# Q LANGUAGE - MESSAGE HANDLER

 

When a **q** process connects to another **q** process via inter-process communication, it is processed by message handlers. These message handlers have a default behavior. For example, in case of synchronous message handling, the handler returns the value of the query. The synchronous handler in this case is **.z.pg**, which we could override as per requirement.

Kdb+ processes have several pre-defined message handlers. Message handlers are important for configuring the database. Some of the usages include −

- **Logging** − Log incoming messages *helpfulincaseofanyfatalerror*,
- **Security** − Allow/disallow access to the database, certain function calls, etc., based on username / ip address. It helps in providing access to authorized subscribers only.
- **Handle connections/disconnections** from other processes.

## Predefined Message Handlers

Some of the predefined message handlers are discussed below.

### .z.pg

It is a synchronous message handler *processget*. This function gets called automatically whenever a sync message is received on a kdb+ instance.

Parameter is the string/function call to be executed, i.e., the message passed. By default, it is defined as follows −

```
.z.pg: {value x}                / simply execute the message
                                  received but we can overwrite it to
give any customized result.
.z.pg : {handle::.z.w;value x}  / this will store the remote handle
.z.pg : {show .z.w;value x}     / this will show the remote handle
```

### .z.ps

It is an asynchronous message handler *processset*. It is the equivalent handler for asynchronous messages. Parameter is the string/function call to be executed. By default, it is defined as,

```
.z.pg : {value x}        / Can be overriden for a customized action.
```

Following is the customized message handler for asynchronous messages, where we have used the protected execution,

```
.z.pg: {@[value; x; errhandler x]}
```

Here **errhandler** is a function used in case of any unexpected error.

### .z.po[]

It is a connection open handler *process − open*. It is executed when a remote process opens a connection. To see the handle when a connection to a process is opened, we can define the .z.po as,

```
.z.po : {Show “Connection opened by” , string h: .z.h}
```

### .z.pc[]

It is a close connection handler *process − close*. It is called when a connection is closed. We can create our own close handler which can reset the global connection handle to 0 and issue a command to set the timer to fire *execute* every 3 seconds 3000*milliseconds*.

```
.z.pc : { h::0; value “\\t 3000”}
```

The timer handler . *z. ts* attempts to re-open the connection. On success, it turns the timer off.

```
.z.ts : { h:: hopen `::5001; if [h>0; value “\\t 0”] }
```

## .z.pi[]

PI stands for process input. It is called for any sort of input. It can be used to handle console input or remote client input. Using .z.pi[], one can validate the console input or replace the default display. In addition, it can be used for any sort of logging operations.

```
q).z.pi
'.z.pi

q).z.pi:{">", .Q.s value x}

q)5+4
>9

q)30+42
>72

q)30*2
>60

q)\x .z.pi
>q)

q)5+4
9
```

## .z.pw

It is a validation connection handler *userauthentication*. It adds an extra callback when a connection is being opened to a kdb+ session. It is called after the –u/-U checks and before the .z.po *portopen*.

```
.z.pw : {[user_id;passwd] 1b}
```

Inputs are **userid** *symbol* and **password** *text*