

TG Motion

version 4

Virtual PLC module

operation manual

Revision History

date	version	revision
31 July 2017	1.0	Initial release

Contents

1. Virtual PLC	4
1.1 Virtual PLC description	4
1.2 PLC creation	4
1.3 Required PLC algorithm	4
1.4 Communication with PLC	5
2. PLC functions	6
2.1 Function general description	6
2.2 Structure PLC_DATA	6
2.3 Functions sorted by priority	7
2.4 Time sequence of function calls	11
3. PLC debugging tools	17
3.1 Control Observer	17
3.2 Listings	17
3.3 Oscilloscope	17
3.4 Windows applications	18

1. Virtual PLC

1.1 Virtual PLC description

The Virtual PLC module executes the program, which has been written by the user. It carries out calculations, ensures the servo drive control, input and output value loading and setting and takes care of the communication with other peripherals through the shared memories.

The Virtual PLC is called by TG Motion periodically within an interval which is set in **Cycle_Time**. The length of **Cycle_Time** is defined in **TGMotion4xx.ini** file.

An advantage of the virtual PLC consists in its speed, as it runs in the CPU machine code directly.



Servo and I/O groups unify the controlling interface for different servo drive types and I/O unit types. The same PLC code can be applied to different servo drives or I/O units. Servo drives or I/O units can be changed in a flexible way without any need for rewriting the PLC code.

1.2 PLC creation

A PLC program may be created in a general development environment, such as Visual Studio, Delphi. The programming language may be C, C++ and Pascal.

In the development environment, a *.tgm file must be created, which must export just six functions named as follows: **Program_01**, **Program_02**, **Program_03**, **Program_04**, **Program_05** and **Program_Ini**. The above-mentioned names are mandatory, including the letter case.

The *.tgm program module must not be linked to any external module or DLL. Any program functionalities (functions, subprograms, ...) must be created by the programmer in the module directly. No API Windows functions must be called. It is forbidden to allocate dynamically the memory (**malloc** function, etc.).



All of the specimen source codes shown below are given for C/C++ programming language.

1.3 Required PLC algorithm

The PLC general algorithm requires the execution procedure shown below to be carried out exactly in the prescribed order:

Input loading

In the first phase, register values of all desired inputs must be loaded to internal variables. This applies to the servo drive positions and statuses as well as the values of digital and analog inputs (see Structure Servo and Structure I/O).

Value processing and calculations

The next step consists in processing the loaded values and calculating new values (desired servo drive positions and digital and analog output values).

Output setting

The last step to take consists in sending the desired positions to the servo drives and setting the digital and analog output values through writing the values in the respective registers.

1.4 Communication with PLC

Other components of **TG Motion** system and user applications, which are running under Windows, may communicate with PLC via **TGM_Data** shared memory. **TG Motion** does not intervene into it and does not overwrite its content in any way. The structure of this memory and its utilization are fully at the PLC programmer's discretion.

The size of the **TGM_Data** shared memory is usually 524 288 bytes. The actual size of the shared memory is contained in the **TGM_System.HEADER.Mem_Size_Data** register.

The memory may contain user registers, cam data, etc. It is most frequently used by user visualization applications running under Windows.

2. PLC functions

2.1 Function general description

Just 6 functions are available in the programming module of the virtual PLC module. They differ from each other in the execution priority and the requirements for complete execution during one **Cycle_Time**. For a correct and flawless operation of the virtual PLC, it is essential that the required code be located in the respective function (see below):

Program_Ini() – this function is called only once, namely, when PLC is started.

Program_01() – lowest priority, optional call period.

Program_02() – lower priority, optional call period.

Program_03() – higher priority, optional call period.

Program_04() – highest priority, calling at every Cycle_Time.

Program_05() – highest priority, it is called synchronously with the interpolator position calculation (several times during a single Cycle_Time).



All of these functions must be implemented, some of them may not contain the execution code. Return value of the function must always be provided. See the example code of Program_02().

2.2 Structure PLC_DATA

This structure is used for the communication between **TG Motion** and PLC. **TG Motion** will create 6 instances of the structure **PLC_DATA**. Each of the 6 functions (Program_01–Program_05, Program_Ini) has just 1 parameter, namely, a pointer to the structure **PLC_DATA**. When a function is called, **TG Motion** pass it a pointer to its respective **PLC_DATA** structure.

The structure **PLC_DATA** contains pointers to shared memories and pointers to diagnostic functions.

Diagnostic function definition

```

#ifdef __cplusplus
extern "C" {
#endif
    typedef int __cdecl RTWPRINTF_STRING(LPCWSTR strText);           // string output to console
    typedef int __cdecl RTWPRINTF_LONG(LPCWSTR strFormat, long lVal); // listing of variable value lVal to console
    typedef int __cdecl SWPRINTF(wchar_t *buffer, size_t sizeOfBuffer, const wchar_t *strFormat, ...); // formatted text entry into buffer string
    typedef void __cdecl SLEEPFT(PLARGE_INTEGER Pause);           // waiting
    typedef BOOL __cdecl CAN_TRANSMIT(ULONG Number, ULONG Id, ULONG Dlc, PCHAR Tx_Data ); // reserved
    typedef BOOL __cdecl CAN_TRANSMITREMOTE(ULONG Number, ULONG Id, ULONG Dlc, PCHAR Tx_Data); // reserved
    typedef int __cdecl RTWPRINTF_EX(int severity, const wchar_t *strFormat, ...); // formatted text output to console
#ifdef __cplusplus
};
#endif
  
```

Diagnostic function pointers

```
typedef struct _PLC_IMPORT_FUNCTIONS
{
    RTWPRINTF_STRING    *pRtWprintf_String;
    RTWPRINTF_LONG     *pRtWprintf_Long;
    SWPRINTF            *pSwprintf;
    SLEEPFT            *pSleepFt;
    CAN_TRANSMIT        *pCAN_Transmit;
    CAN_TRANSMITREMOTE *pCAN_TransmitRemote;
    RTWPRINTF_EX       *pRtWprintf_Ex;
} PLC_IMPORT_FUNCTIONS;
```

Whole PLC_DATA structure

```
typedef struct _PLC_DATA
{
    size_t structSize; // structure size in bytes
    void *PSystem_Memory; // pointer to TGM_System shared memory
    void *PData_Memory; // pointer to TGM_Data shared memory
    void *POsc_Memory; // pointer to TGM_Oscilloscope shared memory
    void *PCam_Memory; // pointer to TGM_Cam_Profile shared memory
    void *PServo_Memory; // pointer to TGM_Servo shared memory
    void *PDio_Memory; // pointer to TGM_Dio shared memory
    void *PInterpolator_Memory; // pointer to TGM_Interpolator shared memory
    void *PInterpolator_Memory; // pointer to TGM_Interpolator shared memory
    void *Pointer_interpolator_get_position; // pointer to TGM_InterpolatorReadMemory shared memory
    void *PCNCEX; // pointer to TGM_CNCEX shared memory
    void *PGCode; // pointer to TGM_GCODE shared memory
    void *PReserve3_Memory; // reserved
    void *PReserve4_Memory; // reserved
    void *PReserve5_Memory; // reserved
    void *PReciveDataCan1; // reserved
    void *PReciveDataCan2; // reserved

    PLC_IMPORT_FUNCTIONS functions; // structure with diagnostic function pointers
} PLC_DATA, *PPLC_DATA;
```

2.3 Functions sorted by priority

In this chapter, the functions are sorted from the lowest priority up to the highest one. In the right sense of the word, priority is not concerned in the case of **Program_Ini()** function. This function is called only once, namely, when the Virtual PLC is started up.

Program_Ini

Declaration: **long Program_Ini(PLC_DATA *pData)**

This function is called only once, namely, when the virtual PLC is started up. It is used, above all, to initialize the virtual PLC variables. The execution code of **Program_Ini()** function must not be empty. The execution length of this function is not limited.



When starting the virtual PLC (in the body of Program_Ini function) it is advisable to check the versions of PLC and TG Motion.

Function return values

- 0 – error (PLC startup is aborted, the user must fix the error and restart the PLC).
- 1 – the function processing was OK.



When a virtual PLC is loaded, global variable values are not initialized and global object constructors are not called. The initialization has to be carried out in **Program_Ini** function.

Example

```
long Program_Ini(PLC_DATA *pData)
{
  if (pData == NULL || pData->structSize != sizeof(PLC_DATA))    return 0;
  if (pData->functions.pRtWprintf_Long == NULL)                  return 0;
  if (pData->functions.pRtWprintf_String == NULL)                return 0;
  if (pData->functions.pswprintf == NULL)                        return 0;
  if (pData->functions.pSleepFt == NULL)                         return 0;
  if (pHeader->Compatibility_Id != ID_COMPATIBILITY)
  {
    pData->functions.pswprintf(info_ini, SIZE_INFO, L"Error start of PLC PLC_COMPABILITY_ID = %d
TGM_COMPABILITY_ID = %d \n", ID_COMPATIBILITY, pHeader->Compatibility_Id);
    pData->functions.pRtWprintf_String(info_ini);
    return 0;
  }

  //***** Update PLC Version *****

  Verze_PLC = Get_Version(PLC_VERSION);

  //*****

  return 1;
}
```

Program_01

Declaration: `long Program_01(PLC_DATA *pData)`

The function is called by **TG Motion** with a period specified in the configuration file of **TGMotion4xx.ini**. The period is defined by an item of **Cycle_Time_Program_01**, whose value ranges from 100 μ s to 10 000 μ s (the upper value is not limited). The function execution length should not exceed 20 % of **Cycle_Time_Program_01**, so that a sufficient time is left to execute the functions of **Program_02** and **Program_03**.

This function is used most frequently for basic operation of the devices, which need not to be operated regularly at every **Cycle_Time**.

The **Program_01** function has the lowest priority and may be interrupted by **Program_02**, **Program_03**, **Program_04** and **Program_05** functions at any time.

Program_02

Declaration: `long Program_02(PLC_DATA *pData)`

The function is called by **TG Motion** with a period specified in the configuration file of **TGMotion4xx.ini**. The period is defined by an item of **Cycle_Time_Program_02**, whose value ranges from 100 μ s to 10 000 μ s (the upper value is not limited). The function execution length should not exceed 20 % of **Cycle_Time_Program_02**, so that a sufficient time is left to execute the **Program_01** and **Program_03** functions.

This function is mostly implemented as an empty function.

The **Program_02** function has a low priority and may be interrupted by **Program_03**, **Program_04**, **Program_05** functions at any time.

Example

```
long Program_02(PLC_DATA *pData)
{
    return 1;
}
```

Program_03Declaration: `long Program_03(PLC_DATA *pData)`

The function is called by **TG Motion** with a period specified in the configuration file of **TGMotion4xx.ini**. The period is defined by an item of **Cycle_Time_Program_03**, whose value ranges from 100 μ s to 10 000 μ s (the upper value is not limited). The function execution length should not exceed 20 % of **Cycle_Time_Program_03**, so that a sufficient time is left to execute the functions of **Program_01** and **Program_02**.

This function is mostly implemented as an empty function.

The **Program_03** function has a medium priority and may only be interrupted by **Program_04**, **Program_05** functions at any time.

Example

```
long Program_03(PLC_DATA *pData)
{
    return 1;
}
```

Program_04Declaration: `long Program_04(PLC_DATA *pData)`

The function is called by **TG Motion** synchronously with the EtherCAT communication, i.e., during every **Cycle_Time**, which is defined in **TGMotion4xx.ini** file (250 μ s, 500 μ s, 1000 μ s).

The **Program_04** function is most frequently used to modify the desired servo drive position and to operate I/O units. It has the highest priority (just as **Program_05** function) and is always executed completely, without interruption.

The execution length of **Program_04** function must not exceed 10 % of **Cycle_Time**, so as to secure the time accuracy of the communication with servo drives and I/O units.



***SleepFt** function must not be called in **Program_04** function.*

Program_05Declaration: `long Program_05(PLC_DATA *pData)`

The function is called by **TG Motion** synchronously with the Interpolator, i.e., several times during every **Cycle_Time**.

The **Program_05** function is most frequently used to modify the position, which has been calculated by the **Interpolator**. It has the highest priority (just as **Program_04** function) and is always executed completely, without interruption.

The execution length of **Program_05** function must not exceed 10 μ s, so as to secure the time accuracy of the communication with the servo drives and the I/O units.



SleepFt function must not be called in **Program_05** function.

2.4 Time sequence of function calls

Cycle_Time = 250 μ s

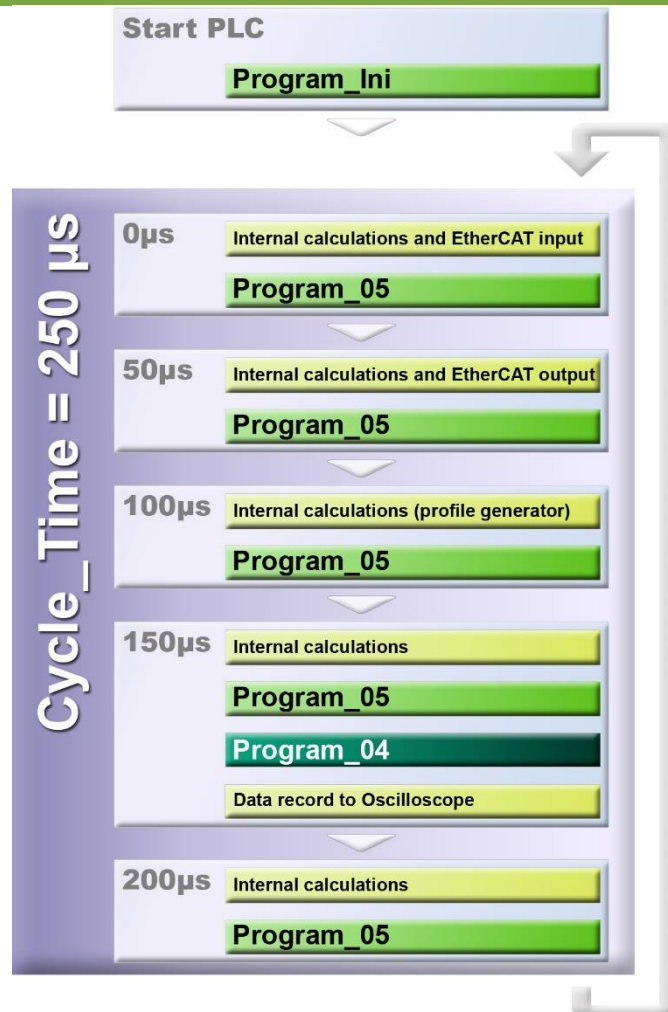


Fig. PLC execution algorithm for Cycle_Time = 250 μ s

After PLC is started and **Program_Ini** function is executed successfully, cyclical calling of a loop lasting 250 μ s is started. The loop is distributed uniformly over 5 equal time intervals, which are called regularly every 50 μ s.

Time = 0 μ s

Necessary internal calculations are carried out and data from EtherCAT are received. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 50 μ s

Necessary internal calculation are carried out and data are sent to EtherCAT. Subsequently, **Program_05** function is called which will be executed completely, without interruption.

Time = 100 μ s

All necessary internal calculations are carried out and profile generator data for all servo drives are calculated. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 150 μ s

Necessary internal calculations are carried out and **Program_05** function is called to be executed completely, without interruption. Subsequently, **Program_04** function is called which will also be executed completely, without interruption. Finally, all data needed for Oscilloscope are recorded.

Time = 200 μ s

Necessary internal calculations are carried out and **Program_05** function is called to be executed completely, without interruption.

If, during any cycle, a free computing capacity is available, **Program_01**, **Program_02** and **Program_03** functions are called according to their cycle time.

Cycle_Time = 500 μ s

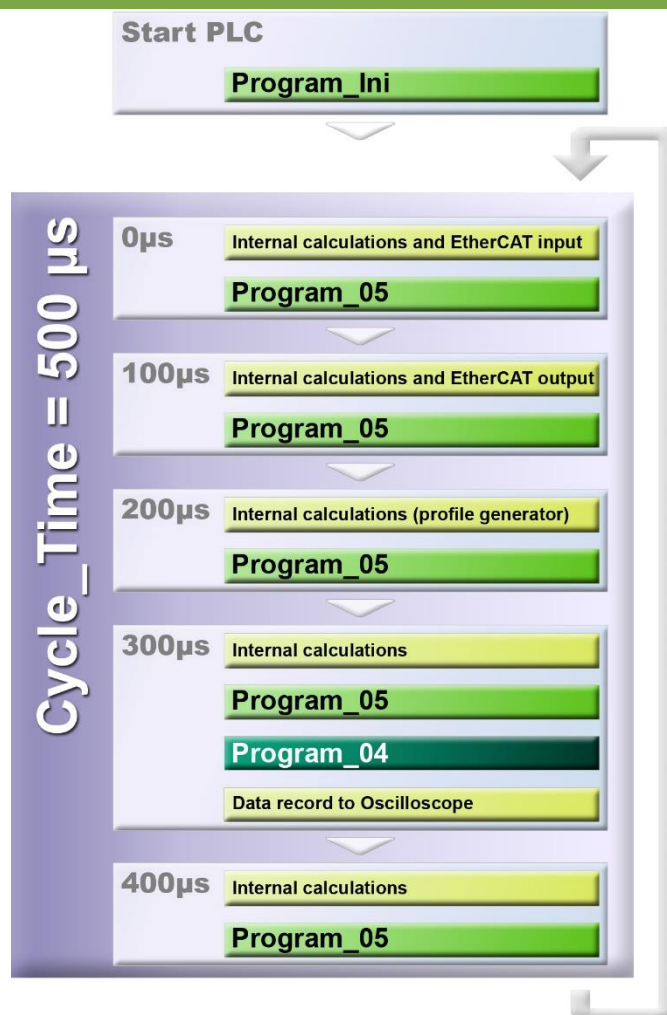


Fig. PLC execution algorithm for Cycle_Time = 500 μ s

After PLC is started and **Program_Ini** function is executed successfully, cyclical calling of a loop lasting 500 μ s is started. The loop is distributed uniformly over 5 equal time intervals, which are called regularly every 100 μ s.

Time = 0 μ s

Necessary internal calculations are carried out and data from EtherCAT are received. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 100 μ s

Necessary internal calculation are carried out and data are sent to EtherCAT. Subsequently, **Program_05** function is called which will be executed completely, without interruption.

Time = 200 μ s

All necessary internal calculations are carried out and profile generator data for all servo drives are calculated. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 300 μ s

Necessary internal calculations are carried out and **Program_05** function is called to be executed completely, without interruption. Subsequently, **Program_04** function is called, which will also be executed completely, without interruption. Finally, all data needed for Oscilloscope are recorded.

Time = 400 μ s

Necessary internal calculations are carried out and **Program_05** function is called to be executed completely, without interruption.

If, during any cycle, a free computing capacity is available, **Program_01**, **Program_02** and **Program_03** functions are called according to their cycle time.

Cycle_Time = 1000 μ s

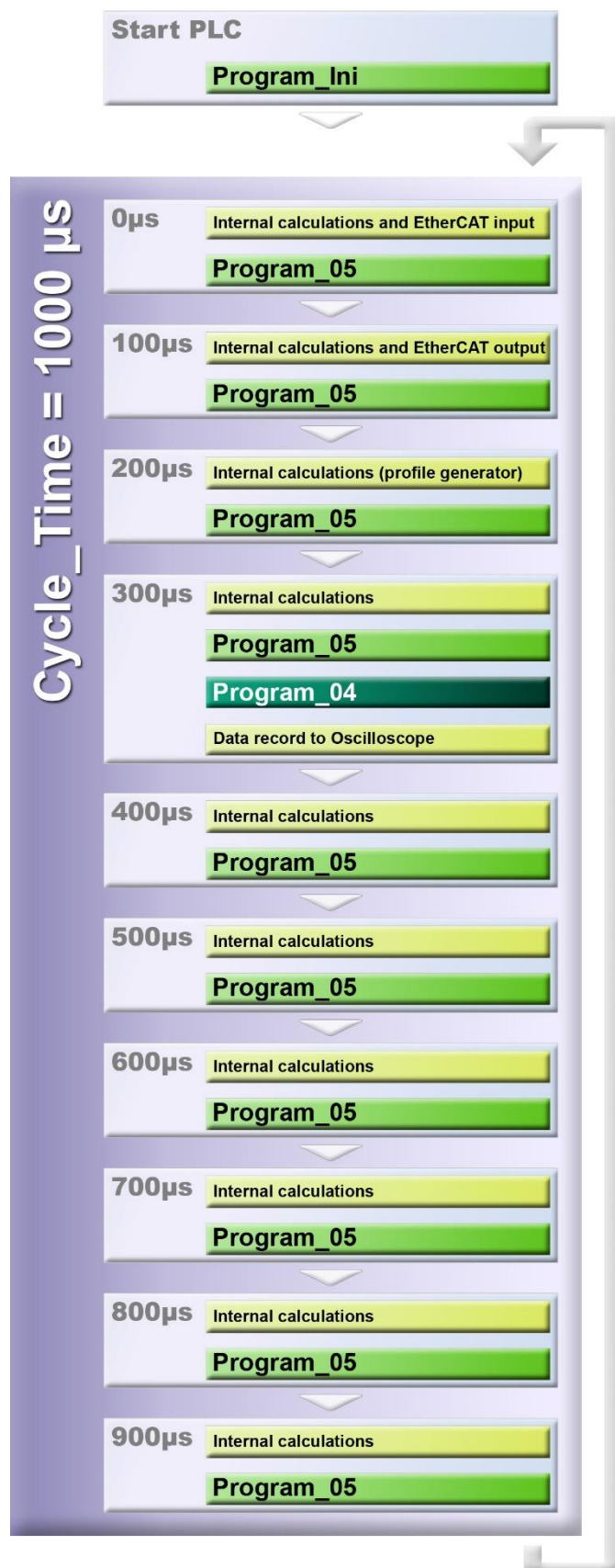


Fig. PLC execution algorithm for Cycle_Time = 1000 μ s

After PLC is started and **Program_Ini** function is executed successfully, cyclical calling of a loop lasting 1000 μs is started. The loop is distributed uniformly over 10 equal time intervals, which are called regularly every 100 μs .

Time = 0 μs

Necessary internal calculations are carried out and data from EtherCAT are received. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 100 μs

Necessary internal calculation are carried out and data are sent to EtherCAT. Subsequently, **Program_05** function is called which will be executed completely, without interruption.

Time = 200 μs

All necessary internal calculations are carried out and profile generator data for all servo drives are calculated. Subsequently, **Program_05** function is called, which will be executed completely, without interruption.

Time = 300 μs

Necessary internal calculations are carried out and **Program_05** function is called to be executed completely, without interruption. Subsequently, **Program_04** function is called which will also be executed completely, without interruption. Finally, all data needed for Oscilloscope are recorded.

Time = 400 μs , 500 μs , 600 μs , 700 μs , 800 μs , 900 μs

Necessary internal calculations are carried out within all of these time intervals and **Program_05** function is called to be executed completely, without interruption.

If, during any cycle, a free computing capacity is available, **Program_01**, **Program_02** and **Program_03** functions called according to their cycle time.

3. PLC debugging tools

3.1 Control Observer

Control Observer is the main debugging tool to operate from Windows environment. It is being supplied with **TG Motion**.

It is a set of utilities, which has been developed for diagnosing the **TG Motion** system, PLC and user Windows application. Control Observer comprises tools for servo drive direct testing and control, PLC code retrieval and System Timer parameter display. Another group of utilities his intended to display, track and change selected shared memory registers.

Control Observer is an independently executable program **Control_Observer_II.exe**, which is being supplied with three libraries:

TGM_Comm_Int_2.dll – secures the communication with TG Motion running on the same computer.

TGM_Mini.dll – is used to connect to TG Motion, which is running in TGMmini, via a LAN network.

TGM_Remote.dll – allows connection to TG Motion, which is running on another computer, via a LAN network.



To access TGM_Data shared memory data, Free_Registers tab in Select_Registers utility, DAT memory type, is used.

Control Observer components

Servo Tester – utility designed to test and control servo drives.

PLC Loader – it is intended to download and start PLC.

System Timer – it displays the actual CPU load caused by each of the TG Motion processes.

Oscilloscope – it provides graphical representation of the time dependence of selected register values.

Graphic Viewer – it is used to provide graphical representation of a continuous series of selected register.



For a more detailed description, refer to Control Observer chapter.

3.2 Listings

Diagnostic functions from the structure **PLC_DATA** are used to output texts to RTX Server console. These diagnostic functions are of three types:

- *string listing into console*
- *variable value listing (long type variable) into console*
- *formatted text listing into console*

3.3 Oscilloscope

Oscilloscope is a stand-alone utility, which is running in **TG Motion**. It is used to capture the values of desired registers within an exact time interval of **Cycle_Time** and their saving into the **TGM_Oscilloscope** shared memory. This happens after **Program_04** function is executed.

Oscilloscope utility of Control Observer is used to record and display captured data.



For a more detailed description, refer to Control Observer chapter.

3.4 Windows applications

Applications running under Windows operating system can also access the shared memories. Register values can be read or changed through these applications.



TG Motion is running under real-time environment. Therefore, its priority is higher than that of the processes running under Windows. It follows that a lossless capture of all required values, or a quick-witted response to an emerging situation, cannot be provided by the Windows applications.