CODESYS Control V3 - Trace Manager

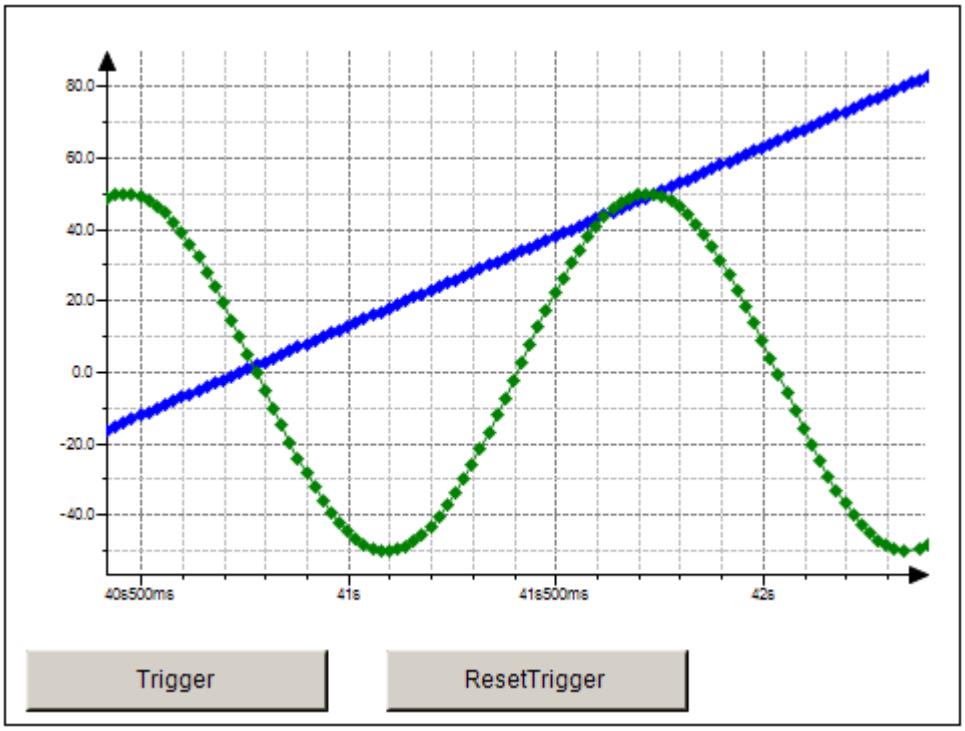
Runtime System Component Description

Version: 6.0
Template: templ_tecdoc_en_V1.0.docx
Dateiname: CODESYSControlV3_TraceManager.docx

CONTENT

1 Overview

The trace functionality is provided with the 'TraceEditor' in CODESYS. It allows to capture the progression of the values of variables on the PLC over a certain time, similar to a digital sampling oscilloscope. Additionally a trigger can be set to control the data capturing with input (trigger) signals. The values of trace variables are steadily written to a CODESYS buffer of a specified size and then can be observed in form of a two-dimensional graph plotted as a function of time.



The configuration of a trace can be found in the CODESYS Online Help.

The Trace Manager ‘CmpTraceMgr’ is a component of the runtime system CODESYS Control V3. This component handles all kind of traces respectively trace configurations.

The Trace Manager has enhanced functionality. It is possible:

- to configure and trace parameters of the control system (like the temperature curve of the CPU or the battery)
- to read out device traces (like the trace of the electric current of a drive)
- to trace system variables of other runtime system components.

A trace configuration is called a “*Trace Packet*”. And a single variable inside a Trace Packet is called a “*Trace Record*” (which is stored in data set that consists of a value and the corresponding timestamp of the shot).

As a Trace Record you can select an IEC variable, a system variable or I/O config variable. For example, the dynamical values like current values of a drive can be stored in a record.

Every component (Runtime or IEC) can define a Trace Packet resp. a Trace Configuration. The in CODESYS integrated Trace Editor uses the same component to store an manage the User Traces.

The trace editor in the CODESYS programming system can display all registered Trace Packets.

The trace configuration and the values can be stored in a file. This feature is called persistent trace. So after a reboot of the controller, the persistent trace configuration will automatically be loaded and restarted.

2 Details of the Trace Manager

For every trace, the configuration can be created in the following ways:

1. The user can create a new Trace Packet and download it to the runtime system
2. An IEC component (POU or Function Block out of an IEC application) can create a Trace Packet (e.g. a fieldbus driver or motion device driver)
3. A Runtime System Component can create a Trace Packet

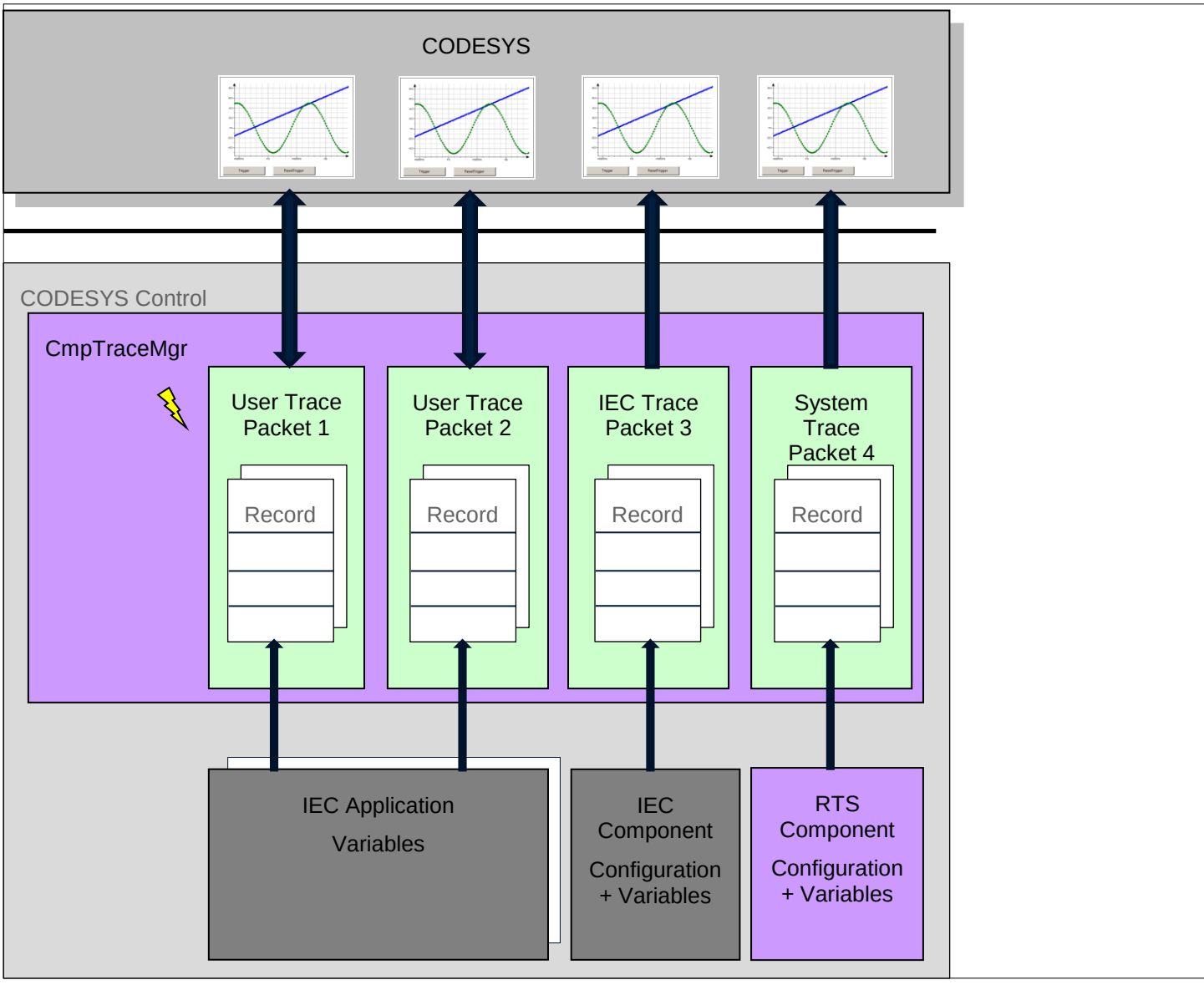
Every Trace Packet is created in the runtime system (CmpTraceMgr).

To get access to IEC or System created Trace Packets, you can get a list of all stored Trace Packets in CODESYS. For this, you have to add a Trace Object right under the Device (not under an application!) and then with the menu command Trace/Upload Trace you can get a list of all registered Trace Packets in the Runtime System.

After a Trace Packet is created, events of the trace manager component are triggered with every operation of the User (‘Start’, ‘Stop’, ‘Upload’, ‘Destroy’, etc.) is done on the Trace Packet.

Reading a trace can be done by connecting a trace in online mode. After that, all data values are cyclically transferred and displayed in the trace editor.

Block diagram CODESYS and trace manager:



The context respectively the task in which the values are captured cyclically inside a Trace Packet is configurable. It can be:

1. An IEC-Task
2. Any System-Task of the Runtime System. This must be specified in the Device Description of a target by the OEM. See the setting “trace \ systemtasks“ in the TargetSettings Documentation!

2.1 Interfaces of the Trace Manager

The Trace Manager Component has 3 different interfaces:

Interface	Function
Component Interface	API for other runtime system components. For detailed description of the interface, see the CmpTraceMgrItf documentation in the Runtime System Reference
Library Interface	The IEC library to access the Trace Manager out of a POU/FunctionBlock/IEC Application. See library documentation of CmpTraceMgr.library for details.
Service Interface (Online)	The online interface for CODESYS

All functional interfaces (component and library) of the trace manager provides the following functions:

- Create a trace packet (with configuration).
- Add a trace record.
- Complete a trace packet (test,that trigger the ‘Configure’ event).
- Delete a trace packet.
- List trace packets.
- Open and close the trace packets (to get the handle)
- Handle the trace packets (‘Start’, ‘Stop’, ‘Reset’)
- Handle the status of a trace packet (degree of filling, trigger received with time stamp)
- Set trigger status of a trace packet
- Acitvate/deactivate the capturing of data
- Acitvate/deactivate the trigger handling of a trace packet
- Write/update data of a trace (n set of data + time, alternatively n set of data with start time and interval)
- Read configuration of a trace packet
- Read data of a trace
- Save trace packet in a file (with and without data)
- Read trace packet from a file (create)

2.2 Usage

2.2.1 General

To define a trace, first you have to create a Trace Packet:

```
RTS_HANDLE TraceMgrPacketCreate(TracePacketConfiguration *pConfiguration, RTS_RESULT *pResult);
```

The trace packet needs some configuration entries:

```
/**
 * Configuration of a trace packet
 */
typedef struct tagTracePacketConfiguration
{
   RTS_IEC_STRING *pszName;           /* The name of the trace packet */
   RTS_IEC_STRING *pszApplicationName;/* The name of the application (optional) */
   RTS_IEC_STRING *pszIecTaskName;    /* IEC-task name in which the samples are recorded (optional) */
   RTS_IEC_STRING *pszComment;        /* A comment for the packet (optional) */
   struct tagTraceTrigger *pttTrigger; /* Pointer to a trigger description (optional) */
   struct tagTraceVariable *ptvCondition; /* A pointer to the description of a boolean variable.
                                             If given, samples are recorded only if the variable has
```

```
                                        value true. (optional, must be present if TRACE_PACKET_FLAGS
                                         CONDITION is set in ulFlags) */
   RTS_IEC_UDINT ulEveryNCycles;  /* Record samples every ulEveryNCycles cycles.  Must be > 0.
                                    (Default: 1) */
   RTS_IEC_UDINT ulBufferEntries;     /* The number of samples that the trace buffer can hold. */
   RTS_IEC_UDINT ulFlags;             /* Trace packet flags.  See TRACE_PACKET_FLAGS. */
   /* The interface, starting from here is only used in C and not
    * exported to IEC.
    */
   RTS_GUID ApplicationDataGuid;
} TracePacketConfiguration;
```

Note: If you specify an ApplicationName and an IecTaskName, the trace records (values) are recorded at the end of this IEC-task cycle! If nothing is specified here, the trigger of recording the values must be done by the owner of the Trace Packet via the functions `TraceMgrRecordUpdate/2/3()`!

After the creation of a Trace Packet you can add single trace records (trace variables) with the following function:

```
RTS_HANDLE TraceMgrRecordAdd(RTS_HANDLE hPacket, TraceRecordConfiguration *pConfiguration, RTS_RESULT
*pResult);
```

A record must be configured too:

```
/**
 * Trace record configuration
 *
 * The colors (ulGraphColor, ulMinWarningColor, ulMaxWarningColor)
 * and are encoded in UDINTs in the ARGB format: the most significand
 * byte is the alpha value, the next byte the red value, followed by
 * green and blue.  Each color component takes a value between 0 and FF.
 * So 0xFF000000 is white, 0xFFFFFFFF is black, 0xFFFF0000 is red,
 * 0xFF00FF00 is green, and 0xFF0000FF is blue.
 */
typedef struct tagTraceRecordConfiguration
{
   RTS_IEC_STRING *pszVariable;       /* Name of the variable */
   TraceVariableAddress tvaAddress;   /* Address definition of the variable */
   RTS_IEC_UDINT tcClass;             /* Type class of the variable. See enum IBase.TypeClass
                                      for the possible values. */
   RTS_IEC_UDINT ulSize;              /* Size in bytes of a single sample */
   RTS_IEC_UDINT ulGraphColor;        /* Color in which the trace curve for the
                                      variable should be displayed */
   RTS_IEC_UDINT ulGraphType;         /* Graph type (1: line (with points),
                                      2: cross, 4: step (with points), 5: point, 8: line (without poin
                                      ts), 9: step (without points), 10: line (with crosses), 11: step
                                      s (with crosses)) */
   RTS_IEC_UDINT ulMinWarningColor;   /* Color to use if a sample
                                       is <= fCriticalLowerLimit */
   RTS_IEC_UDINT ulMaxWarningColor;   /* Color to use if a sample
                                      is >= fCriticalUpperLimit */
   RTS_IEC_REAL fCriticalLowerLimit;  /* The lower limit */
   RTS_IEC_REAL fCriticalUpperLimit;  /* The upper limit */
   RTS_IEC_BOOL bActivateMinWarning;  /* If set, the trace  will be displayed in the
                                       color ulMinWarningColor as soon as a sample is <= fCriticalLowe
                                      rLimit */
   RTS_IEC_BOOL bActivateMaxWarning;  /* If set, the trace  will be displayed in
                                       the color ulMaxWarningColor as soon as a sample is >= fCritical
                                      UpperLimit */
   RTS_IEC_BYTE byYAxis;              /* Not used */
} TraceRecordConfiguration;
```

If all records are added, you have to complete the trace packet with the following function:

```
RTS_RESULT TraceMgrPacketComplete(RTS_HANDLE hPacket);
```

After that the trace configuration is completed.

Then the trace must be explicitly started to begin recording. Thismust be done with the following function:

```
RTS_RESULT TraceMgrPacketStart(RTS_HANDLE hPacket);
```

Recording of the values must be done cyclically via calling the `TraceMgrRecordUpdate/2/3()` function. This can be triggered out of a system task or if an IEC-Task is specified in the trace packet configuration, you get an event (EVT_TRACEMGR_UPDATE_RECORD from CMPID_CmpTraceMgr) at the end oft he IEC-task cycle.

2.2.2 IEC Component

To use the TraceManager out of an IEC application, you can use the CmpTraceMgr.library. An example for the usage of the Trace Manager can be found at the CODESYS Store (store.codesys.com).

2.2.3 Runtime System Component

To use the TraceManager out of a Runtime System Component you can use the CmpTraceMgrItf. The Reference of this API can be found in the Reference Documentation. An example can be found at the Runtime Delivery Starter Package under $/Templates/CmpTraceMgrOEM.

2.3 Specification of the Memory Format of the Trace File

2.3.1 Trace Packet of a Trace File

A Trace Packet can be stored permanently on the Runtime System. This feature can be enabled in the Trace Packet Configuration with the option “persistent Trace”. If this optin is selected, the Trace Configuration (and the values if available) are stored in a so called Trace File.

The format of a trace file is CSV to be portable. This file can be uploaded to CODESYS and offline be loaded.

The format consists of a header part and a data part. The header part of the variables is defined in two columns. The first one is used as “key”, the second one is used as “value”. The keys are always listed with dot notation with full path. The structure is in substance similar to the trace configuration passed from the programming system to the runtime system. The structure is essentially conform to the trace configuration as it is transmitted from the programming system to the runtime system.

Example:

```
[key]; [value]
Name; TestPacket
Trigger.Variable.Name; hugo
Trigger.Variable.AddrFlags; 0x00000000
Trigger.Variable.Class; 0
Trigger.Variable.Size; 0
Trigger.Level; 0
```

Trace variables require different parameters depending on the address flags.

TRACE_VAR_ADDRESS_FLAGS_AREA_OFFSET:

(Convert IEC Pointer to Area Offset)

- **Name** (string)
- **AddrFlags** (hex number)
- **Area** (unsigned integer)
- **Offset** (hex number)
- **Class** (unsigned integer)
- **Size** (unsigned integer)

TRACE_VAR_ADDRESS_FLAGS_POINTER:

- **Name** (string)
- **AddrFlags** (hex number)
- **Address** (hex number)
- **Class** (unsigned integer)
- **Size** (unsigned integer)

TRACE_VAR_ADDRESS_FLAGS_IOCONFIG:

- **Name** (string)
- **AddrFlags** (hex number)
- **ParameterID** (unsigned integer)
- **ModuleType** (unsigned integer)
- **Instance** (unsigned integer)
- **ByteOffset** (hex number)
- **Class** (unsigned integer)

- **Size** (unsigned integer)

TRACE_VAR_ADDRESS_FLAGS_PROPERTY:

- **Name** (string)
- **AddrFlags** (hex number)
- **Instance.Area** (unsigned integer)
- **Instance.Offset** (hex number)
- **PropertyFunction.Area** (unsigned integer)
- **PropertyFunction.Offset** (hex number)
- **Class** (unsigned integer)
- **Size** (unsigned integer)

2.3.2 Trace Records of a Trace File

The records are numbered from 0 to n and the number is set as prefix before the name (e.g. „0.Name; Hugo“). After the header, the data are following. In order to avoid redundancy, no further key name is added.

The first entry is a key without value:

```
<number>.Data;
```

The values itself consist of:

```
<empty field>; <time stamp>; <value>
```

Example:

```
0.Data;
; 2490; 231
; 2500; 232
```

Listing of a record with all possible entries (the optional fields are written in gray):

```
[key]; [value]
0.Name; IecTest
ApplicationName; <Name der Applikation>
ApplicationDataGuid; aaaaaaaa-bbbbbbbb-cccccccc-dddddddd
IecTaskName; <Name der IEC Task>
Comment;
Trigger.Variable.Name; <Name der Trigger Variablen>
Trigger.Variable.AddrFlags; 0x00000011
Trigger.Variable.Area; 0
Trigger.Variable.Offset; 0x00000000
Trigger.Variable.Address; 0x00000000
Trigger.Variable.ParameterID; 0
Trigger.Variable.ModuleType; 0
Trigger.Variable.Instance; 0
Trigger.Variable.ByteOffset; 0x00000000
Trigger.Variable.Instance.Area; 0
Trigger.Variable.Instance.Offset; 0x00000000
Trigger.Variable.PropertyFunction.Area; 0
Trigger.Variable.PropertyFunction.Offset; 0x00000000
Trigger.Variable.Class; 0
Trigger.Variable.Size; 1
Trigger.Level; 0
Trigger.Flags; 0
Trigger.Edge; 1
Trigger.Position; 0
Condition.Name; (null)
Condition.AddrFlags; 0x00000000
Condition.Class; 0
Condition.Size; 0
EveryNCycles; 1
BufferEntries; 600
Flags; 5


0.Variable; DWORD1
0.Address.AddrFlags; 0x00000011
```

```
0.Address.Area; 0
0.Address.Offset; 0x00000000
0.Address.Address; 0x00000000
0.Address.ParameterID; 0
0.Address.ModuleType; 0
0.Address.Instance; 0
0.Address.ByteOffset; 0x00000000
0.Address.Instance.Area; 0
0.Address.Instance.Offset; 0x00000000
0.Address.PropertyFunction.Area; 0
0.Address.PropertyFunction.Offset; 0x00000000
0.Class; 4
0.Size; 4
0.GraphColor; 4278190335
0.MinWarningColor; 0
0.MaxWarningColor; 0
0.CriticalLowerLimit; 0.000000
0.CriticalUpperLimit; 0.000000
0.ActivateMinWarning; 0
0.ActivateMaxWarning; 0
0.YAxis; 0
0.Data;
; 2490; 231
; 2500; 232
; 2510; 233
; 2520; 234
; 2530; 235
; 2540; 236
; 2550; 237
etc.
```

3 Target Settings for the device description

3.1 Enable Trace Mgr

An entry in the device description file defines whether the device will use the functionality of the trace manager or the internal IEC Trace.

For using the Trace Manager, the CmpTraceMgr must be integrated in the Runtime System and the following Target Setting must be added to the Device Description:

```
<section name="trace">
    <setting name="tracemanager" type="boolean" access="visible">
  <value>1</value>
    <setting>
</section>
```

3.2 System Tasks

Additionally, task names for using as Trace tasks can be defined In the target description. Then the names are available in the configuration dialog of the programming system for configuration a Trace Packet:

```
<ts:section name="trace">
  …
  <ts:setting name="systemtasks" type="string" access="visible">
            <ts:value>SystemTask1,SystemTask2,SystemTask3</ts:value>
  </ts:setting>
</ts:section>
```

4 Glossary

Trace Packet	A trace or trace packet consists of a configuration and its data sets. The configuration consists of a context (task), perhaps a trigger, some options and at least one variable which should be traced (record)
Trace Record	A trace record is a data set of one variable that should be traced. This data set contains, the a ring buffer with a value/timestamp pair and some options to record this variable

Bibliography

[1] Runtime System Online Help

[2] CODESYSControlV3_Manual.pdf

Change History

Version	Description	Author	Date
0.1	Issued	AS	28.03.2011
0.2	Review	AH	29.03.2011
1.0	Release after formal Review	AS	29.03.2011
2.0	Release after formal change (CoDeSys -> CODESYS)	MN	05.12.2012
3.0	New doc tempate applied	MN	31.01.2014
3.1	[CDS-17044] Complete rewriting of this documentation	AH	29.06.2015
4.0	Release	AH	22.07.2015
4.1	Added legal note	TZ	23.08.2016
5.0	Release after formal review	MaH	23.08.2016
5.1	Legal note removed	GeH	24.10.2016
6.0	Release after formal review	MN	28.11.2016