
```
l10=1ic\x1Es2=inserted element\x0D\x03
```

Delete elements 1 and 2 in the list (on join 10):
```
l10=0-1d\x0D\x03
```

List Feedback Protocol

Feedback from objects within list elements are very similar to that of normal GUI objects explained at the beginning of this document, with a little twist.

All list feedback data is pre-fixed with the list join, as well as the item number in the list the feedback is coming from. The format is as follows:

```
l<list join>:<item index>:<join>=<new value>
```

For example, the next example demonstrates feedback from a button on digital join 2 coming from list element 5 on list join number 3:

```
l3:5:d2=1\x03
```

The data elements are separated with colons, with the standard message delimiter. The last element is exactly the same as a normal button press.

So using this data, you can determine which list, list element and GUI object the feedback came from and use the information as needed.

Glossary

guiDesigner – Windows software for creating custom Graphic User Interfaces.

Viewer – Software that renders the GUI generated by guiDesigner

iViewer – a Viewer for iPhone, iPad, iPod Touch and Android devices.

Server – Hardware or software that Viewers connect to and communicate with.

Server Module – Software that runs on the specified Server hardware platform allowing Viewers to connect and communicate with the Server.