MODBUSCLIENT DLL API
Rev. 4.

Revision	Functions	Comment

3.0	`startDataClientPLC`
	`updateThicknessPLC`
	`getThicknessUnitPLC`
	`stopMeasurementPLC`
	`startMeasurementPLC`
	`setRecipePLC`	

3.5	`getRawData`
	`getRawDataFloat`
	`calculateData`	

4.0	`getIntTimeMicrosec`
	`getThicknessFloat`
	`updateCalculationPropertiesPLC`	

I. INTRODUCTION

ModbusClient.dll allows easy remote control of the MProbe measurement via TCP-IP using Modbus protocol communication.
ModbusClient.dll can be located on the same computer as TFCompanioo software (that is connected to MProbe system) or on any other computer on the network.

ModbusClient.dll is a C library that is interfacing java implementation of the Modbus Client (located in Modbus.jar). ModbusClient.dll is communicating (via Modbus.jar) with TFCompanion's build-in ModbusServer software**.**

II. Configuration.

1. Java should be installed on the computer.
   If you are using a different computer on a network (not the one where TFCompanion is installed) – check that java is installed.
   You can check it by typing: **java –version** at command prompt. Java version should be 1.8 or 1.9 (earlier versions maybe incompatible). Versions later than 1.9 are not tested
2. **ModbusClient.dll** and **jvm.dll** should be added to the system path
   (jvm.dll is located in java installation directory e.g.
   C:\Program Files\Java\jre1.8\bin\server)

Note. The log file (Modbus.log) will be saved in the working directory of the program that calls **ModbusClient.dll**
The structure of the installation directory:

<home_directory>/
<executable program that uses ModbusClientDLL..dll >
Modbus.properties (optional)
library/
--- ModbusClientDLL..dll
--- Modbus.jar
--- tflib.jar
--- log4j.jar
--- comm..jar
log/
---log_config.properties

<home_directory> is the directory where executable (calling) program (the program that uses ModbusClientDLL.dll) is located e.g. ModbusExample.exe

III. API DETAILS

1. Initialization.

This need to be the first call to the dll. This method is preparing and starting JVM

```
/**
 @param ip - IP address of the computer with Modbus server. To determine IP address, use ipconfig at command prompt. If computer is not connected - the IP address will a "localhost"
It is expected that TFCompanion server (MProbe) is started at the port 502
 If ip address=0, the program will try to read IP address and port information from the Modbus.properties file
 @ret 0 – success, otherwise failed.
*/
long  init(char* ip);
```

2. Connection to server

```
/*
Connects Modbus Client to Modbus server. The IP address is already set during initialization, the  port#502 is a default Modbus port that is used here.
If you are using firewall or virus protection program - please make sure that port#502 is open.
By default connection timeout is 3000 ms. If response from the server is not received during this time - connection is terminated. The client will try to reestablish connection 2 times before giving up.
To set custom timeout use setConnectionTimeout(int time_ms) method - this method need to be called BEFORE connectClient() otherwise it will have not effect.

return boolean. 1 - true: connection is successful, 0 (false) - problem during connection

*/
 unsigned char connectClient();
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

3. **Setting measurement parameters**.

Measurement parameters need to be set before the first measurement is executed. These parameters are remembered and used for all future measurements. If needed, parameters can be reset at any time.

```
/**
     * Sets measurement parameters in the output Hashtable to prepare request.
     * The content of the Hashtable is maintained and used for all transactions.
     * @param channel - channel number (0 or 1)
     * @param waferID - Sample ID (up to 40 bytes length)
     * @param recipeName - name of the measurement or calibration recipe (up to 10
     *  bytes long). If recipe does NOT have filmstack attached – Filmstack needs
     *  to be set separately before calling measure
     *@see measure()
     *@see setFilmstack()
     * @param expectedThickness - OPTIONAL PARAMETER (0 - to ignore). Value of the
         Expected thickness - resets the thickness of the currently used filmstack
     * @param intTime - OPTIONAL PARAMETER (0 - to ignore). Resets integration
     time in the specified recipe measurement recipe
     * @return true (1) if successful
     */

unsigned char setMeasurement(int channel,
                             char* waferID,
                             char* recipeName,
                             int expectedThickness,
                             short intTime)
```

4. Measurement

This methods instructs MProbe to perform the measurement. The measurement is fully defined by the measurement recipe (recipe should have a filmstack attached). Measurement call is blocking (it is waiting for the response from the server). The default timeout is set to 3 sec. (see @setConnectionTimeout()) If response is not received – client will disconnect and try connect again (it will repeat it 3 times before giving up)

```
/**
     * Execute measurement or calibration based on the data set in the output
       Hashtable defined by setMeasurement and setRecipe
     * @return response from the server. This response can be parsed directly
       using Response(Read) table spec or
     * use convenience methods that parse response information.
     * @see  hasException(),getThickness()
     */
signed char* measure()
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

Example of use:

```
unsigned char * measureSample(){
unsigned char res;
long thickness;
signed char * byteResponse=NULL;

double thicknessError;
double GOF;
res=isClientConnected();
if(res==0){
// try to restore the connection
       res=restoreConnection();
}
if(res==1){
// do the measurement
 // perform actual measurement
          byteResponse=measure();
          //byteResponse=measure();
       // need to check if there was any exception
          if(checkExceptionStatus()){
              //check if there are any warning
                 checkWarningStatus();
                 // use utility method to parse server response
                 // 0 is the channel #
                 thickness= getThickness(0);
                 fprintf(stderr,"Thickness=%d \n", thickness);
                 //getting thickness error
                 // this is the confidence interval
                 thicknessError=getThicknessError(0);
                 fprintf(stderr,"Thickness Error =%e\n", thicknessError);
                 // check goodness of fit - GOF
                 GOF=getGOF(0);
                 fprintf(stderr,"GOF =%e\n ", GOF);
          }else{
          fprintf(stderr,"******* MEASUREMENT EXCEPTION ******** \n");
          }
}else{
       fprintf(stderr,"******* PROBLEM ESTABLISHING CONNECTION ******** \n");
       }
return byteResponse;}
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

```
boolean checkExceptionStatus(){
       boolean ret=FALSE;
       signed char exception=getExceptionCode();
       if(exception==0){
           printf("Exception status: NO EXCEPTIONS \n" );
              ret=TRUE;
       }else if(exception==5){
           printf("Exception status: DATA ACQUISITION EXCEPTION \n");
       }else if(exception==6){
           printf("Exception status: CALCULATION EXCEPTION \n");
       }else if(exception==7){
           printf("Exception status: UNKNOWN EXCEPTION \n");
       }else if(exception==8){
           printf("Exception status: REQUEST CONTENT EXCEPTION \n");
              printf("Please check the measurement recipe name and filmsstack name
\n");
        printf("are available in the database \n");
       }

return ret;}
```

5. Checking measurement results

Following are convenience methods that can be called AFTER measurement is performed.

5.1 Checking for exception (Problems during measurement, calculation or communication)

```
/**
     * Parse server response to extract exception code during transaction
        (measurement/calibration)
     * Can be called after response was received
     * @return 0 - no exception, 1,2 or 3 - general Modbus errors (communication/
        format problem),
     * 4 - unknown sensor error, 5 - data acquisition problem, 6 - calculation
        problem,  7 - system problem (generic exception)
     */
signed char getExceptionCode()
```

5.2 Thickness data

```
/**
Parse server response to extract thickness data
@param channel – the measurement channel
@return  - thickness in Angstroms
*/

long getThickness(int channel)


/**
Added in Rev. 4
@ret thickness value as a float in default unit (unit defined in TFCompanion
configuration)
*/
float getThicknessFloat();
```

5.3 Thickness confidence interval (90%)

```
/**
* Parse server response to extract confidence interval data
@param channel: measurement channel
@return confidence interval value
*/
double getThicknessError(int channel)
```

5.4 Goodness of Fit

```
/**
* Parse server response to extract GOF data
@param channel: measurement channel
@return GOF value
*/
double getGOF(int channel)
```

6. Convenience methods to change measurement parameters

6.1 Changing measurement recipe

```
/**
     * A convenience method to change measurement recipe used in transaction.
     * @ see setMeasurement method.
     * The data is stored in the hashtable and will be used to prepare next
       response
     * @param recipeName the name of measurement recipe
     * @param channel measurement channel number
     * @return true (1)- recipe set successfully, false (0) - problem setting
        recipe.
     */

 unsigned char setRecipe(int channel, char* recipeName)
```

6.2.Changing Filmstack

```
/**
* Sets new Filmstack
* If recipe already have filmstack attached - it is replaced with the new
* filmstack
* @param matName - filmstack name, length up to 40 char
* @return true (1)- recipe set successfully, false (0) - problem setting
       recipe.
*/
signed char* setFilmstack(char* matName);
```

6.3. Toggle between measurement and measurement/diagnostics type. measurement/diagnostics is the same as measurement only server send more information about the system

```
/**
     * Convenience method to switch between standard measurement and measurement
       w/ diagnostics Can be applied after setMeasurement was called
```

```
     * @see setMeasurement method
     * @param diag true (1) to enable measurement with diagnostics, false (0)
        standard measurement
     * standard measurement is the default
        *@return  true (1) if successful, false (0) if problem
     */
unsigned char setDiagnostics(unsigned char flag)
```

6.4 Add/Remove layer

```
/**
* Adds a new layer on top of the filmstack with default thickness 10A
* Set thickness of this layer as a calcualted/display parameter and disable
* other calculated parameters
* @param matName the name of the material
*/
signed char* addLayer(char* matName);

/**
* Removes the layer with the specified material
*@param matName the name of the material in the layer to be removed
*/
signed char* removeLayer(char* matName);
```

7. Diagnostic convenience methods.

These methods can be used ONLY when measurement with diagnostics is enabled. They parse server response to extract requested data, so it should be called after the measurement is completed.

7.1. Warnings

```
/**
@return warning code
NO_WARNING=0
LOW_SIGNAL=1 (low light intensity)
HIGH_SIGNAL 2 (intensity maybe too high - saturation)
LOW_MEMORY 3 (low memory available to the program)
*/
signed char getWarning()
```

7.2. Signal level

Need to be called to determine actual signal level if there is a low or high signal warning.

```
/*
@param channel: measurement channel
@return actual signal level (intensity) in % of the total range
*/
short getSignalLevel(int channel)
```

7.3. Integration time

Actual integration time used by spectrometer

```
/**
Parse server response to extract integration time
Return actual integration time used during the measurement
@param channel channel number
```

```
@return integration time in ms
*/
short getIntegrationTime(int channel)



/**
Added in rev.4
@ret integration time in microseconds

*/
int getIntTimeMicrosec();
```

8. System status.

Checking system status after the last measurement call.

```
/*

     * Parse server response to check system status
     * Can be called after response was received
     * @return 0 - ready/idle,1 - measuring, 2 - calculating, 3 - exception
state,4 - busy/initializing
*/
signed char getSystemStatus()
```

9. Connection checking.

```
/*
Checking the status of the connection
@return  0 – false (disconnected), 1-true (connected)

*/
 unsigned char isClientConnected()

/*
Sets connection timeout in milliseconds. Default timeout id 3000ms
Setting timeout to 0 –means connection will never timout.
IMPORTANT: This method should be called BEFORE calling connectClient();
Otherwise it will have no effect
*/
void setConnectionTimeout(int time_ms);

/**
@return current timout in milliseconds
*/
int getConnectionTimeout();
```

10. Checking intensity (Added Jan. 10, 2017)

10.1. Check Maximum Signal.

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

This method allows to check maximum signal from the system before doing actual measurement. A full spectrum is acquired and a maximum value of the intensity is determined. Averaging of the 10 spectra is done to have accurate data.
In case of the system with Flush lamp – setStrobe() command should be send first to activate synchronization pulses.
In case of all other systems, integration time should be checked to make sure it is the same as will be used during measurement (signal value will depend directly on integration time)

```
/**
@return maximum intensity as counts of a 16 bit ADC from 0 to 65500
*/
unsigned short getMaxIntensity()
```

10.2 Set Intensity

This method allows to set intensity of the light source.

```
/**
@param value – value in the range of 0 to 100, 100 – maximum intensity
@return a response from the server  (byte[])
*/
signed char* setIntensity(unsigned char value)
```

Note. Function groups 11 and 12 require TFCompanion build 04052024 or later and corresponding Modbus.jar library version

11. Retrieving measured and calculated Reflectance/Transmittance data

These functions are not compliant with Modbus specification because of the length of the message (Modbus message is limited to 256 bytes)
During each measurement the data is cached. The cache is replaced after each measurement. These methods return the results of the last measurement taken

```
/**
* @ret a byte array of the raw reflectance and calculated data
* as a 4 byte floats: wavelength, measured value, calculated value
* The last 4 bytes is a GOF (goodness of fit value)
* @ret length - the length of the byte array returned
*/
 char* getRawData(int* length);


/**
*
@ret a n array of floats same as @see getRawData
@ret length - the length of the array for convenience
*/
float* getRawDataFloat(int* length);
```

Example of use:

```
char getRawReflectanceFloat() {
	char ret = 1;
	int length = 0;
	float* data;
	int pos = 0;
	data = getRawDataFloat(&length);
	if (length > 0) {
	fprintf(stderr, "returned float array length= %d \n", length);
	fprintf(stderr, "Wavelength : Measured Data : Calculated  data ");

while (pos < (length - 1)) {
fprintf(stderr, " %f    %f   %f \n", data[pos], data[++pos], data[++pos]);
pos++;
 }
		fprintf(stderr, " %f   GOF= \n", data[length-1]);
	}
	else {
		ret = 0;
		fprintf(stderr, "returned float array length= %d \n", length);
	}
	return ret;
}


**
Added Modbus rev. 3.5
This function is intended to be used after measurement is done and raw data
is received
@ see getRawDataFloat
param float* - an array of wavelengths and measured reflectance as floats
(32bit)
Each measurements point includes wavelength and reflectance without
separators: lam[0]reflectance[0]....lam[n]reflectance[n]
@param length - the length of the array
@return 0 - success, 1 - failure
After function executed successfully, can use getRawDataFloat to retrieve data.
@see getRawDataFloat(int* length)
*/
unsigned char calculateData(float* data, int length);
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

Example of use:

```
void testCalculateData(char* recipeName) {
	char res = 1;
	int length = 0;
	float* data;
	float* inputData;
	int pos = 0;
	int count = 0;
	setMeasurement(0, "TestWafer", recipeName, 0, 0);
	executeMeasurement();

	data = getRawDataFloat(&length);
	if (length > 0) {
		fprintf(stderr, "returned float array length= %d \n", length);
		// repackage the data to send it back for recalculation
		//only 2 columns (wavelength and measured data out of 3 columns)
		// need to be sent back
		int inputDataLength = 2*((length - 1)/3);
		inputData=malloc(sizeof(float) * inputDataLength);
		while (pos < (length - 1)) {
			inputData[count] = data[pos];
			count++;
			pos++;
			inputData[count] = data[pos];
			pos += 2;
			count++;

		}
		fprintf(stderr, "input data is prepared \n");
		res=calculateData(inputData, inputDataLength);
		fprintf(stderr, "Calculation completed, return= %d \n", res);
		if (res == 0) {
			data = getRawDataFloat(&length);
		}

	}
}
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

12. Pass-through functions that are executed by the PLC library

```
**
* Following functions require PLC.jar on the classpath.
* They are passthrough function that are executed by the PLC library
* Note.
*  a. Measurement recipe need to be set to PLC using
*     setRecipePLC() function.
*  b. Opposite to other Modbus functions, these functions return 0 if success and
*     1 – is failed.
*  Example of using PLC functions:
*  void testPLCFunctions(char * recipeName, char* ip, int port) {
       char ret = 1;
       fprintf(stderr, "Prepare setRecipe %s= \n",recipeName);
       ret=setRecipePLC(recipeName);
       fprintf(stderr, "Recipe set, ret= %d \n", ret);

       ret=startDataClientPLC(ip, port);
       fprintf(stderr, "DataClient started, ret= %d \n", ret);

       ret=startMeasurementPLC(0);
       fprintf(stderr, "Measurement started, ret= %d \n", ret);

       Sleep(5000);

       ret=stopMeasurementPLC(0);
       fprintf(stderr, "Measurement stopped, ret= %d \n", ret);

       ret=getThicknessUnitPLC();
       fprintf(stderr, "Thickness unit, ret= %d \n", ret);

       ret=updateThicknessPLC("1_Thickness", 210.0, 1);
       fprintf(stderr, "Thickness updated, ret= %d \n", ret);


}
*
*/

/**
Added Rev. 3.0
* Starts build-in Data Client of the PLC server
* Data client connects to an external Data Server.
* The type of the expected  Data server is defined in the PLC.properties
* The options are:
* - analog output is directed to the DAC (in this case IP and port corresponds to
the FPGA board connected to DAC)
* - TCP the TCP data client is starts and expects a TCP data server to connect to.
* - UDP the UDP data client starts and expects a UDP data server to send data (not
current implemented)
* Data server should be running on IP address and port specified in the call
 @param ip IP address of the Data Server
 @param port is a port of the Data server
  @ret 0 - success, 01- failed
*/
unsigned char startDataClientPLC(char* ip, int port);

/**
Added Rev. 3.0
@param name - the name of the measurement recipe
```

```
The measurement recipe is required to have a filmstack attached
This recipe is set in the PLC library and is independent of any
recipe set in Modbus server
@ret 0 - success, 1 - failed
*/
unsigned char setRecipePLC(char* name);


/**
Added Rev. 3.0
Start continuous measurement using PLC server library
@param delay - delay between the measurements in ms.
It overrides the delay set in the PLC.properties file
@ret 0 - success, 1 - failed
*/

unsigned char startMeasurementPLC(int delay);




/**
Added Rev. 3.0

Stops contnuous measurement
@see startMeasurementPLC
@param cond - currently ignored
The program will allow the currently running
calculation to complete before stopping (normal termination)
@ret 0 - success, 1 - failed
*/
unsigned char stopMeasurementPLC( char cond);

/**
@ret currently used thickness unit
0- Angstroms
1- nm
2 - microns
The default unit is set in TFCompanion configuration
*/

unsigned char getThicknessUnitPLC();

/**
Added Rev. 3.0
Updated the thickenss value in the filmstack used for calculations
in the PLC library
@param paraName is the name of the parameter. 1_Thickness - first layer thickness
@param thickness - the value of the new thickness
@param unit unit used for the thickness param
@ret 0 if success, 1 - failed
*/

unsigned char  updateThicknessPLC(char* paramName, float thickness, int
unit);
```

83 Pine Hill Rd. Southborough, MA 01772
Phone +1.617.388.6832 Fax. +1.508.858.5473
email: info@semiconsoft.com http://www.semiconsoft.com

```
/**
Added Rev. 4
* Updates calculation algorithm in the PLC library during the measurement
* This function should be used during the measurement. it will fail if
measurement is not started
*
@param XML element <Calculation_Conditions> form the measurment recipe
       that defines the calculation algorithm and related parameters
@ret 0 if success, 1 - failed
*/

unsigned char  updateCalculationPropertiesPLC(char* xmlString);
```

Example of XML element for FFT calculation

```
"<Calculation_Conditions>"
"<algorithm>Thick Film FFT< / algorithm>"
"<fft_level>0.04</fft_level>"
"<FFT_interface>false</FFT_interface>"
<Calc_Preprocess_remove_trend>true</Calc_Preprocess_remove_trend>"
"<CALC_ACCURACY>0</CALC_ACCURACY>"
"<Calc_total_thickness>false</Calc_total_thickness>"
"</Calculation_Conditions>";
```