
DebugServer-js Documentation

Release 0.1.0

webbcam

Apr 25, 2019

Contents

1	Getting Started	3
1.1	Prerequisites	3
1.2	Install	3
1.3	Configure	3
1.4	Launch	3
1.5	Connecting	4
1.6	Sending Commands	4
2	API Reference	5
2.1	Introduction	5
2.2	Server Commands	6
2.3	Session Commands	14
3	License	23
4	Disclaimer	25

DebugServer-js is a Javascript server that provides a simple API for interacting with [Code Composer Studio's Debug Server](#) over TCP/IP. It is based off of the [Test Server](#) example provided with Code Composer Studio.

Simply launch the server and connect to the specified port. You can then send commands to the socket to control the DebugServer. Please see the API for formatting your commands.

CHAPTER 1

Getting Started

1.1 Prerequisites

You will need to have [Code Composer Studio](#) installed along with drivers for any devices you plan to use (offered during installation of CCS or available in CCS's Resource Explorer).

1.2 Install

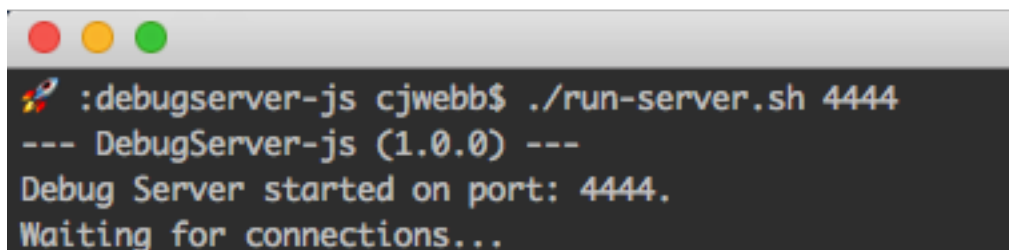
```
git clone https://github.com/webbcam/debugserver-js.git
```

1.3 Configure

Open the file: `run-server.sh` and modify the `CCS_EXE_PATH` variable to reflect your machine and CCS installation location.

1.4 Launch

Open a terminal and run the `run-server.sh` script. (Leave this terminal running in background)

A screenshot of a terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The text in the terminal shows a user prompt, the execution of the script with port 4444, a version banner, and a confirmation message.

```
:debugserver-js cjwebb$ ./run-server.sh 4444
--- DebugServer-js (1.0.0) ---
Debug Server started on port: 4444.
Waiting for connections...
```

1.4.1 Mac OS or Linux

```
./run-server.sh 4444
```

1.4.2 Windows

```
./run-server.bat 4444
```

1.5 Connecting

Once the *DebugServer* is *running*, you can then connect to the Server's socket using any language that supports sockets. Here we show an example in Python:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", 4444))
```

1.6 Sending Commands

Commands can be sent to the Server over the socket in JSON format. Please see the [API](#) for a full list of commands and their format.

```
cmd = json.loads({"name": "ping"})
s.sendall(cmd + "\n")
```


2.1 Introduction

After *connecting to the DebugServer*, you can send commands in the form of *requests*. For every *request* sent the DebugServer will reply with a *response*. The general format of *requests/responses* are shown below:

Note: All *requests* and *responses* are in JSON format.

2.1.1 Requests

Requests are sent to the DebugServer to execute a command. A Request will always have a “name” field.

- The “args” field is a JSON object representing the key-val arguments for the command

Key	Type	Description	Values	Required
“name”	String	Name of command to call		Always
“args”	JSON	Arguments to pass to command		When a command takes an argument(s)

```
// Request
{
  "name": "commandName",
  "args": {
    "arg1": "value",
    "arg2": true
  }
}
```

2.1.2 Responses

Responses are sent back by the DebugServer after completing a Request. A Response will always have a “status” field which will either be “OK” (successful command) or “FAIL” (failed command).

- The “data” field is present when a successful command returns a value.
- The “message” field is present when a command raises an error.

Key	Type	Description	Values	Present
“status”	String	Result of a command	“OK” or “FAIL”	Always
“data”	Command specific	Return value of a command	Depends on command	When a command returns a value
“message”	String	Error message of failed command		When a command returns an error

```
// Success Response
{
  "status": "OK",
  "data": <command return value>
}

// Failure Response
{
  "status": "FAIL",
  "message": "Error message description"
}
```

2.2 Server Commands

Server commands are sent over a socket connected to the Debug Server port (received or specified when *launching the debug server*)

Command List:

- *setTimeout*
- *getTimeout*
- *setConsoleLevel*
- *setConfig*
- *getConfig*
- *createConfig*
- *getListOfCPUs*
- *getListOfDevices*
- *getListOfConnections*
- *getListOfConfigurations*

- *openSession*
- *getListOfSessions*
- *terminateSession*
- *attachCCS*
- *killServer*

2.2.1 setTimeout

Sets the timeout (how long to wait for a DS command to complete)

Request		
Key	Value	Description
"name"	"setTimeout"	-
"args"	"timeout"	Value to set timeout to

```
// Request
{
  "name": "setTimeout",
  "args": {
    "timeout": 300
  }
}
```

2.2.2 getTimeout

Gets the timeout (how long to wait for a DS command to complete)

Request		
Key	Value	Description
"name"	"getTimeout"	-
"args"	-	-

```
// Request
{
  "name": "getTimeout",
}

// Response
{
  "status": "OK",
  "data": -1,
}
```

2.2.3 setConsoleLevel

Sets the console output level

Request		
Key	Value	Description
"name"	"setConsoleLevel"	-
"args"	"level"	Log level to set console output: <ul style="list-style-type: none">• "ALL"• "CONFIG"• "FINE"• "FINER"• "FINEST"• "INFO"• "SEVERE"• "WARNING"• "OFF"

```
// Request
{
  "name": "setConsoleLevel",
  "args": {
    "level": "ALL"
  }
}
```

2.2.4 setConfig

Sets the ccxml file for the DebugServer to use

Request		
Key	Value	Description
"name"	"setConfig"	-
"args"	"path"	Full path to .ccxml file to use

```
// Request
{
  "name": "setConfig",
  "args": {
    "path": "/path/to/config.ccxml"
  }
}
```

2.2.5 getConfig

Returns the ccxml file the DebugServer is using

Request		
Key	Value	Description
"name"	"getConfig"	-
"args"	-	-

```
// Request
{
  "name": "getConfig",
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	String	Path to ccxml file

```
// Response
{
  "status": "OK",
  "data": "/path/to/config.ccxml"
}
```

2.2.6 createConfig

Creates a ccxml configuration file

Request		
Key	Value	Description
"name"	"createConfig"	-
"args"	"name"	Name of ccxml file to create
	"connection"	Connection name to use
	"device"	Devicetype to use (optional)
	"board"	Board to use (optional)
	"directory"	Directory to place ccxml (optional)

```
// Request
{
  "name": "createConfig",
  "args": {
    "name": "config.ccxml",
    "connection": "Texas Instruments XDS110 USB Debug Probe",
    "device": "CC1350F128",
    "directory": "/home/user/ti/CCSTargetConfigurations"
  }
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	JSON object	JSON object containing name of ccxml file and directory

```
// Response
{
  "status": "OK",
  "data": {
```

(continues on next page)

(continued from previous page)

```
    "name": "config.ccxml",
    "directory": "/home/user/ti/CCSTargetConfigurations"
  }
}
```

2.2.7 getListOfCPUs

Returns a list of CPU names which can be used for starting a session.

Request		
Key	Value	Description
"name"	"getListOfCPUs"	-
"args"	-	-

```
// Request
{
  "name": "getListOfCPUs",
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Array	List of CPU names

```
// Response
{
  "status": "OK",
  "data": ["Cortex_M3", "Cortex_M0"]
}
```

2.2.8 getListOfDevices

Returns a list of device names which can be used for creating ccxml files.

Request		
Key	Value	Description
"name"	"getListOfDevices"	-
"args"	-	-

```
// Request
{
  "name": "getListOfDevices"
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Array	List of Device names

```
// Response
{
  "status": "OK",
  "data": ["CC1310F128", "CC1350F128"]
}
```

2.2.9 getListOfConnections

Returns a list of connection names which can be used for creating ccxml files.

Request		
Key	Value	Description
"name"	"getListOfConnections"	-
"args"	-	-

```
// Request
{
  "name": "getListOfConnections"
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Array	List of Connection names

```
// Response
{
  "status": "OK",
  "data": ["Texas Instruments XDS110 USB Debug Probe", "TI MSP430 USB1"]
}
```

2.2.10 getListOfConfigurations

Returns a list of configuration (ccxml) file names.

Request		
Key	Value	Description
"name"	"getListOfConfigurations"	-
"args"	-	-

```
// Request
{
  "name": "getListOfConfigurations",
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Array	List of configuration files

```
// Response
{
  "status": "OK",
  "data": ["L2000FF.ccxml", "L4000XX.ccxml"]
}
```

2.2.11 openSession

Opens a session for the given CPU

Request		
Key	Value	Description
"name"	"openSession"	-
"args"	"name"	CPU to open session with

```
// Request
{
  "name": "openSession",
  "args": {
    "name": "*/Cortex_M3*"
  }
}
```

Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Integer	Port number session opened on

```
// Response
{
  "status": "OK",
  "data": 4444
}
```

2.2.12 getListOfSessions

Returns a list of open sessions running on the DebugServer

Request		
Key	Value	Description
"name"	"getListOfSessions"	-
"args"	-	-

```
// Request
{
  "name": "getListOfSessions"
}
```


Response		
Key	Value	Description
"status"	String	"OK" or "FAIL"
"data"	Array of JSON objects	List of JSON objects containing open session names and their port number

```
// Response
{
  "status": "OK",
  "data": [{
    "name": "Texas Instruments XDS110 USB Debug Probe/Cortex_M3",
    "port": 4445
  }, {
    "name": "Texas Instruments XDS110 USB Debug Probe/IcePick_M0",
    "port": 4446
  }]
}
```

2.2.13 terminateSession

Terminates the specified session.

Warning: The session's socket should be closed by the client before terminating the session to avoid deadlock.

Request		
Key	Value	Description
"name"	"terminateSession"	-
"args"	"name"	Name of session to terminate

```
// Request
{
  "name": "terminateSession",
  "args": {
    "name": "Texas Instruments XDS110 USB Debug Probe/Cortex_M3"
  }
}
```

2.2.14 attachCCS

Open and attach a CCS workbench to the Debug Session

Caution: You can only attach a CCS workbench once per Debug Server. If you exit out of the CCS IDE, you'll have to restart the Debug Server in order to attach again.

Request		
Key	Value	Description
"name"	"attachCCS"	-
"args"	-	-

```
// Request
{
  "name": "attachCCS"
}
```

2.2.15 killServer

Terminates all open sessions and shuts the Debug Server down.

Warning: All open session sockets should be closed before killing the server to avoid deadlock.

Request		
Key	Value	Description
"name"	"killServer"	-
"args"	-	-

```
// Request
{
  "name": "killServer"
}
```

2.3 Session Commands

Session commands are sent over a socket connected to the respective session port (received when *opening a session*)

Important: You must first *open a session* on the DebugServer and connect to it over a socket before sending any session commands.

Command List:

- *connect*
- *disconnect*
- *erase*
- *reset*
- *load*
- *verify*
- *evaluate*
- *readData*
- *writeData*

- *readRegister*
- *writeRegister*
- *getOption*
- *setOption*
- *performOperation*
- *run*
- *halt*
- *stop*

2.3.1 connect

Connect to the target

Request		
Key	Value	Description
"name"	"connect"	-
"args"	-	-

```
// Request
{
  "name": "connect"
}
```

2.3.2 disconnect

Disconnect from the target

Request		
Key	Value	Description
"name"	"disconnect"	-
"args"	-	-

```
// Request
{
  "name": "disconnect"
}
```

2.3.3 erase

Erases flash on target (must be *connected* to device)

Request		
Key	Value	Description
"name"	"erase"	-
"args"	-	-

```
// Request
{
  "name": "erase"
}
```

2.3.4 reset

Resets device (must be *connected* to device)

Request		
Key	Value	Description
"name"	"reset"	-
"args"	-	-

```
// Request
{
  "name": "reset"
}
```

2.3.5 load

Loads file into device's flash (must be *connected* to device)

Request		
Key	Value	Description
"name"	"load"	-
"args"	"file"	Path to file to load
	"binary"	Load image as binary (optional; default=false)
	"address"	Address location to load binary image (optional)

```
// Request
{
  "name": "load",
  "args": {
    "file": "/path/to/image.hex"
  }
}

// Request (binary)
{
  "name": "load",
  "args": {
    "file": "/path/to/image.bin",
    "binary": true,

```

(continues on next page)

(continued from previous page)

```

    "address": 0x10000000
  }
}

```

2.3.6 verify

Verifies a file in device's memory (must be *connected* to device)

Request		
Key	Value	Description
"name"	"verify"	-
"args"	"file"	Path to file to verify
	"binary"	Verify image as binary (optional; default=false)
	"address"	Address location to verify binary image (optional)

```

// Request
{
  "name": "verify",
  "args": {
    "file": "/path/to/image.hex"
  }
}

// Request (binary)
{
  "name": "verify",
  "args": {
    "file": "/path/to/image.bin",
    "binary": true,
    "address": 0x10000000
  }
}

```

2.3.7 evaluate

Evaluates an expression (must be *connected* to device)

Request		
Key	Value	Description
"name"	"evaluate"	-
"args"	"expression"	Expression to evaluate
	"file"	Path to symbols (.out) file to load first (optional)

```

// Request (with symbols)
{
  "name": "evaluate",
  "args": {
    "expression": "&Sensor_msgStats",
    "file": "/path/to/symbols.out",
  }
}

```

(continues on next page)

(continued from previous page)

```
}

// Response
{
  "status": "OK",
  "data": 51234234
}
```

2.3.8 readData

Read memory from device (must be *connected* to device)

Request		
Key	Value	Description
"name"	"readData"	-
"args"	"page"	Page number to read address from
	"address"	Address to read memory from
	"numBytes"	Number of bytes to read starting at 'address'

```
// Request
{
  "name": "readData",
  "args": {
    "page": 0,
    "address": 0x20000000,
    "numBytes": 4
  }
}

// Response
{
  "status": "OK",
  "data": [0xFF, 0xFF, 0xFF, 0xFF]
}
```

2.3.9 writeData

Write to memory on device (must be *connected* to device)

Request		
Key	Value	Description
"name"	"writeData"	-
"args"	"page"	Page number of address to write to
	"address"	Memory address to write to
	"data"	Byte or bytes to write to memory at 'address'

```
// Request
{
  "name": "writeData",
  "args": {
    "page": 0,
    "address": 0x20000000,
    "data": [0xFF, 0xFF]
  }
}

// Response
{
  "status": "OK"
}
```

2.3.10 readRegister

Read device register (must be *connected* to device)

Request		
Key	Value	Description
"name"	"readRegister"	-
"args"	"name"	Name of register to read from

```
// Request
{
  "name": "readRegister",
  "args": {
    "name": "R1"
  }
}

// Response
{
  "status": "OK",
  "data": 0xFFFF
}
```

2.3.11 writeRegister

Write to device's register (must be *connected* to device)

Request		
Key	Value	Description
"name"	"writeRegister"	-
"args"	"name"	Name of register to write to
	"value"	Value to write to register

```
// Request
{
  "name": "writeRegister",
  "args": {
    "name": "R1",
    "value": 0xBEEF
  }
}

// Response
{
  "status": "OK"
}
```

2.3.12 getOption

Get the value of a device option (must be *connected* to device)

Request		
Key	Value	Description
"name"	"getOption"	-
"args"	"id"	option ID

```
// Request
{
  "name": "getOption",
  "args": {
    "id": "DeviceInfoRevision"
  }
}

// Response
{
  "status": "OK",
  "data": "2.1"
}
```

2.3.13 setOption

Set the value of a device option (must be *connected* to device)

Request		
Key	Value	Description
"name"	"setOption"	-
"args"	"id"	option ID
	"value"	Value to set option to


```
// Request
{
  "name": "setOption",
  "args": {
    "id": "ResetOnRestart",
    "value": True
  }
}

// Response
{
  "status": "OK"
}
```

2.3.14 performOperation

Perform flash operation (must be *connected* to device)

Request		
Key	Value	Description
"name"	"performOperation"	-
"args"	"opcode"	operation code for flash operation

```
// Request
{
  "name": "performOperation",
  "args": {
    "opcode": "Erase",
  }
}

// Response
{
  "status": "OK"
}
```

2.3.15 run

Issue run command to target device

Request		
Key	Value	Description
"name"	"run"	-
"args"	"asynchronous"	run and return control immediately (default=False)

Warning: If "async" is set to **False**, this function will not return until one of the following occur:

- target is halted due to hitting a breakpoint

- target hits end of program
- timeout

```
// Request
{
  "name": "run",
  "args": {
    "async": true
  }
}
```

2.3.16 halt

Issue halt command to target device

Request		
Key	Value	Description
"name"	"halt"	-
"args"	"wait"	wait until device is actually halted before returning

```
// Request
{
  "name": "halt",
  "args": {
    "wait": true
  }
}
```

2.3.17 stop

Stop the session thread (does not *terminate session*)

Request		
Key	Value	Description
"name"	"stop"	-
"args"	-	-

```
// Request
{
  "name": "stop"
}
```

CHAPTER 3

License

DebugServer-js is released under the MIT license:

```
Copyright (c) 2019 Cameron Webb

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
↪furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
↪all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```


CHAPTER 4

Disclaimer

This project **is** NOT supported by (nor affiliated **with**) Texas Instruments Inc.
Code Composer Studio **is** a trademark of Texas Instruments Inc.
Any **and all** other trademarks are the **property** of their respective owners.