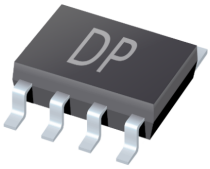
FPC



DevicePlugin

TcpCom

Revision 2020.6.9

Table of Contents

1 Description

General TCP and HTTP (POST/GET) communication client.
Allows regular-expression and XML-based parsing of response.

2 Commands

2.1 *idn? (Identification)

```
*idn?
```

Gets the plug-in identification string.

Parameters

No parameters.

Return value

The identification string in standard format "<company>,<product/name>,<serial-no>,<version>".

2.2 TCP

2.2.1 open (Open connection)

```
open:<host>:<port>{:timeout=<timeout>}
```

Open TCP connection to specified host and port.
If is connection lost, then is needed call close before next connection attempt!

Parameters

host	[string]	Target host IP address, host-name or URL.
port	[int]	Port to be used, 0 to 65535.
timeout	[int]	Open connection and read timeout in range 100 to 10000 milliseconds. Default: 2000

Return value

No return value

Examples

```
open:192.168.1.100:5000
```
Open connection to specified IP address, using port 5000.

```
open:"http://www.fpc.cz":80
```
Open connection to specified URL in the internet, using port 80.

2.2.2 close (Close connection)

```
close
```

Close currently opened TCP connection.

Parameters

No parameters

Return value

No return value

2.2.3 write (Write data)

```
write: <data>
```

Send data over currently opened TCP connection.

Parameters

`data`	[string]	String to be sent over TCP. Use escape-sequences to send any (even non-printable) characters.

Return value

No return value

Examples

```
write: "hello\n"
```
Send 6 bytes: "hello" + LF ("\n") character.

2.2.4 read (Read data)

```
read: <type>: <read>
```

Read specified number of bytes or wait for specified ending string. Method blocks execution until data are received.
When Asynchronous operation is active, this function will be disabled.

Parameters

`type`	[enum]	Read type: • `c` - count, wait for specified number of received bytes • `n` - ending-string, wait for specified ending string
`read`	[int] or [string]	If read-type is: • `c` = [int], 1 to N, number of bytes to receive • `n` = [string], ending-string to wait for

Return value

Received string.

Examples

```
read: c: 10
```
Read 10 bytes

```
read: n: "\n"
```
Read until the "\n" (LF) character is received.

2.2.5 rw (Read/write)

```
rw: <read-type>: <read>: <write>
```

Performs read/write at once (request-answer).
When Asynchronous operation is active, this function will be disabled.

Parameters

`read-type`	[string]	Read type: • `c` - count, wait for specified number of received bytes • `n` - ending-string, wait for specified ending string
`read`	[int] or [string]	If read-type is: • `c` = [int], 1 to N, number of bytes to receive • `n` = [string], ending-string to wait for
`write`	[string]	String to be sent over TCP. Use escape-sequences to send any (even non-printable) characters.

Return value

Received string.

Examples

```
rw: n: "\n": "query?\n"
```
Writes "query?\n" and waits until response with ending-string "\n" is received.

2.2.6 Asynchronous operation commands

After activating this mode standard read commands no longer can be used. It must be used async:count? and async:read?.
The reading will be done asynchronously and the data will be written into internal buffer.
The writing will be still done synchronously.
Mode is deactivated by a connection loss (after automatic connection retries limit) or by close (Close connection).

2.2.6.1 async:start (Start async reading)

```
async: start: <delimiter>: {conRetries=[int]}
```

Load text into internal buffer. On this buffer operate commands template:replace, template:write, template:rw.

Parameters

`delimiter`	[string]	Delimiter used for asynchronous reading of stream to split data into "item". Every item will be stored into internal buffer.
`conRetries`	[int]	Automatic reconnection retry limit. When asynchronous operation is running and connection was lost, then it automatically reconnect to previous host. This value is the limit for retry count. Zero value disable auto-reconnect. **Default:** 5

Return value

No return value

Examples

```
async:start:"\r\n"
```
Every text delimited by "\r\n" will be stored into internal buffer as line.

2.2.6.2 async:count? (Internal buffer items count)

```
async:count?
```

Return count of currently received items from internal buffer.

Parameters

No params

Return value

Count of buffer items.

2.2.6.3 async:read? (Return and remove first value)

```
async:read?
```

Read first item and remove it from buffer.

Parameters

No params

Return value

First value from buffer.
When buffer is empty, exception will be returned.

2.2.6.4 async:clear (Clear internal buffer)

```
async:clear
```

Remove all received item from internal buffer.

Parameters

No params

Return value

No return value

2.2.7 Template Commands

'Template' prefix can be shorten to 'tpl' only.

2.2.7.1 template:load (Load template buffer)

```
template:load:{text=[string]|file=[string]}{:encoding=[int]}
```

Load text into internal buffer. On this buffer operate commands template:replace, template:write, template:rw.

Parameters

`text -or- file`	[string]	Load text from command value or file. Note 1: Separator '\' in file path must be written as '\\' or can be used Unix style '/'. Note 2: File size which can be loaded is internally limited to 16M.
`encoding`	[int]	Change encoding code page. Applies only if is file as source. **Default:** determined by Windows default.

Return value

No return value

Examples

`template:load:text="$Product$ was tested by $User$"`
Loads text into internal buffer.

`template:load:file=Report.txt:encoding=1250`
Loads all text from file into internal buffer. The file encoding will be treated as ANSI Central European.

2.2.7.2 template:replace (Add variable for replace)

`template:replace:{?var0=[string]:?varN=[string] | [var0:[string]]:[varN:[string]]}`

The var=value pair is used, for replace any 'var' in template buffer surrounded by $ with string value.

Parameters

`varN | ?varN`	[string]	Variable(s)=value pair to create or change.

Return value

No return value

Examples

For this example must be loaded template buffer at first.
`template:load:text="Tester name is $User$."`

`template:replace:?User=John`
It will replace the text in buffer from "$User$" to "John".

2.2.7.3 template:clear (Removes all variables)

`template:clear`

Removes all variables added by replace command.

Parameters

No parameters

Return value

No return value

2.2.7.4 template:write (Write buffered data)

`template:write:<data>{:encoding=[int]}`

Send template buffered data over currently opened TCP connection.

Parameters

`encoding`	[int]	Write template buffer as string encoded by setted code page. Read is not affected by this setting. **Default:** determined by Windows default.

Return value

No return value

2.2.7.5 template:rw (Read/write buffered data)

```
rw: <read-type>: <read>{: encoding=[ int]}
```

Performs read/write at once (request-answer). As a data source is used template buffer.
When Asynchronous operation is active, this function will be disabled.

Parameters

`read-type`	[string]	Read type: • `c` - count, wait for specified number of received bytes • `n` - ending-string, wait for specified ending string
`read`	[int] or [string]	If read-type is: • `c` = [int], 1 to N, number of bytes to receive • `n` = [string], ending-string to wait for
`encoding`	[int]	Write template buffer as string encoded by setted code page. Read is not affected by this setting. **Default:** determined by Windows default.

Return value

Received string.

Examples

`template: rw: n: "\n"`
Writes template buffer and waits until response with ending-string "\n" is received.

2.3 HTTP

2.3.1 post (POST request)

```
post: <url>: <arguments>{: ContentType=[ string]}
```

Open connection to HTTP server at URL, send a POST request with arguments, receive response and close the connection.

Parameters

`url`	[string]	Target URL of HTTP server, e.g. "http://example.com:1080/post.php"
`arguments`	[string]	String arguments to send to HTTP server.
`ContentType`	[string]	Specify Content-Type of arguments to HTTP server. **Default:** application/x-www-form-urlencoded

Return value

Server plain-text response.

Examples

`post:"http://example.com/query":"low=1&high=5"`
Send a POST request to example.com HTTP server with arguments "low" and "high".

2.3.2 get (GET request)

```
get:<url>
```

Open connection to HTTP server at URL, send a GET request with arguments, receive response and close the connection.

Parameters

`url`	[string]	Target URL of HTTP server, e.g. "http://example.com:1080/export.xml"

Return value

Server plain-text response.

Examples

`post:"http://example.com/export?format=xml"`
Send a GET request to example.com HTTP server and receive data.

2.3.3 resp? (Work with response)

```
resp?:regex:<pattern>
```

The resp? method simplifies the work with structured response of HTTP server. You can cut the part of response text using the regular expression.

Parameters

`pattern`	[string]	Regex pattern to find a match in last response text.

Return value

Text of matched regex group.

Examples

Response for the example below:
Variant=1
CanTest=fase
Message="message to tester"

`resp?:regex:"Variant=([0-9]+)"`
Try to find a text "Variant=" and a valid integer number: returns the value of first matched group - in this case the return value will be "1".

`resp?:regex:"Message=\"(.*)\""`
Try to find a text of "Message=": returns "message to tester" text

```
resp?:xml:<element-tree>{[<element-index>]}{:<attribute-name>}
```


User a XML parser for response of HTTP server.

Parameters

`element-tree`	[string]	Tree to get value of specified element. The separator of subtrees is a slash "/".
`element-index`	[int] or ?	Index of element to get (if there is more elements with same name at the same level) or question-mark (?) to get the count. **Default:** none, first element is selected
`attribute-name`	[string]	Optional name of argument of specified element to get a value. **Default:** none, use element's value

Return value

Element's or attribute's value.

Examples

```
<?xml version="1.0"?>
<document>
   <var>123</var>
   <date>2020-06-08</date>
   <meas>
         <name>Current</name>
         <value unit="mA">125.9</value>
   </meas>
   <meas>
         <name>Voltage</name>
         <value unit="V">24.1</value>
   </meas>
</document>
```

The parser requires the root element - <document> in this case.

`resp?:xml:date`
Read value of <date> element (inner-text). Returns "2020-06-08".

`resp?:xml:meas/value`
Read value of "value" subtree of first "meas" element: 125.9

`resp?:xml:meas[1]/value:unit`
Read "unit" attribute of value element of the indexed (second in this case) meas element: "V".

`resp?:xml:meas[?]`
Read count of meas elements at the same level - returns "2" in this case.

