Documentation of the E2E Transaction Logger

The Module

Syntax	`require('e2e-transaction-logger');`
Semantics	The module itself is an object of type TransactionLogger.

Examples	`var transactionLogger = require('e2e-transaction-logger');`
	`var trx = transactionLogger.startTransaction('MyTransaction');`
	
	You can use directly the module to trace transactions

TransactionLogger

constructor

Syntax	`TransactionLogger.constructor(logPath)`	
Semantics	Creates an object of type TransactionLogger.	
Arguments	`logPath`	The path where the log files will be written.
Examples	`var TransactionLogger = require('e2e-transaction-logger').TransactionLogger;` `var myTransactionLogger = new TransactionLogger('my/custom/log/path');`	

startTransaction

Syntax	`TransactionLogger.startTransaction(name)`	
Semantics	Traces the start of a transaction and return a Transaction object.	
Arguments	`name`	The name of the transaction.
Examples	`var trx = transactionLogger.startTransaction('MyTransaction');`	

processLogger

Syntax	`require('e2e-transaction-logger').processLogger;`
Semantics	An object of type ProcessLogger.

Examples	`var transactionLogger = require('e2e-transaction-logger');`
	`transactionLogger.processLogger.processStart('ProcessName',`
	`'processId', 'StartEventName');`

ProcessLogger

constructor

Syntax	`ProcessLogger.constructor(logPath)`	
Semantics	Creates an object of type ProcessLogger.	
Arguments	`logPath`	The path where the log files will be written.

Examples	`var ProcessLogger = require('e2e-transaction-logger').`
	`ProcessLogger;`
	`var myProcessLogger = new ProcessLogger('my/custom/log`
	`/path');`	

processStart

Syntax	`ProcessLogger.processStart(processName, processId, eventName)`	
Semantics	Traces the start of a process.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`eventName`	The start event name.
Examples	`processLogger.processStart('ProcessName', 'processId',` `'StartEventName');`	

processEnd

Syntax	`ProcessLogger.processEnd(processName, processId, eventName)`	
Semantics	Traces the end of a process.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`eventName`	The end event name.

Examples	`processLogger.processEnd('ProcessName', 'processId', 'EndEventName');`

processStateStart

Syntax	`ProcessLogger.processStateStart(processName, processId, stateName)`	
Semantics	Traces the start of a state.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`stateName`	The state name.
Examples	`processLogger.processStateStart('ProcessName', 'processId', 'StateName');`	

processStateEnd

Syntax	`ProcessLogger.processStateStart(processName, processId, stateName)`	
Semantics	Traces the end of a state.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`stateName`	The state name.
Examples	`processLogger.processStateEnd('ProcessName', 'processId', 'StateName');`	

processChoice

Syntax	`ProcessLogger.processChoice(processName, processId, gateway, choice)`	
Semantics	Traces a choice.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.

	`gateway`	The gateway name.
	`choice`	The choice name.
Examples	`processLogger.processChoice('ProcessName', 'processId', 'Gateway', 'Choice');`	

processEvent

Syntax	`ProcessLogger.processEvent(processName, processId, eventName)`	
Semantics	Traces an event.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`eventName`	The eventname.
Examples	`processLogger.processStateStart('ProcessName', 'processId', 'EventName');`	

processValueString

Syntax	`ProcessLogger.processValueString(processName, processId, key, value)`	
Semantics	Traces a value of a String type.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`key`	The key.
	`value`	The value
Examples	`processLogger.processValueString('ProcessName', 'processId', 'Key', 'MyValue');`	

processValueFloat

Syntax	`ProcessLogger.processValueFloat(processName, processId, key, value)`	
Semantics	Traces a value of a Float type.	

Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`key`	The key.
	`value`	The value
Examples	`processLogger.processValueString('ProcessName', 'processId', 'Key', 123.456);`	

processValueDateTime

Syntax	`ProcessLogger.processValueDateTime(processName, processId, key, value)`	
Semantics	Traces a value of a Date type.	
Arguments	`processName`	The name of the process.
	`processId`	The process id.
	`key`	The key.
	`value`	The value
Examples	`processLogger.processValueString('ProcessName', 'processId', 'Key', new Date());`	

Transaction

A Transaction object is returned by the **startTransaction** method of a TransactionLogger. It is then used to trace what happens inside the transaction using **startIO** or any ProcessLogger method. Using transactions to trace processes instead of a ProcessLogger directly allows you to correlate the process steps and IOs of these transactions.

end

Syntax	`Transaction.end(state)`	
Semantics	Traces the end of a transaction.	
Arguments	`state`	The state of the transaction. It can be true or 'OK' for a success, false or 'ERROR' for a failed. Default: 'OK'

Examples	`var trx = transactionLogger.startTransaction ('MyTransaction');`
	
	`// do some things`
	
	`trx.end();`

startIO

Syntax	`Transaction.startIO(name, domain, system)`	
Semantics	Trace the start of an IO call and return an IO object.	
Arguments	`name`	The name of the IO call
	`domain`	The domain of the IO call
	`system`	The system of the IO call
Examples	`var io = trx.startIO('SELECT Users','SQL','oracle/XE');`	

IO

An IO object is returned by the **startIO** method of a Transaction. It is then used to trace the end of the IO call.

end

Syntax	`IO.end(state)`	
Semantics	Traces the end of a IO call.	
Arguments	`state`	The state of the IO call. It can be true or 'OK' for a success, false or 'ERROR' for a failed. Default: 'OK'

Examples	`var io = trx.startIO('Read','FILE','HelloWorld.txt');`
	
	`fs.readFile(__dirname + '/resource/HelloWorld.html',function (err, data){`
	`  io.end();`
	`});`	

transactionLoggerMiddleware

A function that returns a middle ware which can be used with express. It will automatically start a transaction when a request arrives and end it when the response is sent. The Transaction object will be available in the request object as **trx**.

Arguments	`options`	object
	• `logPath`	The path where the log files will be written.
	• `name`	The name that should be given to the transactions. Can be a function taking the request and response as parameter and returning the name. By default the name will be the URL of the request.
Examples		

```
var express = require('express');
var fs = require('fs');
var transactionLoggerMiddleware = require('e2e-transaction-
logger').transactionLoggerMiddleware;

var app = express();

app.use(transactionLoggerMiddleware());

app.get('/hello.html', function(req, res){
    var io = req.trx.startIO('Read','FILE','HelloWorld.
html');

    fs.readFile(__dirname + '/resource/HelloWorld.html',
function(err, data){
        if(err){
            io.end('ERROR');
            res.send(503, err); // the status code define if
the transaction failed or succeed. >= 400 it failed
            return;
        }

        io.end();
        res.set('Content-Type', 'text/html');
        res.send(data);
    });
});
```