# Protocol for Data Exchange with OpenDSS-G - DSTCP

*Davis Montenegro*
*Last update: 09/16/2019*

OpenDSS-G is the graphical interface for EPRI's Open Source Distribution System Simulator, OpenDSS. Because OpenDSS has been evolving into a parallel processing software (OpenDSS V8), the need to develop a better interface for assessing future smart grid requirements for planning and operation studies was identified. OpenDSS-G is a comprehensive interface for facilitating the use of the advanced features of OpenDSS, providing a development environment that mimics a control room surrounded by graphical tools that can be intuitively interpreted by users.

The OpenDSS-G architecture is developed using the actor model for handling parallelism and coordinating multitasking. In the actor model, as in object oriented programming, everything is an actor. However, in this case, each actor is a program executed in a separate thread that is executed concurrently with other actors. In order to coordinate and guarantee consistency, these actors communicate with each other by sending messages. Therefore, with OpenDSS-G, it is possible to start a simulation and at the same time request the simulation progress and have access to the variables inside the simulation without breaking the simulation loop, thus avoiding the unwanted overhead observed in traditional co-simulation.

OpenDSS-G features four main components or actors: The OpenDSS manager, graphical user interface (GUI), database server, and Transmission Control Protocol (TCP) server. These actors interact by sending messages to each other, creating and destroying actors when needed to handle data, launching simulations, exchanging data, and handling the user actions, among other functionalities. The communication between actors is handled with queues to avoid data losses when several actors send data to a single actor.

Each OpenDSS process is handled and centralized by the OpenDSS actor. If an actor needs to interact with OpenDSS, it will have to send a message to the OpenDSS actor. This process maintains the simulation consistency with all the actors around OpenDSS through the TCP server, including those outside of OpenDSS-G such as co-simulation platforms and external code.

Additional functionalities were added to the TCP server in OpenDSS-G in order to provide access from external programs to the functionalities of OpenDSS. This document describes the protocol structure and functionalities around the TCP server included with OpenDSS-G. This protocol is called DSTCP.

## The DSTCP protocol

To connect to the OpenDSS-G using an external application, if you are in the local computer the IP address is 127.0.0.1 (localhost) and the port is 6345; if you are in a different computer the IP address will be the one of the computer hosting OpenDSS-G. The port number may increase depending on the number of TCP actors configured in OpenDSS-G, however, the initial port number will always be 6345.

DSTCP is an ASCII protocol to allow an easy interface for data exchange. The basic structure of the protocol is as follows:

| FF; | LLLL MMMM… | ; |
|---|---|---|
| Function, the function is composed by two characters and defines the type of data to be handled. After the function a semicolon character (;) needs to be added | The first four characters (LLLL) defines the length of the message content in the MMMM… characters. The length does not include the termination char. The length of the message is sent in hex (ASCII). | Termination char, is a character semicolon (;) |

Using the correct function, the remote client can test his control and monitoring algorithms integrated with the model simulated in OpenDSS-G, as well as be able to modify the graphical objects, display/hide model features, among other activities. Using this protocol, the user will be able to control OpenDSS-G entirely.

New functions are continuously added to cover more needs, the functions included in this document are the ones available at this point.

## Function 01: Request simulation time

With this function, the remote client can read the simulation time in seconds.

From the client to the TCP server:

01;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message.

## Function 02: Request the simulation Days

With this function, the remote client can read the simulation time in days.

From the client to the TCP server:

02;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message.

## Function 03: Request the nodes' Voltages (p.u.)

With this function, the remote client can read the voltages (p.u.) for the nodes within the circuit after a power flow solution.

From the client to the TCP server:

03;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each voltage comes in pu in a TAB separated array. The values length may vary depending on the number of nodes at the bus. Each pair represents the value at the node, the list of node names can be requested with function 43. The order in which the names are delivered in function 43 corresponds to the order in which the data is delivered with this function.

## Function 04: Request the nodes' voltages

With this function, the remote client can read the voltages for the nodes in the circuit after a power flow solution.

From the client to the TCP server:

04;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each voltage comes in complex pairs (a + jb) in a TAB separated array. The values length may vary depending on the number of nodes at the bus. Each pair represents the value at the node, the list of node names can be requested with function 43. The order in which the names are delivered in function 43 corresponds to the order in which the data is delivered with this function.

## Function 05: Request the capacitors' currents

With this function, the remote client can read the currents for the capacitors in the circuit after a power flow solution.

From the client to the TCP server:

05;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each current comes in pairs (magnitude and angle) in a TAB separated array. The values length may vary depending on the number of phases of the capacitors. Each pair represents the value at the phase of the capacitor, the list of capacitors can be requested with function 46. The order in which the names are delivered in function 46 corresponds to the order in which the data is delivered with this function.

## Function 06: Request the capacitors' voltages

With this function, the remote client can read the voltages for the capacitors in the circuit after a power flow solution.

From the client to the TCP server:

06;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each voltage comes in complex pairs (a + jb) in a TAB separated array. Each pair represents the value at the phase of the capacitor, the list of capacitors can be requested with function 46. The order in which the names are delivered in function 46 corresponds to the order in which the data is delivered with this function.

## Function 07: Request the capacitors' losses

With this function, the remote client can read the losses for the capacitors in the circuit after a power flow solution.

From the client to the TCP server:

07;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The losses for each capacitor come in complex pairs (a + jb), the real part represents the total losses in W and the imaginary part represents the losses in vars. The data is delivered in a TAB separated array. Each pair represents the losses value per capacitor, the list of capacitors can be requested with function 46. The order in which the names are delivered in function 46 corresponds to the order in which the data is delivered with this function.

## Function 08: Request the capacitors' powers

With this function, the remote client can read the powers for the capacitors in the circuit after a power flow solution.

From the client to the TCP server:

08;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The powers for each capacitor come in complex pairs (a + jb), the real part is the power in kW and the imaginary part is the power in kvars. The data is delivered in a TAB separated array. Each pair represents the value at the phase of the capacitor, the list of capacitors can be requested with function 46. The order in which the names are delivered in function 46 corresponds to the order in which the data is delivered with this function.

## Function 09: Request the transformers' currents

With this function, the remote client can read the currents for the transformers in the circuit after a power flow solution.

From the client to the TCP server:

09;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each current comes in pairs (magnitude and angle). The data is delivered in a TAB separated array. Each pair represents the value at the phase of the transformer at each winding, the list of transformers can be requested with function 45. The order in which the names are delivered in function 45 corresponds to the order in which the data is delivered with this function.

## Function 10: Request the transformers' Voltages

With this function, the remote client can read the voltages for the transformers in the circuit after a power flow solution.

From the client to the TCP server:

10;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each voltage comes in complex pairs (a + jb) in a TAB separated array. Each pair represents the value at the phase and winding of the transformer, the list of transformers can be requested with function 45. The order in which the names are delivered in function 45 corresponds to the order in which the data is delivered with this function.

## Function 11: Request the transformers' losses

With this function, the remote client ca read the losses for the transformers in the circuit after a power flow solution.

From the client to the TCP server:

11;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The losses for each transformer come in complex pairs (a + jb), the real part represents the total losses in W and the imaginary part represents the losses in vars. The data is delivered in a TAB separated array. Each pair represents the losses value per transformer, the list of transformers can be requested with function 45. The order in which the names are delivered in function 45 corresponds to the order in which the data is delivered with this function.

## Function 12: Request the transformers' powers

With this function, the remote client can read the powers for the transformers in the circuit after a power flow solution.

From the client to the TCP server:

12;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The powers for each transformer come in complex pairs (a + jb), the real part is power in kW and the imaginary part is power in kvars. The data is delivered in a TAB separated array. Each pair represents the value at the phase and winding of the transformer, the list of transformers can be requested with function 45. The order in which the names are delivered in function 45 corresponds to the order in which the data is delivered with this function.

## Function 13: Request the loads' currents

With this function, the remote client can read the currents for the loads in the circuit after a power flow solution.

From the client to the TCP server:

13;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each current comes in pairs (magnitude and angle). The data is delivered in a TAB separated array. Each pair represents the value at the phase of the load, the list of loads can be requested with function 44. The order in which the names are delivered in function 44 corresponds to the order in which the data is delivered with this function.

## Function 14: Request the loads' powers

With this function, the remote client can read the powers for the loads in the circuit after a power flow solution.

From the client to the TCP server:

14;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The powers for each load come in complex pairs (a + jb), the real part is the power in kW and the imaginary part is the power in kvars. The data is delivered in a TAB separated array., the list of loads can be requested with function 44. The order in which the names are delivered in function 44 corresponds to the order in which the data is delivered with this function.

## Function 15: Request the lines' currents

With this function, the remote client can read the currents for the lines in the circuit after a power flow solution.

From the client to the TCP server:

15;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. Each current comes in pairs (magnitude and angle). The data is delivered in a TAB separated array. Each pair represents the value at the phase of the line in both terminals, the list of lines can be requested with function 33. The order in which the names are delivered in function 33 corresponds to the order in which the data is delivered with this function.

## Function 16: Request the lines' powers

With this function, the remote client can read the powers for the lines in the circuit after a power flow solution.

From the client to the TCP server:

16;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The powers for each line come in complex pairs (a + jb), the real part is the power in kW and the imaginary part is the power in kvars. The data is delivered in a TAB separated array., the list of lines can be requested with function 33. The order in which the names are delivered in function 33 corresponds to the order in which the data is delivered with this function.

## Function 17: Request the lines' losses

With this function, the remote client can read the losses in the lines within the circuit model after a power flow solution.

From the client to the TCP server:

17;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The losses for each line come in complex pairs (a + jb), the real part represents the total losses in W and the imaginary part represents the losses in vars. The data is delivered in a TAB separated array. Each pair represents the losses value per line, the list of lines can be requested with function 33. The order in which the names are delivered in function 33 corresponds to the order in which the data is delivered with this function.

## Function 18: Write the switch status

With this function, the remote client can update the state of a switch present in the circuit.

From the client to the TCP server:

18; NNNN MMMM…= S ;

Where NNNN is the length of the message in hex (four characters), and MMMM… corresponds to the name of the switch to be updated, and S is the new state. If the new state is a character one (1) it will close the switch, but if the character is a zero (0), the switch will open.

From the TCP server to the client:

LLLLLLLL OK…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters). The list of switches present in the circuit can be obtained with function 19.

### Example

Assume that in your model there is a switch called 672692 and you want to open it, in this case the message to send will be as follows:

18;0006675a=0;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 19: Request the switch status and names

With this function, the remote client can read the state of the switches within the circuit model.

From the client to the TCP server:

19;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The switch status comes in pairs (status and the switch name) in a TAB separated array. The switch status is represented by 1 if the switch is closed or 0 if the switch is open.

### Example

Assume that in your model there is a switch called 672692 and you want to know the state of all the switches in the model, the string to send will be as follows:

19;

If the transaction was successful, OpenDSS-G will return the following message:

0000000B1**CR LF**671692**CRLF**;

Where **CRLF** are two bytes representing the end of line (0xD and 0xA respectively). The end of line represents the beginning of a new row in a TAB separated 2D array. In this structure, row one corresponds to the status of the switches and row 2 are the switch names. Switch name and status should be in the same column.

## Function 1A: Fix Load Power

With this function, the remote client sends to OpenDSS-G the power value (kW) of a specific load. It has been implemented to allow the user to modify dynamically the load behavior within a simulation.

From the client to the TCP server:

1A; NNNN EEEE…-MMMM… ;

Where NNNN is the length of the message in hex (four characters), EEEE… is the name of the load to be updated and MMMM… is the value of the multiplier to affect the nominal value of the load. Both, the name and the value must be separated by the character line (-).

From the TCP server to the client:

LLLLLLLL OK ;

Where *LLLLLLLL* is the length of the message in hex (8 characters).

### Example

Assume that in your model there is a load called 675a and you want to set its kW value to 50 kW, the string sent to OpenDSS-G will be:

1A;0007675a-50;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 1B: Modify Element property

With this function, the remote client can modify an element's property using the name of the element, the property and the value to be modified. It has been implemented to allow the user to modify dynamically the parameters of control object like reclosers, to perform dynamic analyses like reprogramming adaptive protections.

From the client to the TCP server:

1B; NNNN EEEE…-MMMM…-VVVV… ;

Where NNNN is the length of the message in hex (four characters), EEEE… is the name of the object to be updated (it must be specified including the class name, for example, if it is a recloser and its name is RC1, the text to send will be *recloser.RC1*), MMMM… is the value of the property to be changed and VVVV… is the new value for the specified property. All the values must be separated by the character line (-).

From the TCP server to the client:

LLLLLLLL OK ;

Where *LLLLLLLL* is the length of the message in hex (8 characters).

### Example

Assume that in your model there is a load called 675a and you want to set its kW value to 60 kW, the string sent to OpenDSS-G will be:

1B;000Fload.675a-kw-60;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 1C: Request the readings of the energy meters

With this function, the remote client can read the content of the energy meter specified.

From the client to the TCP server:

1C; NNNN EEEE… ;

Where EEEE… is the name of the energy meter to be queried.

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. Values within the returned message are the registers of the energy meter specified. The first set of strings are the headers of the energy meter registries separated by TAB characters. In the next row (rows are separated by **CRLF**) the values for the registries will follow. Each value will match with the header at each column. The values are also separated by TAB.

## Function 1D: Request the branch-to-node incidence matrix

With this function, the remote client can read the branch-to-node incidence matrix of the compiled system.

From the client to the TCP server:

1D;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. The values for the matrix are delivered in compressed format specifying the row, column and value of each non-zero value. The message is delivered as a matrix separated by TAB characters. The row identifier is **CRLF**. Row 0 is the row index, row 1 is the column index and row 2 is the value for the specified cell. Row index, column index and value should match on each column of the TAB separated matrix.

## Function 1E: Request the list of storage elements within the circuit.

With this function, the remote client can read the list of storage elements within the active circuit.

From the client to the TCP server:

1E;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. The message is composed by a TAB separated array containing the list of the storage elements within the circuit.

## Function 1F: Request the storage elements and Electric Vehicles status.

With this function, the remote client can read the storage elements' status (Charging, discharging or idling) as well as some of the state variables for the storage within the circuit.

From the client to the TCP server:

1F;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. The message is a matrix of Four columns with a number of rows equal to the number of elements (EV and storage elements) present within the circuit. The first column is the name of the element and the second one is the amount of energy stored (%), the third one is the amount of energy stored in kWh and the last one is the state of the charger (Charging, discharging or idling). Each element within the row is separated by TAB and each row is separated by a **CRLF**.

## Function 20: Request all switch currents.

With this function, the remote client can read the currents of the switches placed in the circuit.

From the client to the TCP server:

20;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. Each current comes in pairs (magnitude and angle). The data is delivered in a TAB separated matrix. Each pair represents the value at the phase of the switch, first for terminal 1 and then for terminal 2, a switch per row. The list of switches can be requested with function 19. The order in which the names are delivered in function 19 corresponds to the order in which the data is delivered with this function.

## Function 21: Request all switch losses.

With this function, the remote client can read the losses of the switches placed in the circuit.

From the client to the TCP server:

21;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The losses for each switch come in complex pairs (a + jb), the real part represents the total losses in W and the imaginary part represents the losses in vars. The data is delivered in a TAB separated matrix (rows separated by **CRLF**), each row corresponds to a switch. Each pair represents the losses value per switch, the list of switches can be requested with function 19. The order in which the names are delivered in function 19 corresponds to the order in which the data is delivered with this function.

## Function 22: Request all switch powers.

With this function, the remote client can read the powers of the switches placed in the circuit.

From the client to the TCP server:

22;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The powers for each switch come in complex pairs (a + jb), the real part is the power in kW and the imaginary part is the power in kvars. The data is delivered in a TAB separated matrix (rows separated by **CRLF**) each row corresponds to a switch. The list of switches can be requested with function 19. The order in which the names are delivered in function 19 corresponds to the order in which the data is delivered with this function.

### Function 23: Step in command.

With this function the remote client forces the simulation to step in.

From the client to the TCP server:

23;

From the TCP server to the client:

LLLLLLLL OK ;

Where *LLLLLLLL* is the length of the message in hex (8 characters).

### Function 24: End simulation command.

With this function, the remote client forces the simulation to terminate.

From the client to the TCP server:

24;

From the TCP server to the client:

LLLLLLLL OK ;

Where *LLLLLLLL* is the length of the message in hex (8 characters). It is recommended to use this command only when strictly necessary. This command will set the simulation abort flag in OpenDSS, which could generate unexpected issues for the next simulations.

### Function 25: Send OpenDSS command.

With this function, the remote client can interact directly with OpenDSS, it does not matter in which mode the simulation is running.

From the client to the TCP server:

25;CCCCC….;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *CCCCC…* is the string containing the OpenDSS command, *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the returned message.

#### Example

Assume that your simulation was set to run in *time* mode and want to make sure using a remote client, in this case the OpenDSS command to be sent is as follows:

25;0008get mode;

If the transaction was successful, OpenDSS-G will return the following message:

00000004Time;

### Function 26: Request monitors names.

With this function, the remote client can read the names of the monitors placed in the circuit.

From the client to the TCP server:

26;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The monitors names are delivered as a TAB separated array.

### Function 27: Request monitor data.

With this function, the remote client can read the content of a monitor.

From the client to the TCP server:

27;CCCCC….;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *CCCCC…* is the name of the monitor that the user wants to query, *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The sent message corresponds to a TAB separated matrix (rows separated by **CRLF**). The first row corresponds to the name of each variable. The following rows are the registers of the monitor. The values will be aligned with the header provided in row 0.

#### Example

Assume that your simulation was set to run in *time* mode and want to read a monitor called *meter_l118_0*, which is set in mode 0. In this case the OpenDSS command to be sent is as follows:

27;000Cmeter_l118_0;

If the transaction was successful (the monitor exists), OpenDSS-G will return the following message:

00000198hour**TAB** t(sec)**TAB** V1**TAB** VAngle1**TAB** V2**TAB** VAngle2**TAB** V3**TAB** VAngle3**TAB** I1**TAB** IAngle1**TAB** I2**TAB** IAngle2**TAB** I3**TAB** IAngle3**CRLF**0.000000**TAB**0.000000**TAB**2369.356445**TAB**-0.328578**TAB**2400.569336**TAB**-120.095154**TAB**2409.166504**TAB**120.137726**TAB**31.782425**TAB**-112.333206**TAB**39.646210**TAB**84.960808**TAB**34.650993**TAB**-22.327108**CRLF**;

## Function 28: Set the line list at OpenDSS-G.

With this function, the remote client can set the name of the lines that will be colored by the remote client. This functionality is useful when the remote client wants to change the color of specific lines displayed in OpenDSS-G.

From the client to the TCP server:

28; CCCC… ;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the list of lines to be colored, this list is a TAB separated array, *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The lines not specified in with this command will not be colored after refreshing the OpenDSS-G or when using function 29.

### Example

There are 3 lines that need to be colored later in the remote client's code. The line names are 650632, 632670, 670671. To activate these lines, the command to be send will be as follows:

28;0016650632**TAB**632670**TAB**670671**CRLF**;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 29: Set the line colors at OpenDSS-G.

With this function, the remote client can set the colors for the lines specified using function 28. If function 28 hasn't been used before function 29, the color list sent with function 29 will apply for the lines as they are sorted in OpenDSS. If the number of colors (*n*) in the array is less than the number of lines, only the first *n* lines will be affected.

From the client to the TCP server:

29; CCCC… ;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the list of colors. The list of colors is a byte array:
The first 4 bytes are the number of colors included in the list. The colors are 24 bit true-color defined using 4 bytes.
*LLLLLLLL* is the length of the message in hex (8 characters).

### Example

It is required to color the lines defined in the example for function 28. The colors to be used are red, green and blue. The command to be send will be as follows:

29;0010 x0 x0 x0 x3 x0 xFF x0 x0 x0 x0 xFF x0 x0 x0 x0 xFF;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

The effects of this function will be visible the next time the user refreshes the GUi in OpenDSS-G. To refresh the GUi use function 31.

## Function 30: Sets the color mode at OpenDSS-G.

With this function, the remote client can set the color mode for the active model in OpenDSS-G. The color modes are the ones that the user can select to highlight features such as voltage level, overloaded elements and power level, among others. There are 4 color modes that can be set with this function: Normal, power, isolated and emergency.

From the client to the TCP server:

29; CCCC… ;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the color mode index coded as a 5-digit integer (ASCII). *LLLLLLLL* is the length of the message in hex (8 characters).

### Example

It is required to set the power color mode. The command to be send will be as follows:

30;000500001;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 31: Refresh GUI.

With this function, the remote client can refresh the GUI in OpenDSS-G to show the latest modifications/changes.

From the client to the TCP server:

31;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 32: Read the list of elements selected in OpenDSS-G.

With this function, the remote client can obtain information about the elements selected using the selection tools provided with OpenDSS-G.

From the client to the TCP server:

32;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The message sent by OpenDSS-G is a TAB separated matrix (rows separated by **CRLF**) that includes the following information:

1. The index of the element in the database.
2. The type of object (Element or Line).
3. Basic description.
4. Element type (node, transformer, etc.) if the element is a line, this field will be empty.
5. Absolute coordinate X in the GUI.
6. Absolute coordinate Y in the GUI.
7. Rotation in the GUI (always 0).
8. Name of the element.
9. Relative coordinate X1 in the GUI.
10. Relative coordinate Y1 in the GUI.
11. Relative coordinate X2 in the GUI. If the element is a line, transformer, series reactor or series capacitor, X2 should be different from X1.
12. Relative coordinate Y2 in the GUI. If the element is a line, transformer, series reactor or series capacitor, Y2 should be different from Y1.
13. Element's BUS1 (if not a PDElement it will be empty)
14. Element's BUS2 (if not a PDElement it will be empty)
15. Element's polyline coordinates (deprecated)
16. Element's layer. The name of the layer where the element is located.

### Example

The user selects a bus using the *select several* tool in OpenDSS-G. To ask OpenDSS-G about the latest selection the command to be send will be as follows:

32;

If the transaction was successful, OpenDSS-G will return the following message:

0000003712**TAB**ELEMENT**TAB**NODE**TAB**1438**TAB**480**TAB**0**TAB**675**TAB**1452**TAB**480**TAB**1452 **TAB**480**TABTABTAB CRLF**;

## Function 33: Get the list of lines.

With this function, the remote client gets the list of lines placed at the active OpenDSS-G model.

From the client to the TCP server:

33;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The list of names if provided through a TAB separated array of strings.

## Function 34: Get the line ratings.

With this function, the remote client gets the list of ratings (NormAmps) for the lines included in the active OpenDSS-G model.

From the client to the TCP server:

34;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The list of ratings if provided through a TAB separated array of strings. The order in which the ratings are delivered is the same as the sorted list delivered in function 33.

## Function 35: Get the most recent event in OpenDSS-G (deprecated).

With this function, the remote client gets the latest event triggered in OpenDSS-G. If there was no event OpenDSS-G will answer with a *No event* message.

From the client to the TCP server:

35;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The message is a string describing the latest event occurred in OpenDSS-G. The events are user generated actions.

## Function 36: Refresh the overloads color combination.

With this function, the remote client commands OpenDSS-G to refresh the overloads visualization in OpenDSS-G.

From the client to the TCP server:

36;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 37: Refresh the voltage level color combination.

With this function, the remote client commands OpenDSS-G to refresh the voltage level visualization in OpenDSS-G.

From the client to the TCP server:

37;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 38: Refresh the powers color combination.

With this function, the remote client commands OpenDSS-G to refresh the powers visualization in OpenDSS-G.

From the client to the TCP server:

38;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 39: Get the column names of the incidence branch-to-node matrix.

With this function, the remote client can get the column names of the previously calculated incidence branch-to-node matrix. It is required before executing this function to execute function 1D at least once.

From the client to the TCP server:

39;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

## Function 3A: Get the row names of the incidence branch-to-node matrix.

With this function, the remote client can get the row names of the previously calculated incidence branch-to-node matrix. It is required before executing this function to execute function 1D at least once.

From the client to the TCP server:

3A;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

## Function 3B: Sort the branch-to-node incidence matrix.

With this function, the remote client can get the sorted version of the incidence matrix for the active circuit model. The sorting process occurs in OpenDSS by using the circuit tree to estimate the incidence branch-to-node matrix. The sorting process uses the substation as reference for positioning each PDElement from the substation to the farther bus in the circuit respect to the substation.

With this function, the remote client can read the branch-to-node incidence matrix of the compiled system.

From the client to the TCP server:

3A;

From the TCP server to the client:

LLLLLLLL MMMM…. ;

Where *LLLLLLLL* is the length of the message in hex (8 characters) and MMMM… is the message. The values for the matrix are delivered in compressed format specifying the row, column and value of each non-zero value. The message is delivered as a matrix separated by TAB characters. The row identifier is **CRLF**. Row 0 is the row index, row 1 is the column index and row 2 is the value for the specified cell. Row index, column index and value should match on each column of the TAB separated matrix.

## Function 3C: Get the column names of the sorted incidence branch-to-node matrix.

With this function, the remote client can get the column names of the previously sorted incidence branch-to-node matrix. It is required before executing this function to execute function 3B at least once.

From the client to the TCP server:

3C;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

## Function 3D: Get transformers’ ratings.

With this function, the remote client gets the list of ratings (NormAmps) for the transformers included in the active OpenDSS-G model.

From the client to the TCP server:

3D;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The list of ratings if provided through a TAB separated array of strings. The order in which the ratings are delivered is the same as the sorted list delivered in function 45.

## Function 3E: Get switches' ratings.

With this function, the remote client gets the list of ratings (NormAmps) for the switches included in the active OpenDSS-G model.

From the client to the TCP server:

3E;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The list of ratings if provided through a TAB separated array of strings. The order in which the ratings are delivered is the same as the sorted list delivered in function 19.

## Function 3F: Clear the control queue.

With this function, the remote client clears the control queue for the simulation running in OpenDSS-G.

From the client to the TCP server:

3F;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 40: Read the control queue.

With this function, the remote client reads the control queue for the simulation running in OpenDSS-G.

From the client to the TCP server:

40;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The list of ratings if provided through a TAB separated array of strings. Element 0 of the array is the header of the control queue, each register contains the features of the control action at the queue separated by commas (,).

## Function 41: Refresh the isolation color combination.

With this function, the remote client commands OpenDSS-G to refresh the isolated elements visualization in OpenDSS-G.

From the client to the TCP server:

41;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 42: Get bus names.

With this function, the remote client can get the bus names within the active circuit.

From the client to the TCP server:

42;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 43: Get node names.

With this function, the remote client can get the node names within the active circuit.

From the client to the TCP server:

43;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 44: Get load names.

With this function, the remote client can get the load names within the active circuit.

From the client to the TCP server:

44;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 45: Get transformer names.

With this function, the remote client can get the transformer names of active circuit.

From the client to the TCP server:

45;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 46: Get capacitor names.

With this function, the remote client can get the capacitor names within the active circuit.

From the client to the TCP server:

46;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 47: Get storage names.

With this function, the remote client can get the storage names within the active circuit.

From the client to the TCP server:

47;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 48: Draw flags at the buses listed.

With this function, the remote client can draw flags on the active circuit displayed in OpenDSS-G. the flags can only be associated to buses, so it is expected to receive the list of buses where the flag will be placed. This command refreshes the GUI automatically.

From the client to the TCP server:

48; CCCC…;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the list of buses where the flags will be placed. The format of the list is a TAB separated matrix (rows separated by **CRLF**). Since flags can have different colors, the matrix format allows to group buses to have the same flag color. As a result, buses defined in row 0 will have one color, buses in row 1 another color and so on. *LLLLLLLL* is the length of the message in hex (8 characters).

### Example

The user wants to place flags at buses *sourcebus*, *rg60* and *632*, each one will have a different color. In this case the frame to send will be as follows:

48;0016sourcebus**CRLF**rg60**CRLF**632**CRLF**;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 49: Set flag colors.

With this function, the remote client can set the colors for the flags defined in function 48. Function 49 can be used before using function 48, the only condition is to make sure that the number of colors defined with this function is equal to the number of rows of the matrix defined for function 48.

From the client to the TCP server:

49; CCCC… ;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the list of colors. The list of colors is a byte array: The first 4 bytes are the number of colors included in the list. The colors are 24 bit true-color defined using 4 bytes.
*LLLLLLLL* is the length of the message in hex (8 characters).

### Example

It is required to color the flags defined in the example for function 48. The colors to be used are red, green and blue. The command to be send will be as follows:

49;0010 x0 x0 x0 x3 x0 xFF x0 x0 x0 x0 xFF x0 x0 x0 x0 xFF;

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 4A: Get PV system names.

With this function, the remote client can get the PV system names within the active circuit.

From the client to the TCP server:

4A;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 4B: Get generator names.

With this function, the remote client can get the generator names within the active circuit.

From the client to the TCP server:

4B;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 4C: Get UPFC names.

With this function, the remote client can get the UPFC names within the active circuit.

From the client to the TCP server:

4B;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM*... is the message. The data is provided as a TAB separated array of strings.

## Function 4D: Draw text at the buses listed.

With this function, the remote client can draw text on the active circuit displayed in OpenDSS-G. the text can only be associated to buses, so it is expected to receive the list of buses where the text will be placed. This command refreshes the GUI automatically.

From the client to the TCP server:

4D; CCCC…;

From the TCP server to the client:

LLLLLLLL OK ;

*CCCC…* is the list of buses where the flags will be placed. The format of the list is a TAB separated matrix (rows separated by **CRLF**). The color of the text can be set with function 49; the matrix format allows to group buses to have the same flag color. As a result, buses defined in row 0 will have one color, buses in row 1 another color and so on. Each bus name must have associated the text to place next to it. For linking the text to the bus use the less than character (>). For example, to write the text *hello* at bus *examplebus*, the string will be *examplebus>hello*. *LLLLLLLL* is the length of the message in hex (8 characters).

### Example

The user wants to write the text *note1*, *note2*, *note3* at buses *sourcebus*, *rg60* and *632* respectively. In this case the frame to send will be as follows:

4D;0026sourcebus>note1**TAB**rg60>note2**TAB**632>note3**CRLF**;

In this case all the texts will have the same color.

If the transaction was successful, OpenDSS-G will return the following message:

00000002OK;

## Function 4E: Clear flags.

With this function, the remote client will remove all the flags and text placed on the GUI remotely or locally.

From the client to the TCP server:

4E;

From the TCP server to the client:

LLLLLLLL OK ;

*LLLLLLLL* is the length of the message in hex (8 characters).

## Function 4F: Get energy meter names.

With this function, the remote client can get the energy meter names within the active circuit.

From the client to the TCP server:

4F;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

## Function 50: Get VCCS names.

With this function, the remote client can get the VCCS names within the active circuit.

From the client to the TCP server:

50;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

## Function 51: Get the names of the components within a class.

With this function, the remote client can get the names of the objects created for an specific class within the active circuit. A class can be storage, load, line, transformer, PVsystem, UPFC, etc. This is a simplified function to perform this type of query in a global way and considering all the classes within OpenDSS.

From the client to the TCP server:

51 CCCC… ;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*CCCC…* is the name of the class to be query. *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings.

### Example

The user wants to know the names of the VSources within the active model, in this case the frame to be sent is as follows:

51;0007vsource;

If the transaction was successful, OpenDSS-G will return the following message:

00000008source**CRLF**;

## Function 52: Get OpenDSS-G simulation engine status.

With this function, the remote client can get the status of the active actors in the current session of OpenDSS-G. The status is represented by an integer number where “1” means that the actor is ready for a new simulation job, otherwise, the actor will report “0” indicating that there is a simulation job in progress.

From the client to the TCP server:

52;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*CCCC…* is the name of the class to be query. *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings. The number of elements of the TAB separated array depends on the number of actors created for the current simulation job.

Example

The user has a yearly simulation carried out using 6 actors (temporal parallelization). The simulation has finished, however, the user wants to know the simulation status remotely. In this case the frame to be sent is as follows:

52;

If the transaction was successful, OpenDSS-G will return the following message:

0000000D1**TAB**1**TAB**1**TAB**1**TAB**1**TAB**1**CRLF**;

## Function 53: Get OpenDSS-G simulation engine progress.

With this function, the remote client can get the progress of the active actors in the current session of OpenDSS-G. The progress is represented by an integer number for showing the progress in percentage of the simulation job in progress.

From the client to the TCP server:

53;

From the TCP server to the client:

LLLLLLLL MMMM… ;

*CCCC…* is the name of the class to be query. *LLLLLLLL* is the length of the message in hex (8 characters) and *MMMM…* is the message. The data is provided as a TAB separated array of strings. The number of elements of the TAB separated array depends on the number of actors created for the current simulation job.

Example

The user has a yearly simulation carried out using 6 actors (temporal parallelization). The simulation has finished, however, the user wants to know the simulation progress remotely. In this case the frame to be sent is as follows:

53;

If the transaction was successful, OpenDSS-G will return the following message:

00000019100**TAB**100**TAB**100**TAB**100**TAB**100**TAB**100**CRLF**;