Socket App User Guide

V1.0 – MAB

1. Introduction

The Socket App provides an interface for basic incoming and outgoing network communication. Text based messages can be received and parsed for number values that the robot program can copy into variables. The robot program can send text-based messages which may contain formatted number values from variables.

2. Use-Cases

Due to the low-level nature of TCP and UDP communication the Socket App can be useful in many situations. The main limitations are that the protocol that must be text based and simple enough to be parsed by the generic parsers of the Socket App. This rules out many use cases that use more complex communication schemes (e.g. popular protocols like MQTT and Modbus but also the robot control's CRI interface).

Simple enough use cases may include:

- Receiving XYZ positions from an object recognition camera
- Receiving position data from a custom data source
- Receiving sensor data to be used in IF/ELSE decisions
- Sending position and program state information to a data sink, e.g. for logging or displaying
- Synchronizing two robots (see example at the end of this document)

3. Features

3.1 Connection

The Socket App supports the following transport layer communication types:

- TCP client (connect to a server)
- TCP server (waits for an incoming connection from a client)
- UDP (connection-less)

The application layer protocol is user defined but must be ASCII text based. The communication target can be specified as IP address or hostname but IPv6 addresses may not work.

3.2 Message Parsing

Received text-based messages can be parsed for number values using one of the following parsers:

- Simple parser – Type the expected message with placeholders for number values.
- Regex parser – Very powerful but also difficult to write and read. This is intended for experts.

Binary messages cannot be parsed.

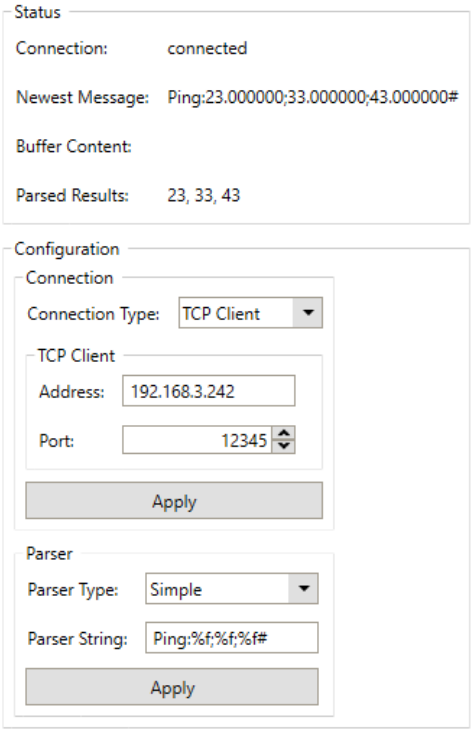
3.3 Program Functions

The Socket App adds the following functions to the program editor:

- Check Connection – Sets a variable to 1 if a connection is established or to 0 otherwise
- Get Result – Copies the last received values into variables
- Send Message – Sends a message to the communication target. The message may contain values from variables.

4. Configuration

After installing and enabling the app you must set up the connection and the parser. To do this open the Socket App tab in the main view of iRC (go to the 3D view, then click “Socket App” at the very top, next to the “File” menu button).



4.1 Status

The Status box shows the current state of the app:

- The “Connection” status will tell whether the app is connected or the error message that occurred last.
- “Newest Message” contains the message that was received last, it may be a fragment of a longer message.

- The “Buffer Content” shows the received data that is not parsed yet. If the parser did not find the defined message further incoming messages are appended (max 1024 bytes). This will show nothing if the parser has successfully parsed a message at the end of the buffer.
- “Parsed Results” lists the number values that the parser has found. The “Get Result” command will copy these into program variables.

4.2 Connection Set-Up

To set up the connection select the connection type and enter the parameters:

- TCP Client
  - Target IP or hostname
  - Target port number
- TCP Server
  - Local port number
- UDP
  - Target IP or hostname
  - Target port number
  - Local port number

Click “Apply” to save and apply the changes. The Socket App will connect automatically. If the connection is lost it tries to reconnect each ca. 10s.

4.3 Parser

The parser tries to read the received messages and returns the number values it found. The Socket App supports the following parsers:

- Simple
- Regular Expression (Regex)

The simple parser has a very simple syntax similar to the scanf function of the C programming language – you basically type the expected message with placeholders for the numbers. This allows one message type to be parsed.

Regular expressions on the other hand can be very complex and difficult to read and write but are very powerful. For example, they allow alternatives which you may be able to use to read different message types (however, you will need to distinguish them by the received number values).

Select the parser that you would like to use, enter the parser string (message format), then click “Apply”.

4.3.1 Simple Parser

The simple parser uses a scanf-like syntax, but misses some features like parsing strings, character count and scansets.

To create your parser string write the message to be parsed and replace all changing numbers with placeholders of the following format:

Placeholder	Explanation	Examples
%%	Matches the '%' sign	%
%i	Matches a signed decimal integer, base 8, 10 or 16	4, +23, -123, 0123, 0x4f, -0x4F
%d	Matches a signed decimal integer, base 10	4, +23, -123
%u	Matches an unsigned decimal integer, base 10	4, 23, 123
%f, %e, %g	Matches a floating point number	1.234, -1.234, 1.234e-3

Each matched number is added to the result list. To match a number without adding it to the result (e.g. irrelevant values) add a '*' after the '%', e.g. "%*f".

Examples:
"%f;%f;%f#" matches the message "1.1;2.2;3.3#", returning the values 1.1, 2.2 and 3.3.

"%f;%*f;%f#" ignores the second value, returning the values 1.1 and 3.3.

"0000startPASS#0#0#%d#%d#%d#%f#0stop" parses the results of the ifm O2D200 cameras, e.g. "0000startPASS#0#0#0#427#332#182.71#0stop" is parsed for the values 0, 427, 332 and 182.71

4.3.2 Regular Expressions

With regular expressions you get complete control over the message format.

The parser string must be in ECMAScript regex syntax, as described by the following articles:

- https://cplusplus.com/reference/regex/ECMAScript/
- https://en.cppreference.com/w/cpp/regex/ecmascript

All captured groups are parsed for number values, so make sure to make all non-number groups non-capturing. If parsing a group for a number fails only the successfully parsed values up to that point are returned (while groups are wrapped in '(' and ')' non-capturing groups are wrapped in '(?:' and ')'). The C function *strtod* is used to parse the captured groups for numbers.

Examples:

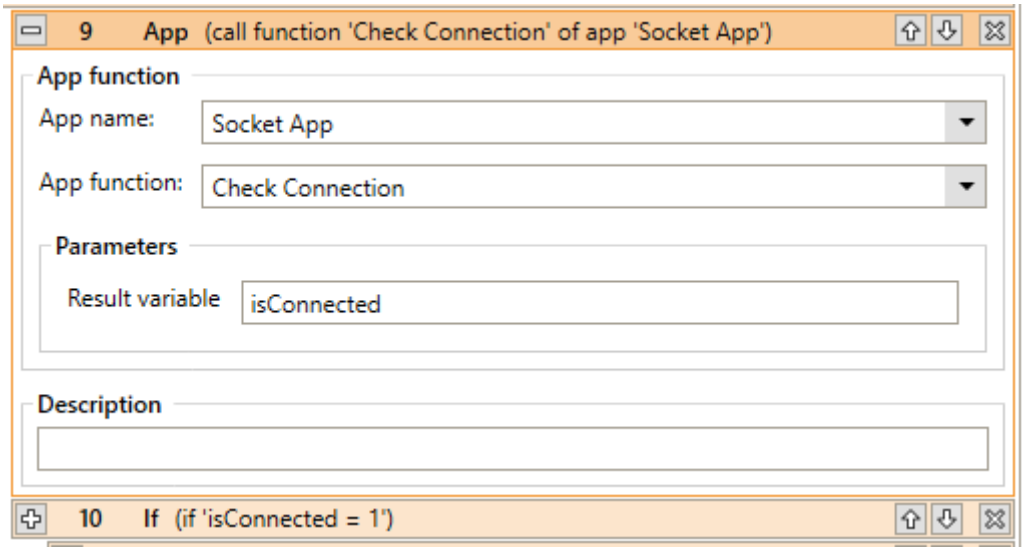
"[0a-zA-Z]+#0#0#([01])#([+-]?\d+)#([+-]?\d+)#([+-]?\d+(?:\.\d+)?)#0stop" parses results of the ifm O2D200 cameras (see example for the simple parser).

5. Integration in robot programs

The Socket App adds the following commands to the program editor. Add them via the "App Function" command in the "Special" menu, then select the app function.

- Get Result – Writes the received values into program variables
- Send Message – Sends a message to the communication target

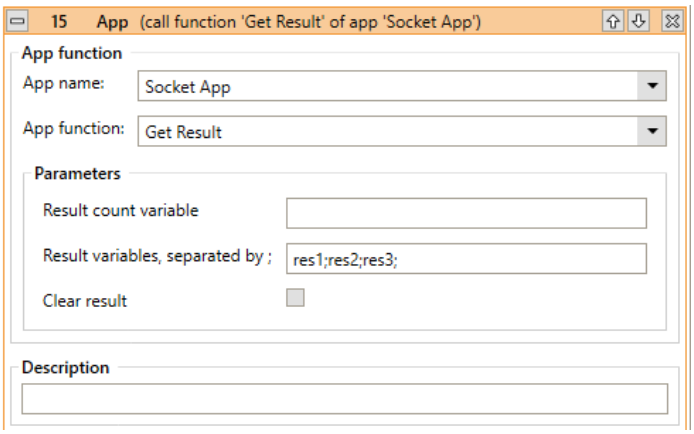
5.1 Check Connection



Check connection sets a variable value to 1 if a connection is established (i.e. messages can be sent out). Otherwise it sets the variable to 0.

Check the connection before sending messages, if the connection is lost the robot program will stop with an error message. Checking the connection before reading the result is not necessary.

5.2 Get Result



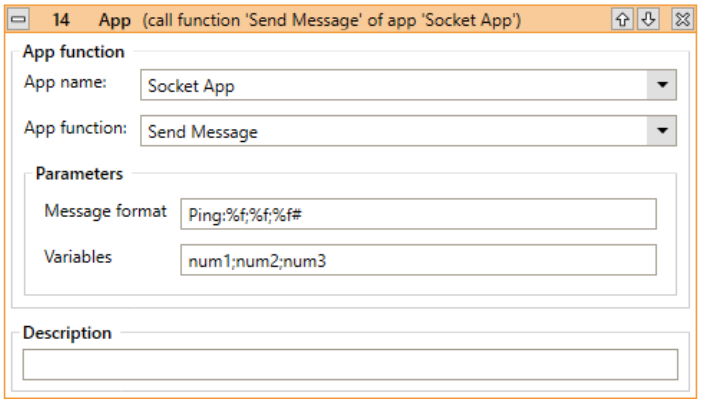
The app function “Get Result” copies the received values into program variables. It has the following parameters:

- Result count variable: If a variable name is given the number of successfully parsed values is written here
- Result variables: Enter a list of variables (number variables or position variable components), separated only by ‘;’. The parsed values are written to these variables.
- Clear result: If checked the result buffer is cleared so that a following call of Get Result will only return values if a new message was received.

Get Result does not wait until a message is received. If no values are available the result count will be 0. Check the result count to make sure a message was received.

If more values are received than variables are given the additional values are ignored.

5.3 Send Message



The app function “Send Message” has the following parameters:

- Message format: Type the message to be sent. You may use printf-style formatted placeholders (similar to those of the simple parser) that will be replaced with variable values
- Variables: List of variables (separated by ‘;’) to replace the placeholders with.

Note that this function fails if the communication target is not available (e.g. not connected). This will abort the robot program with an error message.

The following article explains the message format syntax:
https://cplusplus.com/reference/cstdio/printf/

The Socket App supports the following format specifiers, any other specifier is ignored:

- Integers: %c, %d, %i
- Unsigned integers: %o, %u, %x, %X
- Floating point numbers: %a, %A, %e, %E, %f, %F, %g, %G
- %% prints a %

You may specify flags, width, and precision but no data type length (this is chosen automatically).

Examples:

The message format “%f;%f;%f#” with variable values 1.1, 2.2 and 3.3 creates the message “1.1;2.2;3.3#”.

6. Examples

This section contains setup examples.

6.1 Synchronizing Robots

The Socket App can be used to send values between robots. For example, one robot could tell the other what state it currently is in or share a position value received from a camera.

Note: This has not been tested in a production environment. Please do a reliability test if you consider using the Socket App to send messages to other robots.

You can connect two robots either via TCP or UDP. Since TCP is connection based it is easier for the app to recognized connection loss. UDP does not handle connections, so you could couple multiple robots with only one instance of the Socket App (this has not been tested, you may miss messages).

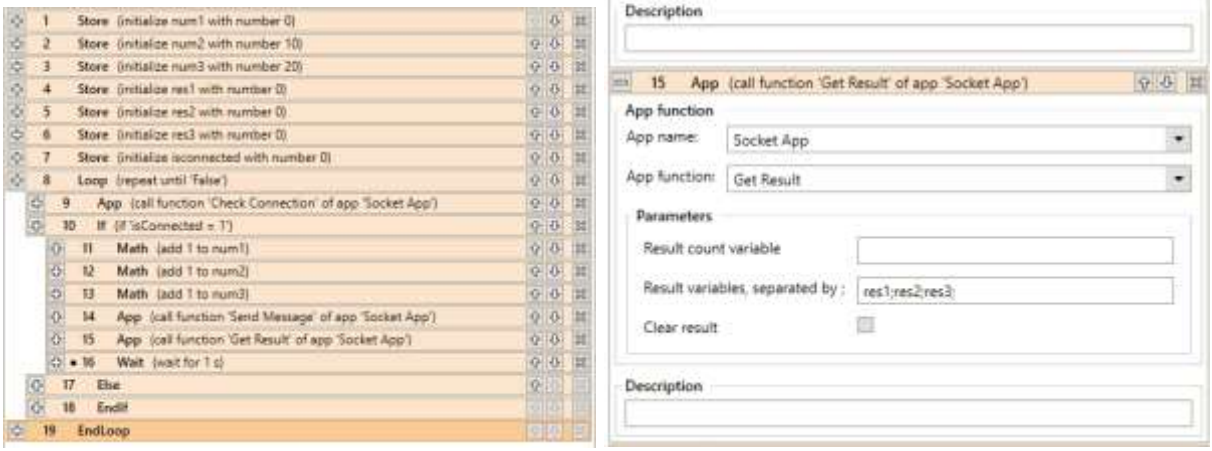
On one robot select connection type “TCP Server”, on the other “TCP Client”. Enter the IP address of the server at the client. Choose a port number, e.g. 12345 for the server and also enter it for the client (note: some port numbers are reserved, choose one between 1024 – 49151).

Alternatively, if you decide to use UDP, set the connection type to “UDP” on both robots, enter the IP address of the other robot in “Remote Address” and select two ports. The local port of one robot will be the remote port of the other and vice-versa.

Select the Simple parser and define the message format in the parser string box – this is up to you but a good choice is to concatenate numbers with a separator symbol and a different symbol for the end of the message as shown in the following example (‘%f’ is the placeholder for a number, ‘;’ is the number separator and ‘#’ is the end of the message). You may add as many numbers as you need.

%f;%f;%f#

The following screenshots show an example program that transmits and receives three number values (note that the numbers are just counting up in lines 11-13, set them to the values you need to transmit). In an infinite loop the program checks whether the connection is okay, then it sends the values to the other robot and copies the received values into variables.



Note the message format in the Send Message function (command 14), it resembles the format set in the parser string.