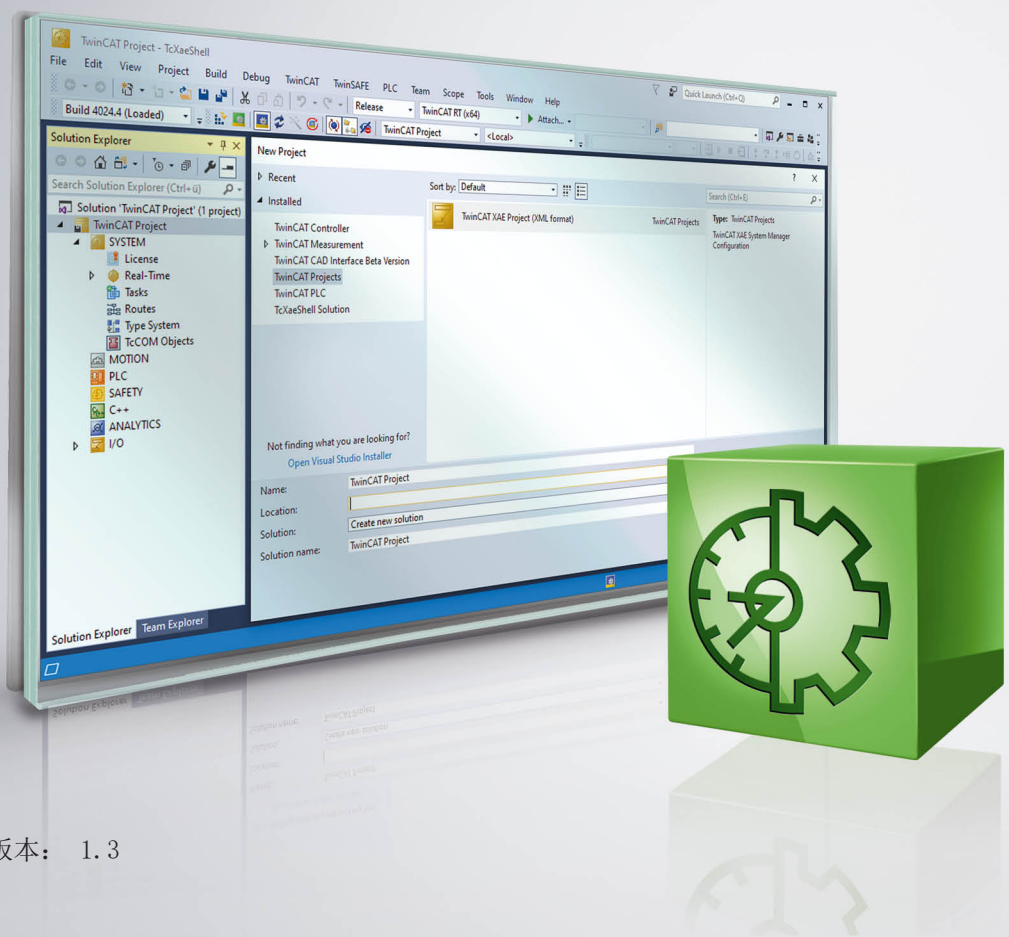
BECKHOFF New Automation Technology

手册 | ZH

TE1000

TwinCAT 3 | EventLogger



2020/10/13 | 版本： 1.3

目录

1 前言

1.1 文档说明

本说明仅供熟悉适用国家标准的控制和自动化工程专家使用。
在安装和调试元器件时，必须遵循本文档及以下注意事项和说明。
技术人员应负责在每次安装和调试时使用已发布的文档。

负责人员必须确保所述产品的应用或使用符合所有安全要求，包括所有相关法律、法规、准则和标准。

免责声明

本文档经过精心准备。然而，所述产品正在不断开发中。
我们保留随时修改和更改本文档的权利，恕不另行通知。
不得依据本文档中的数据、图表和说明对已供货产品的修改提出赔偿。

商标

Beckhoff®、TwinCAT®、EtherCAT®、EtherCAT G®、EtherCAT G10®、EtherCAT P®、Safety over EtherCAT®、TwinSAFE®、XFC®、XTS® 和 XPlanar® 均为倍福自动化有限公司的注册商标并由公司授权使用。
本出版物中使用的其他名称可能是商标，第三方出于自身目的使用它们可能侵犯商标所有者的权利。

正在申请的专利

涵盖 EtherCAT 技术，包括但不限于以下专利申请和专利：：
EP1590927、EP1789857、EP1456722、EP2137893、DE102015105702
包括在其他各国家的相应专利申请或注册。

EtherCAT® 是注册商标和专利技术，由德国倍福自动化有限公司授权使用

版权所有

1.2 安全说明

安全规范

请注意以下安全说明和阐述！
可在以下页面或安装、接线、调试等区域找到产品相关的安全说明。

责任免除

所有元器件在供货时都配有适合应用的特定硬件和软件配置。禁止未按文档所述修改硬件或软件配置，德国倍福自动化有限公司不对此承担责任。

人员资格

本说明仅供熟悉适用国家标准的控制、自动化和驱动工程专家使用。

符号说明

在本文档中，下列符号随安全指示或说明一起使用。必须仔细阅读并严格遵守安全说明！

⚠ 危险

严重受伤的风险！

未遵守带有此符号的安全说明将直接危及人员生命和健康。

⚠ 警告

受伤的风险！

未遵守带有此符号的安全说明将危及人员生命和健康。

⚠ 谨慎

人身伤害！

未遵守带有此符号的安全说明可能导致人员受伤。

注意

危害环境或损坏设备

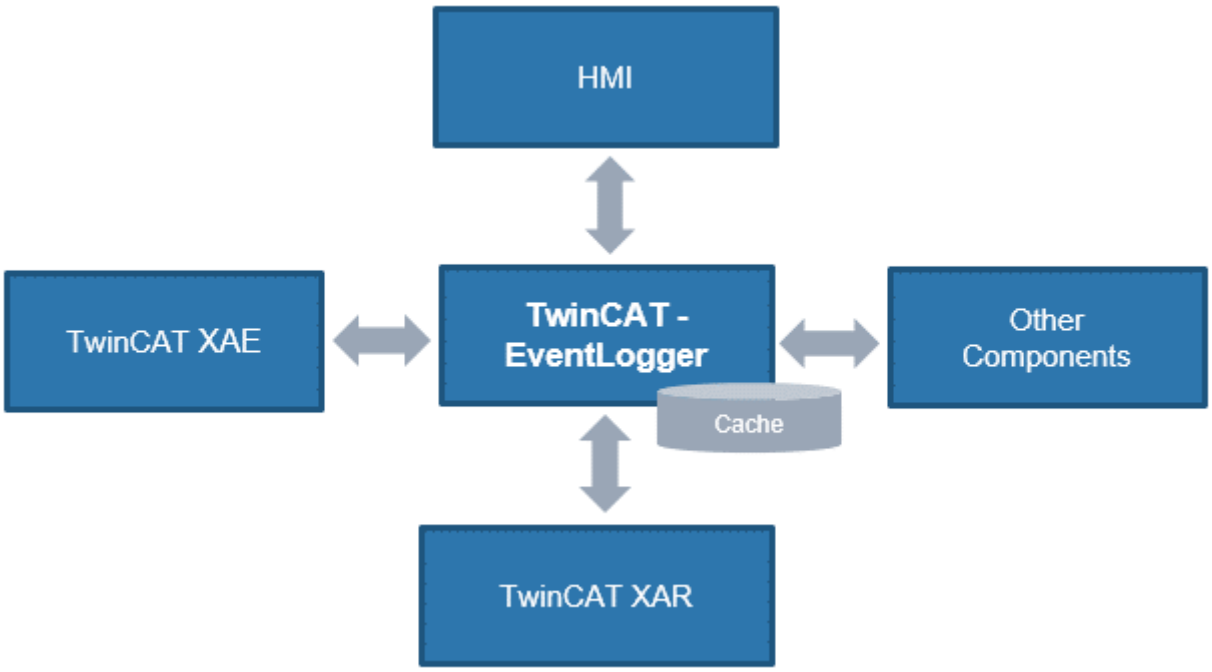
未遵守带有此符号的安全说明可能危害环境或损坏设备。

提示或指示

此符号表示该信息有助于更好地理解。

2 概述

TwinCAT 3 EventLogger 提供了用于在 TwinCAT 组件和非 TwinCAT 组件之间交换消息的接口。



本文档适用于 TwinCAT 3 EventLogger 的用户。有关如何通过不同的 TwinCAT 3 组件使用 TwinCAT 3 EventLogger 的信息，请参见相应的产品文档。

产品组件

TwinCAT 3 EventLogger 的所有组件均包含在相应的基本安装程序中。

TwinCAT 3 Engineering 中包含多个用于编程的接口。必要的模块和库是相应运行时的一部分。

使用 TwinCAT 3 EventLogger 无需授权费。

3 系统要求

技术数据	要求
操作系统	Windows 7/10、Windows Embedded Standard 7、Windows CE 7
目标平台	PC 体系结构 (x86、x64 或 ARM)
TwinCAT 版本	TwinCAT 3.1 Build 4022.20 或更高版本
所需 TwinCAT 设置级别	TwinCAT 3 XAE、XAR
所需 TwinCAT 授权	任何运行时授权 (PLC、C++)
Visual Studio 版本	可通过 Visual Studio 2013 版本使用 TwinCAT 3 EventLogger 的组件。

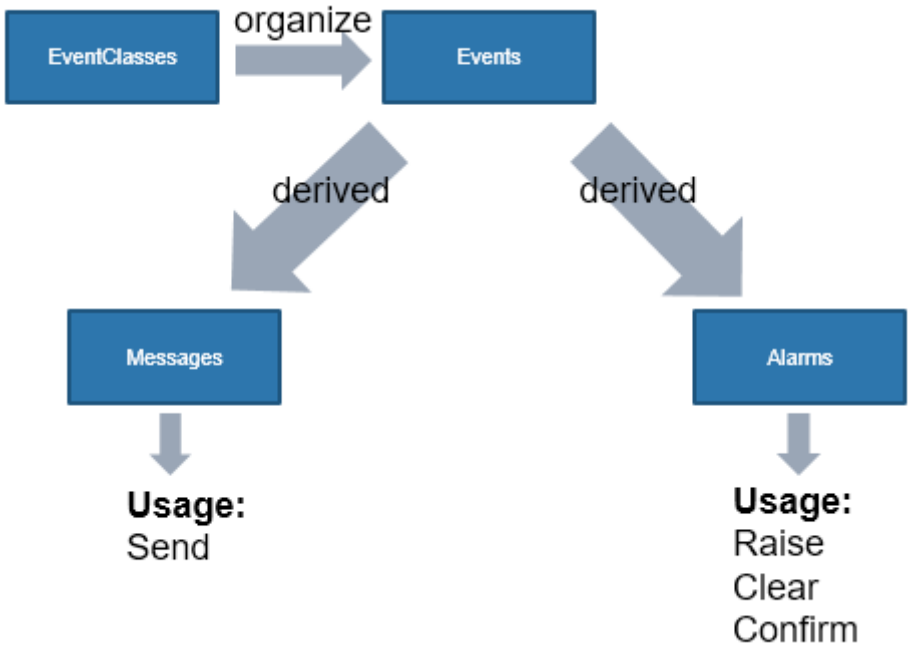
4 限制

- 传输事件时的最大大小为 8 kb。使用 TwinCAT 3 EventLogger 时，请确保遵守此限制。这适用于本文档中传输和描述的所有元素，包括动态元素 (Attributes、SourceName、JSON Attributes)。
- 用于实时接收事件的接口最多临时存储 1024 个事件，直到必须对其进行检索为止。
  如果未及时检索，事件将丢失。
- TwinCAT 3 EventLogger 提供了用于连接 TwinCAT HMI (TF2xxx) 的接口。而 TwinCAT PLC HMI (TF18xx) 无法接收事件。

5 技术简介

TwinCAT 3 EventLogger 传输所谓的事件。事件是指消息或警报。

本技术简介着重介绍作为事件内容传输的数据，因为它们对于理解此过程是必不可少的。事件的发送和接收在 PLC 和 C++ 的 API 描述以及示例中进行详细说明。



事件

并不直接使用事件本身，而是使用派生的类型“消息”或“警报”。

事件提供消息和警报的以下常见元素：

EventClass (GUID)	事件类是事件组（可能用于主题）。
Event-ID (UDINT)	事件在事件类中由 Event-ID 明确标识。
文本 (String)	事件描述。描述面向人提供，因此也可以国际化。可以在运行时插入参数，以使文本能够单独修改。
Source Info	事件发生的来源。来源包括三个元素。可以根据需要使用，而对于如何使用，提供了相应的建议。 • Source-ID (INT)：TcCOM 对象 ID • Source Name (STRING)：TcCOM 对象内的路径，例如 PLC 项目中函数块的路径。 • Source-GUID (GUID)：例如，可用于标识项目或（子）产品。
JSON Attribute (STRING)	可以根据需要使用。“文本”（见上文）面向作为接收者的人提供，而 JSON 属性用于程序化接收。可以轻松创建（序列化）和处理（反序列化）JSON string。TwinCAT 为此提供实时的 JsonXml 库（请参见 Documentation PLC library Tc3_JsonXml）。

除了这些元素之外，事件还包含其他元素，均在 TMC 编辑器 [▶ 12]中进行描述。

消息

消息是无状态的。它们在调用时被发送，并发送相应的注册组件。

标识

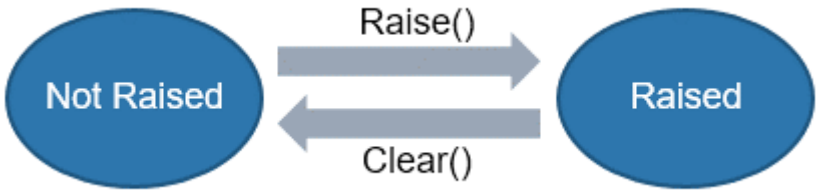
消息由 EventClass 和 Event-ID 标识。

警报

与消息相反，警报是有状态的。

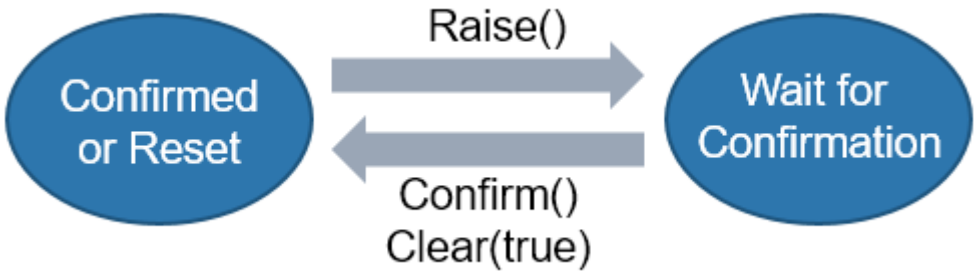
它具有以下警报状态：

- Not-Raised（未触发）
- Raised（已触发）



另外，还可能需要确认。以下确认状态之间有所区别：

- WaitForConfirmation（待确认）
- Confirmed（已确认）或 Reset（重置）



调用相应的方法会导致发送事件，并将警报设置为相应的状态。

- 如果通过 Confirm() 进行了确认，则状态设置为 Confirmed（已确认）。
- 如果通过 Clear(TRUE) 进行了确认，则状态设置为 Reset（重置）。

如果在当前状态下对方法的调用无效，则将通过返回值对此进行指示。

当关闭 TwinCAT（RUN（运行）→ CONFIG（配置）转换）时，将在内部为处于 Raised（已触发）状态的所有警报执行设置 Clear（清除）时间戳处理。未对这些警报进行确认。

标识

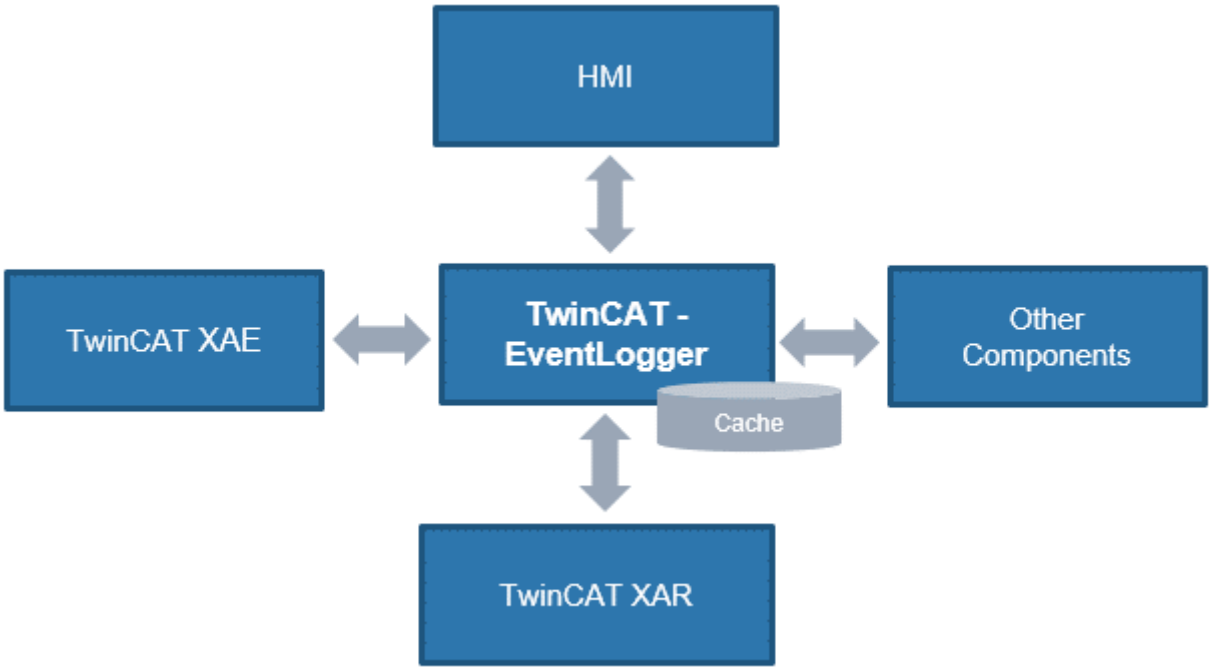
TwinCAT 3 EventLogger 使用 EventClass、Event-ID 和 Source Info 标识警报。因此，可以在程序的不同点使用警报（EventClass 和 Event-ID 的组合）。例如，警报“storage empty（存储空间不足）”可用于不同的存储设备，因为运行时会提供不同的“Source Info（源信息）”（另请见处理源 [▶ 24]）。

体系结构

TwinCAT 3 EventLogger 在其他组件之间集中传输事件。这些组件包括作为主要事件源的实时编程接口 PLC 或 C++。

在开发过程中，消息可以显示在 TwinCAT Engineering（XAE）中。

例如，HMI 可以接收消息并相应地进行显示。客户可以自行创建更多组件来接收事件。



TwinCAT 3 EventLogger 在缓存中保留有限数量的以往事件。例如，可以在重新启动后通过 Engineering 对此进行查询，以便进行诊断。缓存与计算机的安全关闭有关。

工作流程

在组件之间建立异步通信的一般工作流程如下：

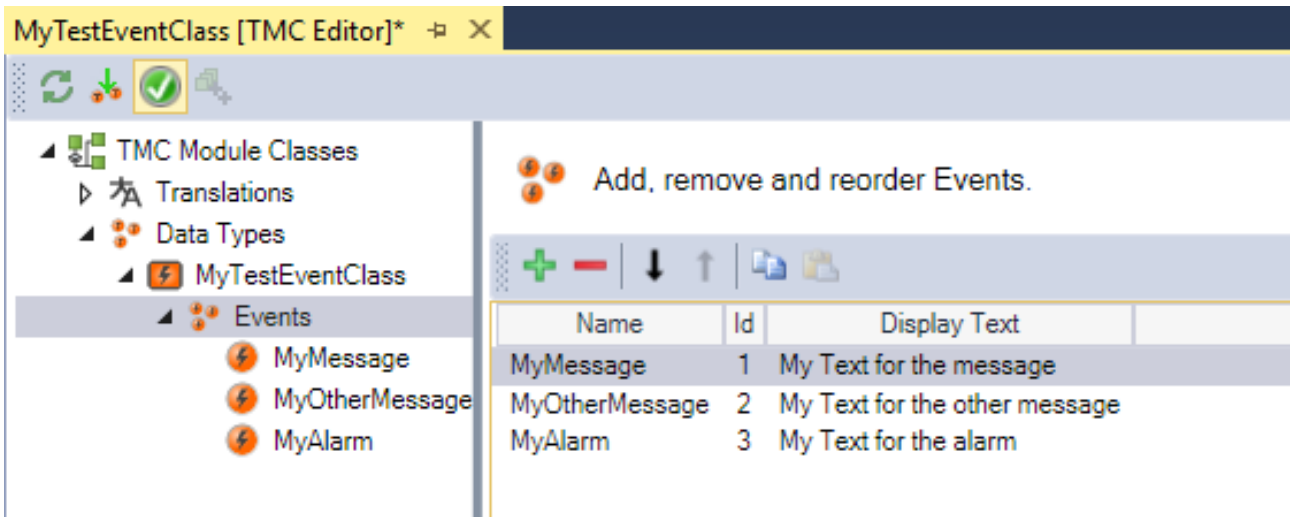
1. 创建新 TwinCAT 项目。
2. 在 TwinCAT 类型系统中定义事件类和事件。
3. 执行自动代码生成，以便为 TwinCAT 中的实时编程语言提供源代码。
4. 实现事件的使用及发送和接收。

另请参见：

- 文档 TwinCAT 3 类型系统
- API 描述 PLC [▶ 27] 和 C++ [▶ 83]
- 示例 [▶ 139]

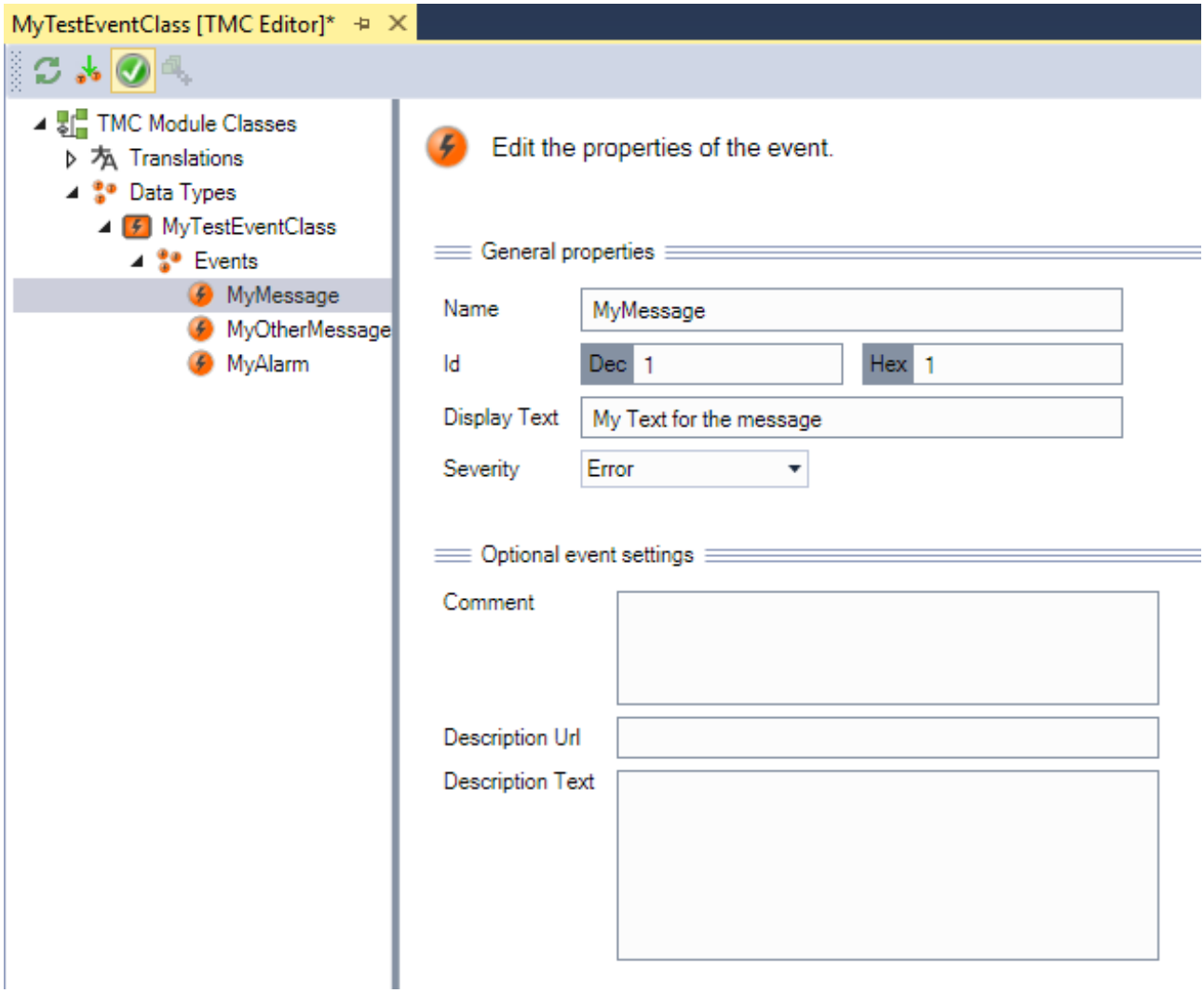
5.1 事件

事件在事件类中进行描述。



可以通过子元素相应地配置事件。

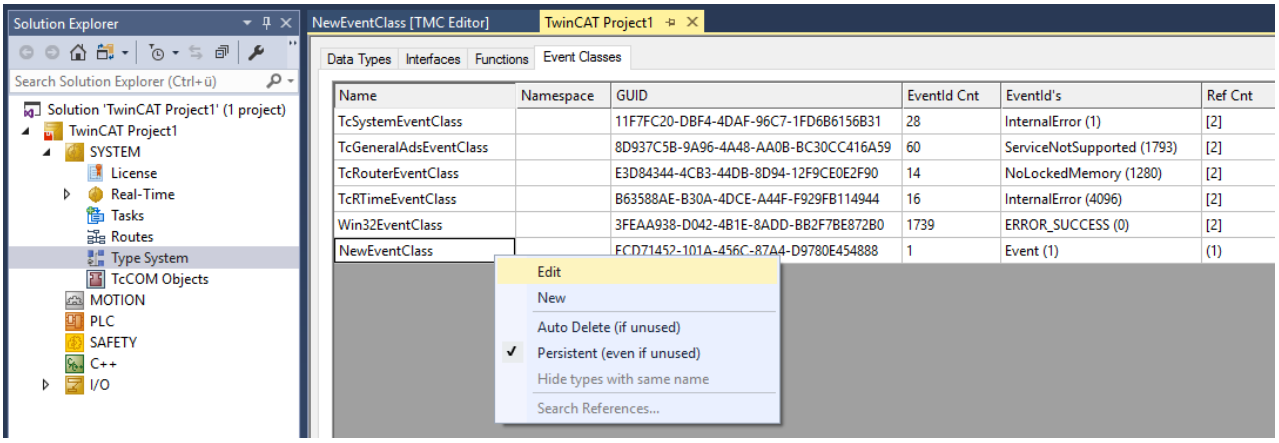
事件属性



名称	此名称表示事件，例如在生成的源代码中。
ID	在事件类中明确标识事件。
显示文本	该文本用于表示显示的事件类。可以国际化（请参见国际化/翻译 [▶ 17]）。
严重级别	事件的严重级别。这由生成的源代码提供，因此表示默认行为；但是，也可以在此处使用其他值： • Verbose • Info • Warning • Error • Critical

5.2 事件类

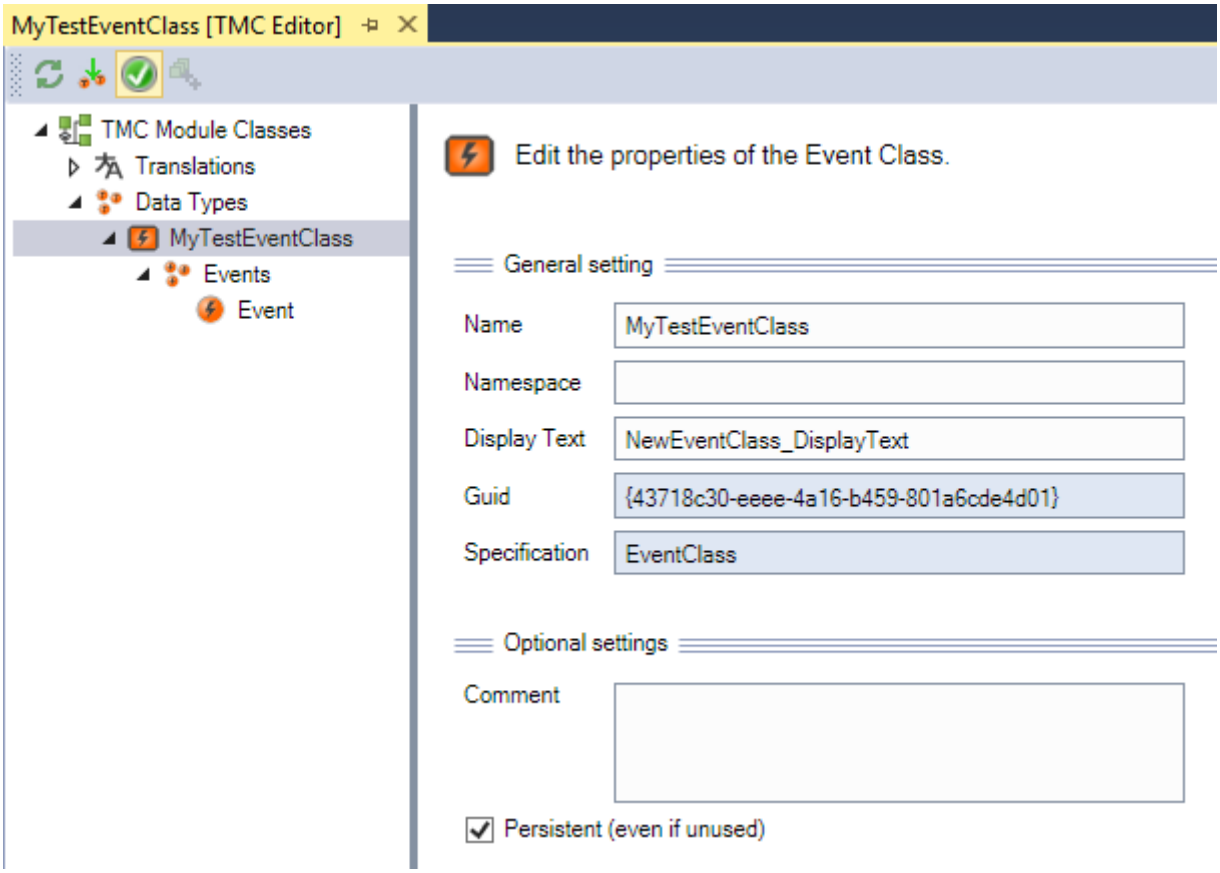
事件类是事件组（可能用于主题），而对于 TwinCAT 类型系统而言，则为可以在不同模块中使用的数据类型。因此，它们在 TwinCAT 类型系统（System（系统） > Type System（类型系统） > Event Classes（事件类））中作为数据类型进行创建。



TwinCAT 类型系统的 **Event Classes（事件类）** 选项卡下列出了所有已知事件类。可以通过上下文菜单命令 **Edit（编辑）** 和 **New（新建）** 打开 TMC 编辑器，并可在其中定义和编辑事件类。

除项目事件类外，TwinCAT 还提供其他事件类，例如 ADS 返回代码和系统事件。

事件类的属性



名称	此名称表示事件类，例如在生成的源代码中。
命名空间	像所有数据类型一样，事件类也可能属于命名空间。
显示文本	该文本用于表示显示的事件类。可以国际化（请参见国际化/翻译 [▶ 17]）。
Guid	像所有数据类型一样，它标识当前描述的事件类。它是自动计算的，并且在事件类中发生更改时发生更改。

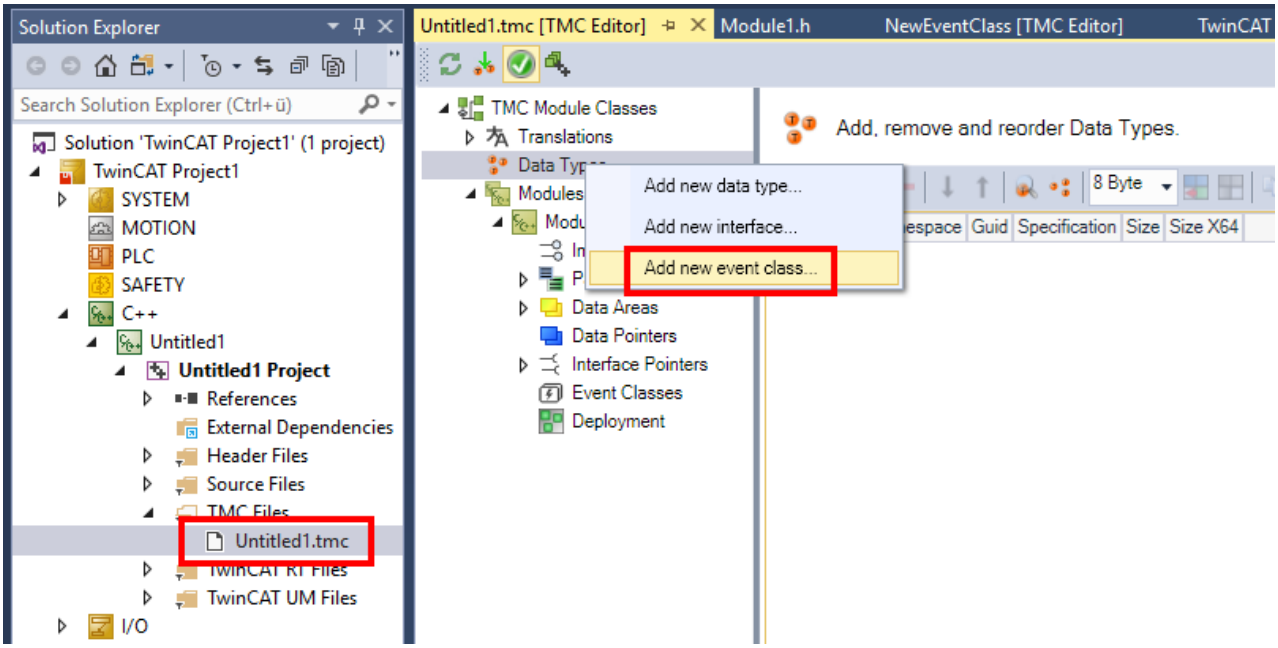
PLC 库中的事件类

如果在 PLC 库中使用了事件类，则事件类也应该成为该库的一部分，以确保数据类型包含在每个使用 PLC 库的应用程序的 TwinCAT 类型系统中。

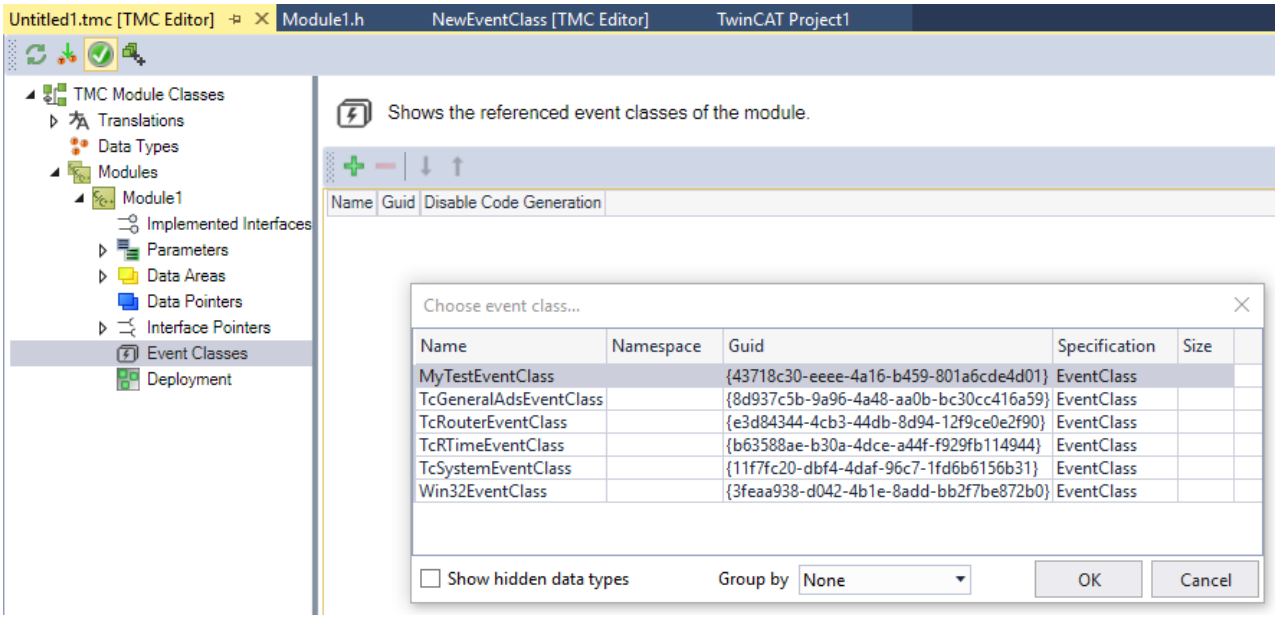
为此，必须在 PLC 库中“锁定”事件类。为此，请在 PLC 库对象中 **External Types（外部类型）** 下创建的数据类型上下文菜单中选择 **Pin Global Data Type（锁定全局数据类型）** 命令。

C++ 项目中的事件类

与其他 TwinCAT 数据类型类似，对于 C++ 项目，也可以在 C++ 项目中对事件类进行本地定义。



声明事件类以在相应的 C++ 模块中使用（无论它们在何处定义）。如果将事件类添加到 C++ 模块，它将自动嵌入到模块的 TMC 文件中。



5.3 事件定义的代码生成

在 PLC 和 C++ 中均根据事件类及事件的定义生成源代码。

事件“名称”和事件类用于代码生成。因此，事件类可以通过其名称在事件类的不同版本中使用。

“严重级别”通过代码生成提供。因此，程序员可以在创建事件时单独设置。因此，应将 TMC 编辑器 [▶ 12]中描述的严重级别视为默认行为。在特定的应用案例中，严重级别可能与此不同。

PLC

GVL TC_EVENTS 在 PLC 中创建，具有作为子元素的事件类，并在更改后进行更新（保存/关闭 TMC 编辑器）。这些全局常数又将作为子元素的事件本身，以及它们所属的事件类的各个元素 EventId、Severity 和 UUID 包含在内。

可以通过 IntelliSense 将这些元素用于参数，例如使用 Create()/CreateEx()。

登录 PLC

如果与 PLC 进行了连接（登录），则不会更新代码。然后在注销后进行更新。

C++

TcCOM 模块必须使用事件类，即必须按照在 TMC 编辑器中使用的方式输入。然后，代码生成将创建一个命名空间“TcEvents”，作为 <DriverName>Services.h- 文件的一部分。可以通过 IntelliSense 将命名空间用于事件类/事件的参数，例如通过 CreateMessage()/CreateAlarm()。

兼容性

C++ 源代码生成需要 Visual Studio 2013 或更高版本。

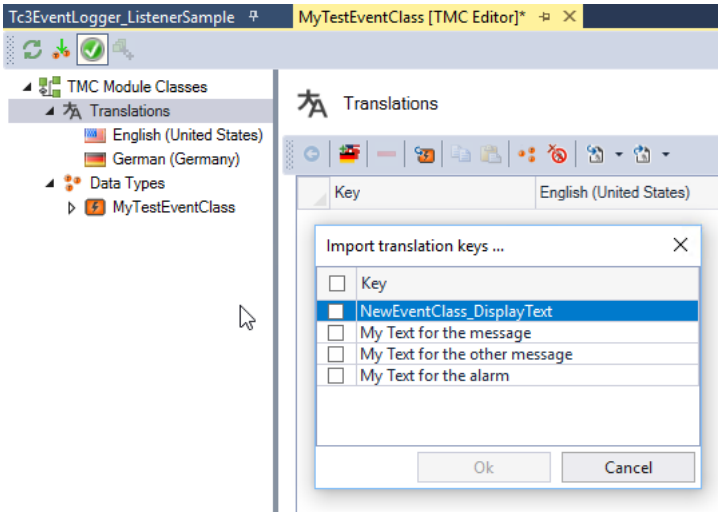
5.4 国际化/翻译

例如，在 HMI 中显示的事件文本（“DisplayText（显示文本）”）可以国际化。

为此在 TMC 编辑器中提供了“Translations（翻译）”部分。本部分描述了一个表，在该表的行中，密钥已分配给不同语言的文本。

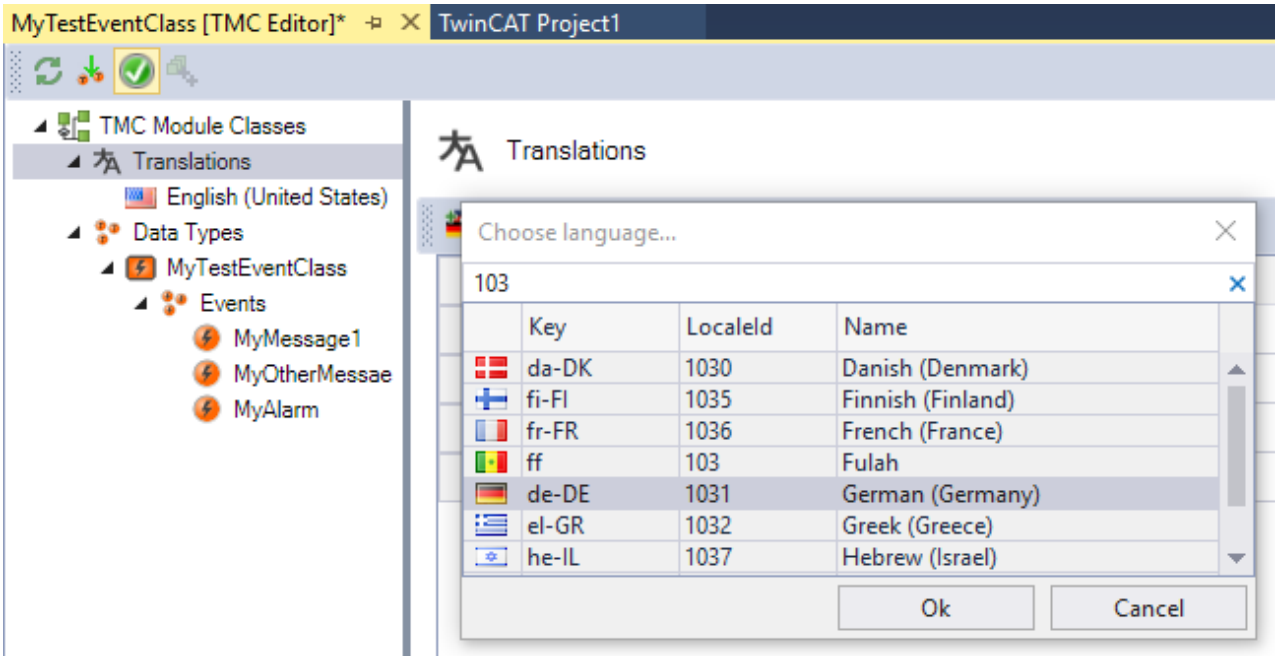
要翻译的密钥

密钥根据事件和事件类的文本（“Display Text（显示文本）”）自动确定。可以单独描述是否要在翻译中采用密钥：



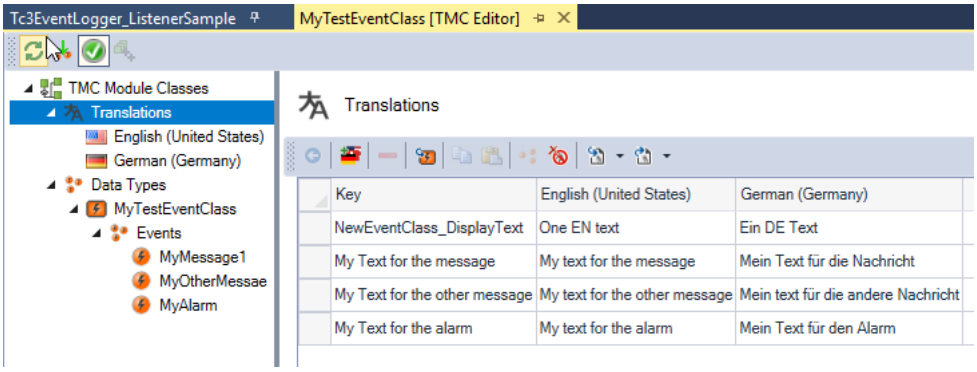
要包含的语言

可以选择和添加语言。如果在运行时需要未存储任何文本的语言，则将使用英语文本。

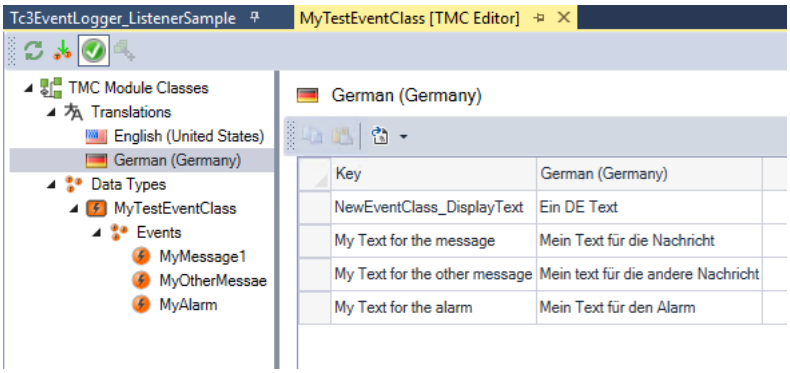


翻译

可以在“Translations（翻译）”下面的表格中放置译文。

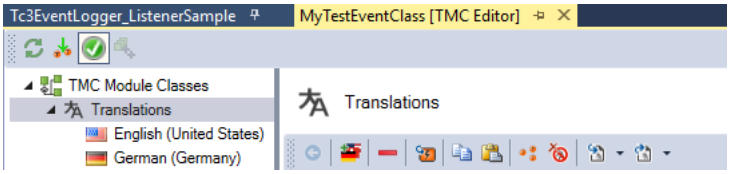


或者，也可以在相应子节点上单独编辑语言。



附加功能

可用于国际化的其他功能：



提供下列功能：

- 添加语言：如上所述添加语言。
- 删除选定的译文：删除密钥及译文。
- 导入翻译密钥：如上所述，导入事件类中使用的密钥。
- 复制：复制译文。
- 粘贴：粘贴译文。
- 显示类型：显示所选密钥的使用情况。
- 删除未使用的密钥：从表中删除任何 EventClass 中未使用的密钥。
- 导入：从 XML 或 CSV 文件导入译文。
- 导出：将译文导出到 XML 或 CSV 文件。

导入和导出功能与 XML/CSV 格式相关，可以通过导出示例对其进行检查。

TwinCAT (XML) 外部的翻译

翻译信息存储在配置文件的专用区域，也可以在 TwinCAT XAE 外部进行编辑。

5.5 目标系统

可以通过注册表项在目标系统上配置 TwinCAT 3 EventLogger。

在 HKEY_LOCAL_MACHINE\SOFTWARE\[WOW6432Node\]Beckhoff\TwinCAT3 下可以使用以下密钥：

Timestamp		
\EventLogger\TimestampSource	DWORD	0 = CurPentiumTime [默认] 2 = CurSystemTime
\EventLogger\TimestampBase	DWORD	0 = SystemTime [默认] 1 = ExternalTimeHard 2 = ExternalTimeMedium 3 = ExternalTimeSoft 对于 1 至 3，请遵守文档“更正时间戳” https://infosys.beckhoff.com/index.php?content=../content/1031/corrected_timestamps/index.html&id=。
消息缓存的最大大小		
\EventLogger\MaxDatabaseSize	DWORD	20 [默认] 单位：MB 一旦达到限制，将丢弃一半的消息。
Windows 错误日志中的消息存储器		
\EventLogger\WindowsEventLog\TypesSupported	DWORD	0 = 无 [默认] | 0x1 = 消息 | 0x2 = 警报 | 0x3 = 消息和警报
\EventLogger\WindowsEventLog\LogLocaleId	DWORD	1033 [默认] 0 = 当前区域设置
\EventLogger\WindowsEventLog\MinLogLevel	DWORD	0 = Verbose [默认] 1 = Info 2 = Warning 3 = Error
记录 TwinCAT 系统错误 **另请见：**https://infosys.beckhoff.com/index.php?content=../content/1031/tceventlogger/html/tceventlogger/setup.htm&id=4066395915595472851		
HKEY_LOCAL_MACHINE\SOFTWARE\[WOW6432Node\]Beckhoff\TwinCAT\System[LogMessageType]	DWORD	3 = 将系统错误接受到 Tc3 EventLogger 中 4 = 将系统错误接受到 Tc3 EventLogger 和 XAE 的错误列表中。

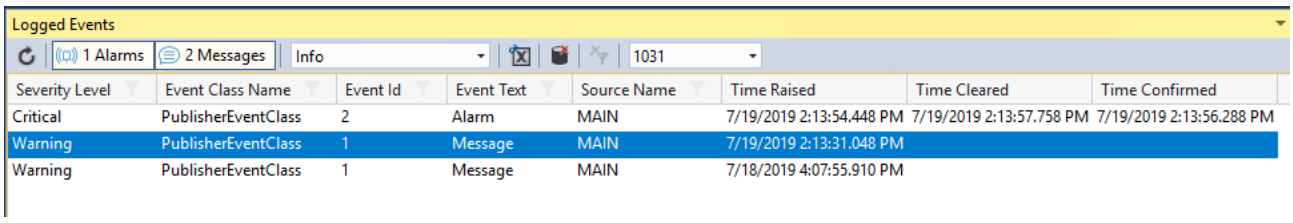
5.6 Engineering

TwinCAT Logged Events (TwinCAT 记录事件) 窗口

兼容性

TwinCAT Logged Events (TwinCAT 记录事件) 窗口通过 Visual Studio 2013 提供。

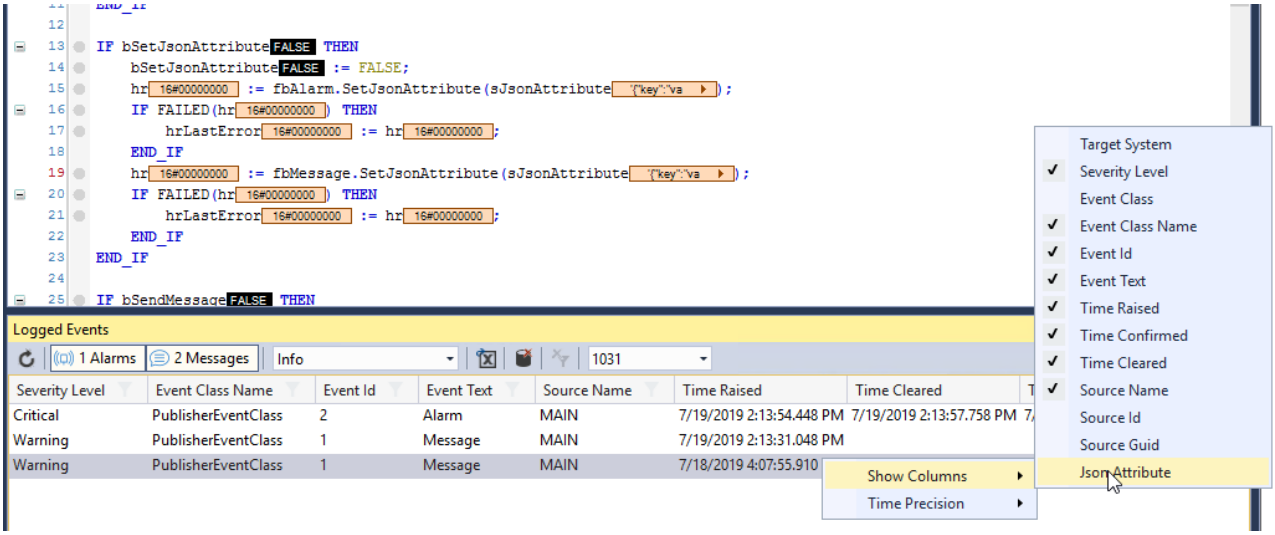
目标系统事件可以从上述缓存数据库中加载，并通过 **Logged Window (记录窗口)** 显示。在 TwinCAT 3 Engineering (XAE) 中，通过 **View (视图) > Other Windows (其他窗口) > TwinCAT Logged Events (TwinCAT 记录事件)** 打开该窗口。



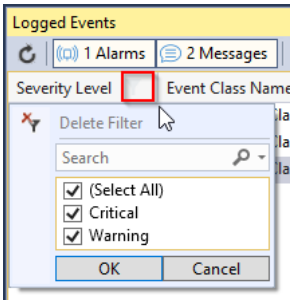
窗口中的工具栏提供以下功能：

	从选定的目标系统加载事件。
	您可以分别通过 **Alarms（警报）** 或 **Messages（消息）** 按钮配置是否显示警报或消息。
	可以通过下拉菜单选择应从哪个严重级别开始显示事件。
	提供 CVS 格式的数据导出，导出当前显示的信息。
	删除（进行提示后）目标系统上的缓存数据库。
	可以通过下拉菜单选择或输入语言。

可以使用上下文菜单中的命令来配置窗口中的列和时间分辨率：

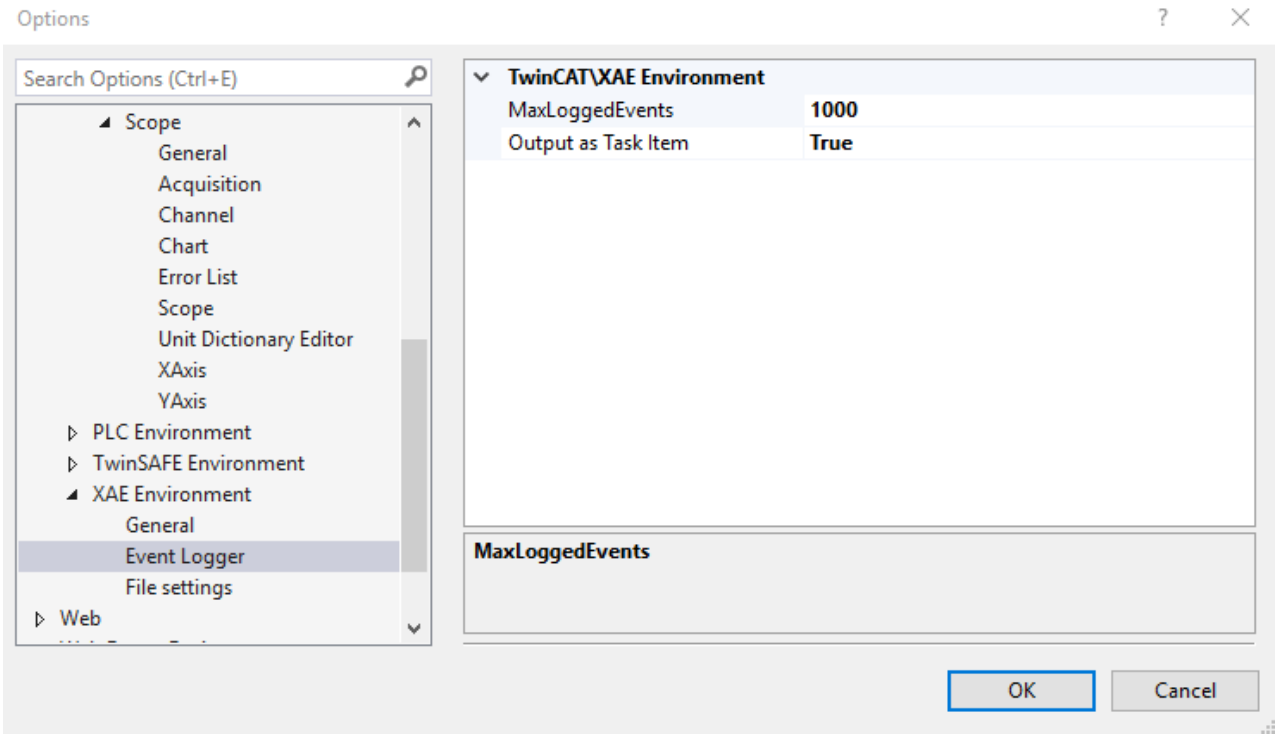


可以通过过滤器功能选择条目：



TwinCAT 选项

TwinCAT 选项（**Tools （工具）> Options（选项）**）中的 TwinCAT Engineering 设置为 TwinCAT 3 EventLogger 提供基本设置。



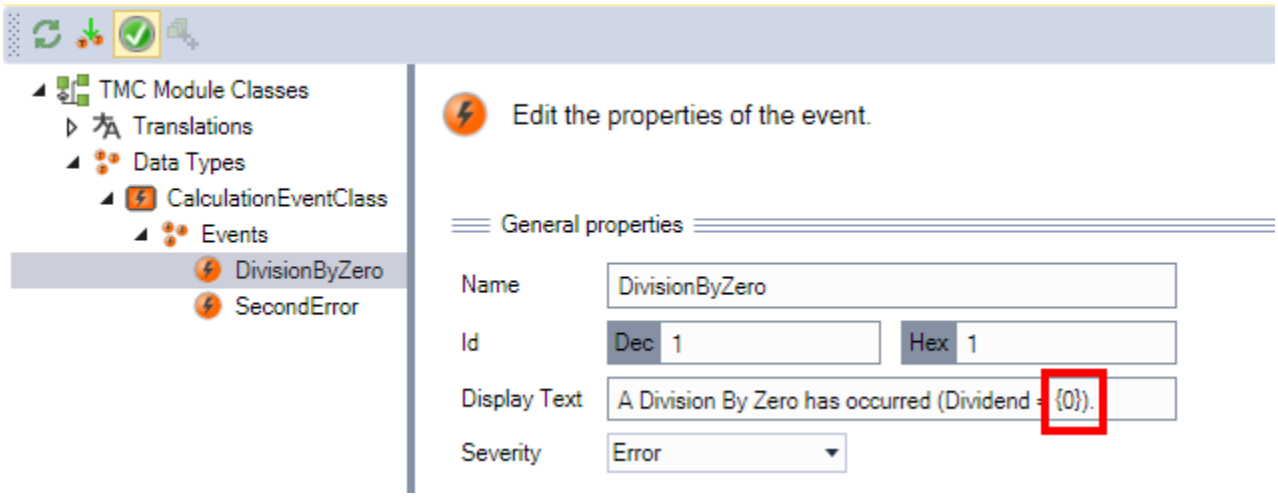
MaxLoggedEvents	**TwinCAT Logged Events (TwinCAT 记录事件)** 窗口中显示的最大消息数。
输出为任务项	显示 **Error List (错误列表)** 窗口中的事件。 该窗口中的显示很有意义，因为无需加载，而是同步进行。但是，该显示并不是针对大量消息而设计的，因此，这意味着必要时还可以使用该选项停用该显示。

5.7 参数

事件文本可以通过使用“参数”进行编程来实现个性化。

为此，在 TMC 编辑器中描述事件时，可使用带有注释 {n} 的标记来表示，其中 n 是一个从 0 开始递增的数字。

例如，在 TMC 编辑器中，此类显示文本用于事件：



然后可以在源代码中按以下方式进行使用：

PLC

可以在 PLC 中按以下方式处理参数：

```
IF NOT fbResult.EqualsToEventEntryEx(stOther:=TC_EVENTS.CalculationEventClass.DivisionByZero) THEN
    hr := fbResult.CreateEx(TC_EVENTS.CalculationEventClass.DivisionByZero, 0 (*fbSource*) );
END_IF

fbResult.ipArguments.Clear().AddLReal(fDividend); //set Argument
```

因此，必须在 Create()/CreateEx() 之后但 Send() 之前定义参数。

可以串联形式指定几个参数。

```
fbResult.ipArguments.Clear().AddLReal(fDividend).AddString(sMyString);
```

在这种情况下，设置 x 取代 {0}，并设置 sMyString 取代 {1}。

C++

可以在 C++ 中按以下方式处理参数：

```
TcArgs tcArgs(m_spMessage);
tcArgs->Clear();
tcArgs.AddArgument(m_dividend);
```

因此，必须在 CreateMessage()/CreateAlarm() 之后但 Send() 之前定义参数。

为此，TcEventLoggerTemplate.h 必须包含在 <ProjectName>Interfaces.h 中。

```
#include"TcRouterInterfaces.h"
#include"TcEventLoggerTemplates.h"
///<AutoGeneratedContent id="Interfaces">
///</AutoGeneratedContent>
```

输出

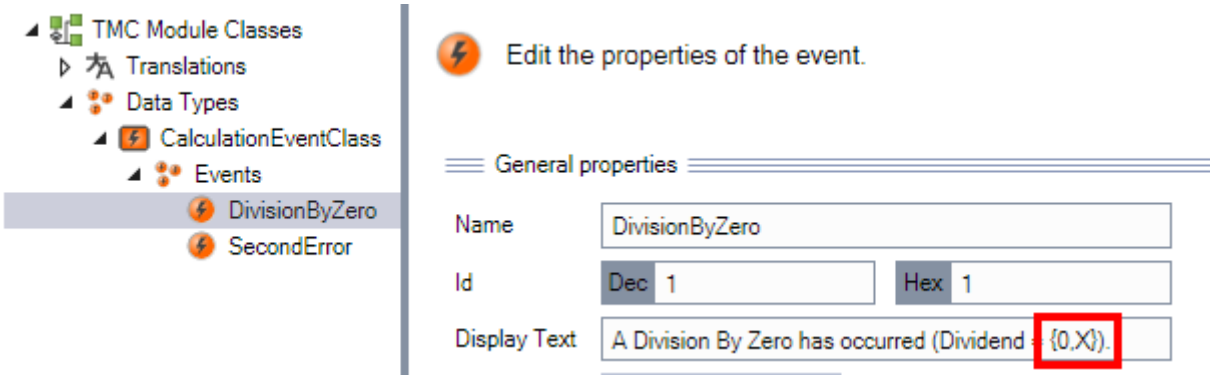
相应的输出为：

Severity Level	EventClassName	EventId	Text
Error	CalculationEventClass	1	A Division By Zero has occurred (Dividend = 42).

该注释也可以用作翻译中的文本。

格式化

参数的输出也可以格式化。为此，在 TMC 编辑器中相应地使用语法 {n,<Format>}。



提供下列格式：

类型	格式	描述
数值	d / D	十进制表达式
	e / E	指数表达式
	x / X	十六进制表达式
	f / F	定点

例如，请注意 REAL 不能表示为“d”或“x”等。

5.8 处理源

同一事件可以在程序的不同点发生。事件源在程序中由“源信息”描述，并在传输时发送。

SourceInfo 由三部分组成（请参见事件 [▶ 10]）。

当使用两种编程语言创建事件时，也会指定来源。

PLC

为此，可使用 PLC 中的 FB_TcSourceInfo。

```
VAR
    fbResult : FB_TcMessage;
    fbSource : FB_TcSourceInfo; // optional
```

在调用 Create()/CreateEx() 之前，相应地对其进行参数化：

```
//adapt source name if required (default is ads symbol name)
fbSource.Clear();
fbSource.sName := 'Math Calculation';
fbSource.nId := 12;

IF NOT fbResult.EqualsToEventEntryEx(stOther:=TC_EVENTS.CalculationEventClass.DivisionByZero) THEN
    hr := fbResult.CreateEx(TC_EVENTS.CalculationEventClass.DivisionByZero, fbSource ); //This uses dyn ressources and shouldn't be called cyclically.
```

或者，也可以在调用 Create()/CreateEx() 时将零分配给相应的参数，以使用 PLC 的内部标准源信息。如果随后**未**指定任何明确的 SourceInfo，则输出符号路径，其中事件被实例化为 SourceName，而 PLC 实例的对象 ID 作为 SourceID。

Logged Events

1 Alarms | 3 Messages | Info | 1031

Severity Level	EventClassName	EventId	Text	SourceName	SourceId
Error	CalculationEventClass	1	A Division By Zero has occurred (Dividend = 42)	MAIN.fbMath	0x08502000
Error	CalculationEventClass	1	A Division By Zero has occurred	MAIN.fbMath	0x08502000

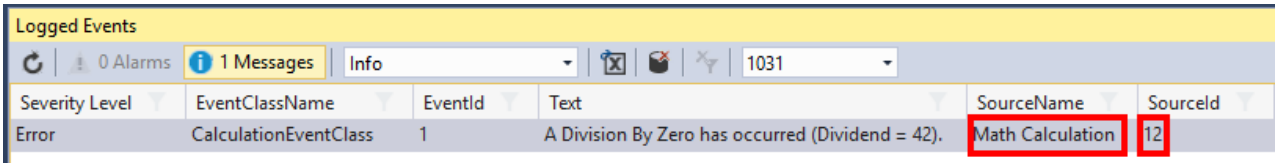
C++

TcSourceInfo 用于 C++ 中，例如，可以使用 CreateMessage()/CreateAlarm() 通过以下方式传输：

```
m_spEventLogger->CreateMessage(TcEvents::MyCppEventClass::MyInit.uuidEventClass,
    TcEvents::MyCppEventClass::MyInit.nEventId,
    TcEvents::MyCppEventClass::MyInit.eSeverity,
    &TcSourceInfo("Math Calculation"),
    &m_spMessageInit);
```

输出

此 SourceInfo 可以相应地显示在 LoggedEvents 窗口中：



5.9 JSON 属性

如技术简介 [▶ 10]章节的介绍部分所述，可传输附加 JSON 属性及消息。

JsonXml 库（PLC library Tc3 JsonXml）可用于在创建和接收时生成 JSON。

PLC

可在 Send() 之前但 Create()/CreateEx() 之后指定 JSON 属性。

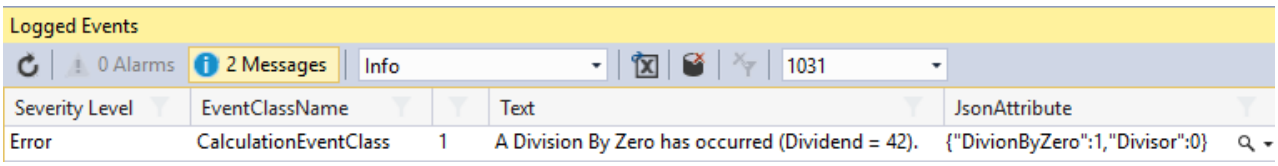
```
//fbSource.Clear();
//fbSource.sName := 'Math Calculation';
IF NOT fbResult.EqualsToEventEntryEx(stOther:=TC_EVENTS.CalculationEventClass.DivisionByZero) THEN
    hr := fbResult.CreateEx(TC_EVENTS.CalculationEventClass.DivisionByZero, 0 ); //This uses dyn ressources and shouldn't be called cyclically.
    IF FAILED(hr) THEN
        hrLastInternalError := hr;
    END_IF

    //set Arguments if required
    //fbResult.ipArguments.Clear();

    fbResult.SetJsonAttribute('{"DivionByZero":1,"Divisor":0}');

    IF fbResult.eSeverity >= eTraceLevel THEN
        hr := fbResult.Send(0);
        IF FAILED(hr) THEN
            hrLastInternalError := hr;
```

C++

可在 Send() 之前但 CreateMessage()/CreateAlarm() 之后指定 JSON 属性。

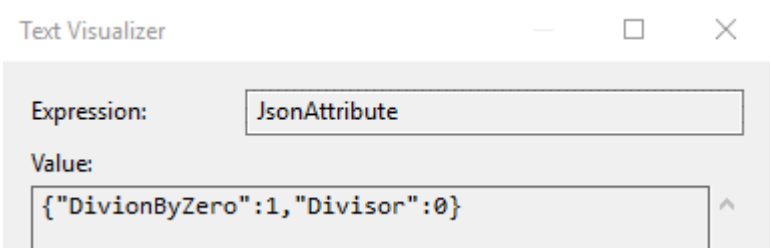
```
237
238         m_spMessage->SetJsonAttribute(m_pjsonAttribute);
239         hr = m_spMessage->Send(0);
240
```

输出



Logged Events（记录事件）窗口为 JSON 属性提供两种可视化方式，可通过信息列内的下拉菜单进行选择，并通过单击放大镜打开：

Text Visualizer



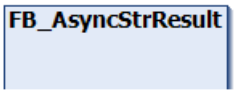
JSON Visualizer

6 PLC API

6.1 函数和函数块

6.1.1 异步文本请求

6.1.1.1 FB_AsyncStrResult



此函数块可实现文本异步请求。

语法

定义：

```
FUNCTION_BLOCK FB_AsyncStrResult
```

方法

名称	描述
GetString [▶ 27]	只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。

属性

名称	类型	访问	描述
bBusy	BOOL	获取	只要处理尚未完成，则为 TRUE。
bError	BOOL	获取	发生错误时为 TRUE。
hrErrorCode	HRESULT	获取	如果 bError 为 TRUE，则输出错误信息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.1.1.1 GetString

只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。

语法

```
METHOD GetString : BOOL
VAR_INPUT
    sResult : REFERENCE TO STRING;
    nResult : UDINT;
END_VAR
```

输入

名称	类型	描述
sResult	引用 STRING	所请求文本的缓冲区变量
nResult	UDINT	缓冲区大小（字节）

返回值

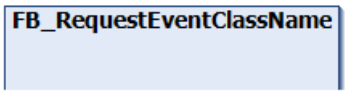
名称	类型	描述
GetString	BOOL	如果可以分配文本，则返回 TRUE。 如果由于指定的缓冲区变量太小而无法完全分配文本，则返回 FALSE。

示例

只有当 bBusy = FALSE 和 bError = FALSE 表示文本可用时，才能调用此方法。

```
IF NOT fb.bBusy AND NOT fb.bError THEN
    bGetStringSuccess := fb.GetString(sText, SIZEOF(sText));
END_IF
```

6.1.1.2 FB_RequestEventClassName



此函数块可实现事件类名称异步请求。

语法

定义：

```
FUNCTION_BLOCK FB_RequestEventClassName
```

方法

名称	描述
GetString [▶ 29]	只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。
请求 [▶ 29]	调用此方法将触发异步文本请求。

属性

名称	类型	访问	描述
bBusy	BOOL	获取	只要处理尚未完成，则为 TRUE。
bError	BOOL	获取	发生错误时为 TRUE。
hrErrorCode	HRESULT	获取	如果 bError 为 TRUE，则输出错误信息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.1.2.1 GetString

只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。

语法

```
METHOD GetString : BOOL
VAR_INPUT
    sResult : REFERENCE TO STRING;
    nResult : UDINT;
END_VAR
```

输入

名称	类型	描述
sResult	引用 STRING	所请求文本的缓冲区变量
nResult	UDINT	缓冲区大小 (字节)

返回值

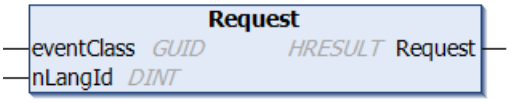
名称	类型	描述
GetString	BOOL	如果可以分配文本，则返回 TRUE。 如果由于指定的缓冲区变量太小而无法完全分配文本，则返回 FALSE。

示例

只有当 bBusy = FALSE 和 bError = FALSE 表示文本可用时，才能调用此方法。

```
IF NOT fb.bBusy AND NOT fb.bError THEN
    bGetStringSuccess := fb.GetString(sText, SIZEOF(sText));
END_IF
```

6.1.1.2.2 请求



调用此方法将触发异步文本请求。

语法

```
METHOD Request : HRESULT
VAR_INPUT
    eventClass : GUID;
    nLangId    : DINT;
END_VAR
```

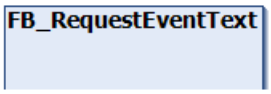
输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031

返回值

名称	类型	描述
请求	HRESULT	返回可能的错误信息。

6.1.1.3 FB_RequestEventText



此函数块可实现所需语言的事件文本异步请求。

语法

定义：

```
FUNCTION_BLOCK FB_RequestEventText
```

方法

名称	描述
GetString [▶ 30]	只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。
请求 [▶ 31]	调用此方法将触发异步文本请求。

属性

名称	类型	访问	描述
bBusy	BOOL	获取	只要处理尚未完成，则为 TRUE。
bError	BOOL	获取	发生错误时为 TRUE。
hrErrorCode	HRESULT	获取	如果 bError 为 TRUE，则输出错误信息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.1.3.1 GetString

只要 bBusy 为 FALSE 并且没有发生错误 (bError = FALSE)，就可以使用此方法来获取请求文本。

语法

```
METHOD GetString : BOOL
VAR_INPUT
    sResult : REFERENCE TO STRING;
    nResult : UDINT;
END_VAR
```

输入

名称	类型	描述
sResult	引用 STRING	所请求文本的缓冲区变量
nResult	UDINT	缓冲区大小（字节）

返回值

名称	类型	描述
GetString	BOOL	如果可以分配文本，则返回 TRUE。 如果由于指定的缓冲区变量太小而无法完全分配文本，则返回 FALSE。

示例

只有当 bBusy = FALSE 和 bError = FALSE 表示文本可用时，才能调用此方法。

```
IF NOT fb.bBusy AND NOT fb.bError THEN
    bGetStringSuccess := fb.GetString(sText, SIZEOF(sText));
END_IF
```

6.1.1.3.2 请求

调用此方法将触发异步文本请求。

语法

```
METHOD Request : BOOL
VAR_INPUT
    eventClass : GUID;
    nEventId   : UDINT;
    nLangId    : DINT;
    ipArgs     : I_TcArguments;
END_VAR
```

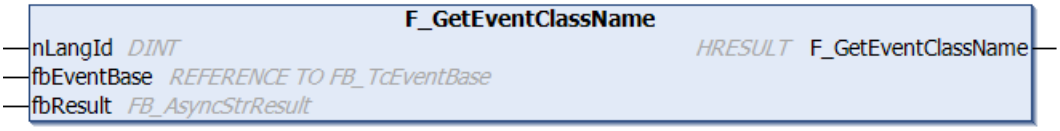
输入

名称	类型	描述
eventClass	GUID	指定事件类。
nEventId	UDINT	事件 ID。
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
ipArgs	I_TcArguments [▶ 64]	可选参数规范。

返回值

名称	类型	描述
请求	HRESULT	返回可能的错误信息。

6.1.1.4 F_GetEventClassName



此函数触发事件类名称异步请求。

语法

定义：

```
FUNCTION F_GetEventClassName : HRESULT
VAR_INPUT
    nLangId     : DINT;
    fbEventBase : REFERENCE TO FB_TcEventBase;
END_VAR
VAR_IN_OUT
    fbResult : FB_AsyncStrResult;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
fbEventBase	引用 FB_TcEventBase [▶ 48]	事件/警报/消息对象规范。

输入/输出

名称	类型	描述
fbResult	FB_AsyncStrResult [▶ 27]	函数块实例规范，以便跟踪异步文本请求。

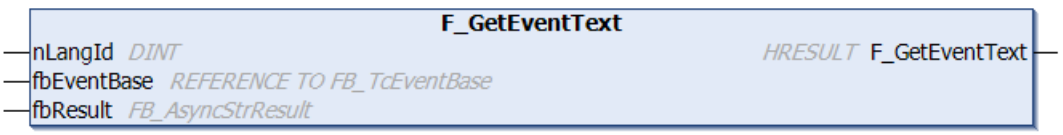
返回值

名称	类型	描述
F_GetEventClassName	HRESULT	返回可能的错误信息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.1.5 F_GetEventText



此函数触发事件文本异步请求。

语法

定义：

```
FUNCTION F_GetEventText : HRESULT
VAR_INPUT
    nLangId     : DINT;
    fbEventBase : REFERENCE TO FB_TcEventBase;
END_VAR
VAR_IN_OUT
    fbResult : FB_AsyncStrResult;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
fbEventBase	引用 FB_TcEventBase [▶ 48]	事件/警报/消息对象规范。

输入/输出

名称	类型	描述
fbResult	FB_AsyncStrResult [▶ 27]	函数块实例规范，以便跟踪异步文本请求。

返回值

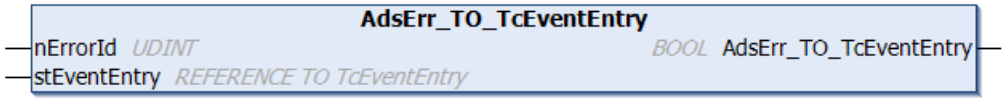
名称	类型	描述
F_GetEventText	HRESULT	返回可能的错误信息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.2 EventEntry 转换

6.1.2.1 AdsErr_TO_TcEventEntry



此函数将标准 ADS 错误转换为 TcEventEntry。

语法

定义：

```
FUNCTION AdsErr_TO_TcEventEntry : BOOL
VAR_INPUT
    nErrorId     : UDINT;
    stEventEntry : REFERENCE TO TcEventEntry;
END_VAR
```

输入

名称	类型	描述
nErrorId	UDINT	要转换的错误代码。
stEventEntry	引用 TcEventEntry [▶ 80]	输出生成的事件定义。

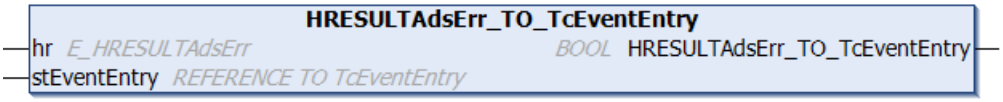
返回值

名称	类型	描述
AdsErr_TO_TcEventEntry	BOOL	如果转换成功，则返回 TRUE。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.2.2 HRESULTAdsErr_TO_TcEventEntry



此函数将标准 ADS 错误 (HRESULT) 转换为 TcEventEntry。

语法

定义：

```
FUNCTION HRESULTAdsErr_TO_TcEventEntry : BOOL
VAR_INPUT
    hr           : E_HRESULTAdsErr;
    stEventEntry : REFERENCE TO TcEventEntry;
END_VAR
```

输入

名称	类型	描述
hr	E_HRESULTAdsErr	要转换的错误代码。
stEventEntry	引用 TcEventEntry [▶ 80]	输出生成的事件定义。

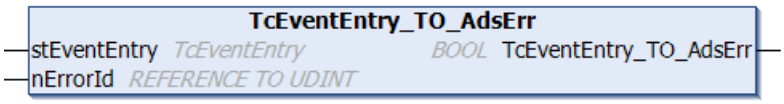
返回值

名称	类型	描述
HRESULTAdsErr_TO_TcEventEntry	BOOL	如果转换成功，则返回 TRUE。 如果指定 HRESULT 中的设备代码未知，则调用失败。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.2.3 TcEventEntry_TO_AdsErr



此函数将 TcEventEntry 转换为标准 ADS 错误。

语法

定义：

```
FUNCTION TcEventEntry_TO_AdsErr : BOOL
VAR_INPUT
    stEventEntry : TcEventEntry;
    nErrorId     : REFERENCE TO UDINT;
END_VAR
```

输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	要转换的事件定义。
nErrorId	引用 UDINT	输出生成的错误代码。

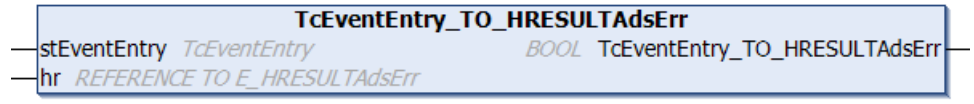
返回值

名称	类型	描述
TcEventEntry_TO_AdsErr	BOOL	如果转换成功，则返回 TRUE；如果事件类未知，则返回 FALSE。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.2.4 TcEventEntry_TO_HRESULTAdsErr



此函数将 TcEventEntry 转换为标准 ADS 错误 (HRESULT)。

语法

定义：

```
FUNCTION TcEventEntry_TO_HRESULTAdsErr : BOOL
VAR_INPUT
    stEventEntry : TcEventEntry;
    hr           : REFERENCE TO E_HRESULTAdsErr;
END_VAR
```

输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	要转换的事件定义。
hr	REFERENCE TO E_HRESULTAdsErr	输出生成的错误代码。

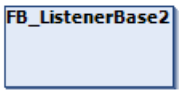
返回值

名称	类型	描述
TcEventEntry_TO_HRESULTAdsErr	BOOL	如果转换成功，则返回 TRUE；如果事件类未知，则返回 FALSE。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.3 FB_ListenerBase2



此函数块用作事件监听器的实现基础。

可以通过覆盖事件驱动方法来标识警报的新消息和状态变化。

语法

定义：

```
FUNCTION_BLOCK FB_ListenerBase2 IMPLEMENTS I_Listener2
```

方法

名称	定义位置	描述
执行 [▶ 37]	本地	必须循环调用，以便可以处理事件队列。
订阅 [▶ 39]	本地	订阅消息。
取消订阅 [▶ 39]	本地	取消订阅消息。

事件驱动方法（回调方法）

名称	定义位置	描述
OnAlarmCleared [▶ 37]	I_Listener2	在警报状态从“Raised (已触发)”更改为“Clear (清除)”时调用。
OnAlarmConfirmed [▶ 37]	I_Listener2	在确认警报时调用。
OnAlarmDisposed [▶ 38]	I_Listener2	再次释放警报实例时调用。
OnAlarmRaised [▶ 38]	I_Listener2	在警报状态从“Clear (清除)”更改为“Raised (已触发)”时调用。
OnMessageSent [▶ 38]	I_Listener2	当发送消息时调用。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4024.0	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger (>= v3.1.21.0)

6.1.3.1 执行

必须循环调用此方法，以便可以处理事件队列。

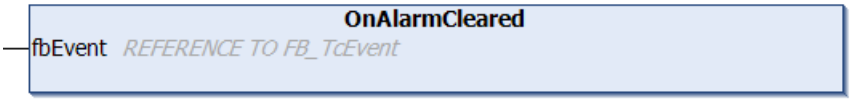
语法

```
METHOD Execute : HRESULT
```

返回值

名称	类型	描述
执行	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.3.2 OnAlarmCleared



如果警报状态从 Raised（已触发）更改为 Clear（清除），则调用此方法。

语法

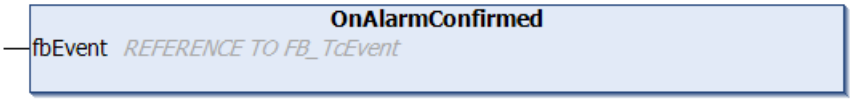
```
METHOD OnAlarmCleared : HRESULT
VAR_INPUT
    fbEvent : REFEENCE TO FB_TcEvent;
END_VAR
```

如果实现回调方法将返回返回代码 <> S_OK，将暂停进一步的回调调用，直到下一次执行。

输入

名称	类型	描述
fbEvent	引用 FB_TcEvent [▶ 47]	引用已发出的警报。

6.1.3.3 OnAlarmConfirmed



在确认警报时调用此方法。

语法

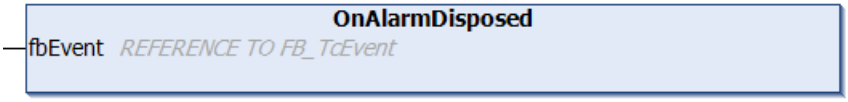
```
METHOD OnAlarmConfirmed : HRESULT
VAR_INPUT
    fbEvent : REFEENCE TO FB_TcEvent;
END_VAR
```

如果实现回调方法将返回返回代码 <> S_OK，将暂停进一步的回调调用，直到下一次执行。

输入

名称	类型	描述
fbEvent	引用 FB_TcEvent [▶ 47]	引用已发出的警报。

6.1.3.4 OnAlarmDisposed



再次释放警报实例时调用此方法。

语法

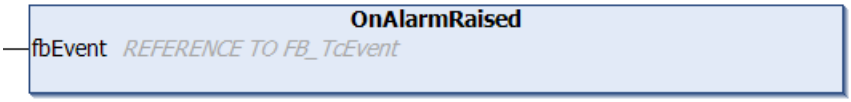
```
METHOD OnAlarmConfirmed : HRESULT
VAR_INPUT
    fbEvent : REFEENCE TO FB_TcEvent;
END_VAR
```

如果实现回调方法将返回返回代码 <> S_OK，将暂停进一步的回调调用，直到下一次执行。

输入

名称	类型	描述
fbEvent	引用 FB_TcEvent [▶ 47]	引用已发出的警报。

6.1.3.5 OnAlarmRaised



如果警报状态从 Clear（清除）更改为 Raised（已触发），则调用此方法。

语法

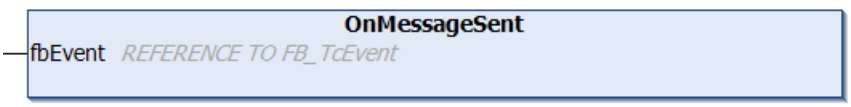
```
METHOD OnAlarmRaised : HRESULT
VAR_INPUT
    fbEvent : REFERENCE TO FB_TcEvent;
END_VAR
```

如果实现回调方法将返回返回代码 <> S_OK，将暂停进一步的回调调用，直到下一次执行。

输入

名称	类型	描述
fbEvent	引用 FB_TcEvent [▶ 47]	引用已发出的警报。

6.1.3.6 OnMessageSent



当发送消息时调用此方法。

语法

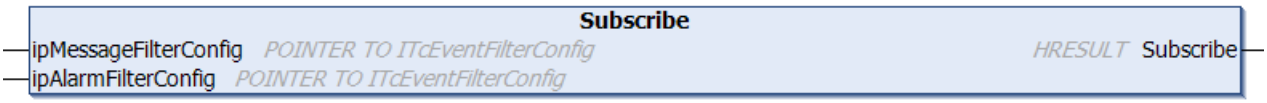
```
METHOD OnMessageSent : HRESULT
VAR_INPUT
    fbEvent : REFEENCE TO FB_TcEvent;
END_VAR
```

如果实现回调方法将返回返回代码 <> S_OK，将暂停进一步的回调调用，直到下一次执行。

输入

名称	类型	描述
fbEvent	引用 FB_TcEvent [▶ 47]	引用已发生的事件。

6.1.3.7 订阅



使用此方法订阅消息监听器。

语法

```
METHOD Subscribe : HRESULT
VAR_INPUT
    ipMessageFilterConfig : POINTER TO ITcEventFilterConfig;
    ipAlarmFilterConfig   : POINTER TO ITcEventFilterConfig;
END_VAR
```

输入

名称	类型	描述
ipMessageFilterConfig	指向 ITcEventFilterConfig	如果要激活过滤器，则指向 ITcEventFilterConfig。
ipAlarmFilterConfig	指向 ITcEventFilterConfig	如果要激活过滤器，则指向 ITcEventFilterConfig。

返回值

名称	类型	描述
订阅	HRESULT	如果方法调用成功，则返回 S_OK。 如果已订阅监听器，则返回 ADS_E_EXISTS。 否则，返回 HRESULT 作为错误代码。

6.1.3.8 取消订阅

使用此方法取消订阅监听器。

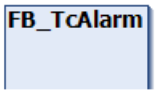
语法

```
METHOD Unsubscribe : HRESULT
```

返回值

名称	类型	描述
取消订阅	HRESULT	如果方法调用成功，则返回 S_OK。 如果未订阅监听器，则返回 ADS_E_NOTFOUND。 否则，返回 HRESULT 作为错误代码。

6.1.4 FB_TcAlarm



函数块表示 TwinCAT 3 EventLogger 的警报。

语法

定义：

```
FUNCTION_BLOCK FB_TcAlarm EXTENDS FB_TcEventBase
```

继承层次结构

FB_TcEventBase [▶ 48]

FB_TcAlarm

方法

名称	定义位置	描述
EqualsTo [▶ 49]	从 FB_TcEventBase [▶ 48] 中继承	将事件与其他实例进行比较。
EqualsToEventClass [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件类与其他事件类进行比较。
EqualsToEventEntry [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件类、事件 ID 和事件严重级别与其他事件进行比较。
EqualsToEventEntryEx [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件定义与其他事件定义进行比较。
GetJsonAttribute [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	返回 Json 属性。
释放 [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	再次释放通过 EventLogger 创建的实例。
RequestEventClassName [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	请求事件类的名称。
RequestEventText [▶ 53]	从 FB_TcEventBase [▶ 48] 中继承	返回事件文本。
清除 [▶ 41]	本地	将警报状态设置为 Not Raised (未触发)。
确认 [▶ 42]	本地	确认警报。
创建 [▶ 43]	本地	在 EventLogger 中创建警报实例。
CreateEx [▶ 43]	本地	根据事件定义在 EventLogger 中创建警报实例。
触发 [▶ 44]	本地	将警报状态设置为 Raised (已触发)。
SetJsonAttribute [▶ 44]	本地	设置 Json 属性。

属性

名称	类型	访问	定义位置	描述
eSeverity	TcEventSeverity [▶ 81]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回严重级别。
EventClass	GUID	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件类的 GUID。
ipArguments [▶ 54]	I_TcArguments [▶ 64]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回参数的接口指针。
ipSourceInfo [▶ 54]	I_TcSourceInfo [▶ 79]	获取	从 FB_TcEventBase [▶ 48] 中继承	SourceInfo 在内部创建为默认行为。然后，它包含函数块的符号名称，将 FB_TcMessage 实例化为 SourceName，并将 PLC 的对象 ID 实例化为 SourceID。 如果使用属性“hide (隐藏)”隐藏了 FB_TcMessage 的实例，则无法在内部为默认行为创建符号名称。
nEventId	nEventId	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件 ID。
stEventEntry	TcEventEntry [▶ 80]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件定义。
bRaised	BOOL	获取	本地	如果警报处于已触发状态，则返回 TRUE。
eConfirmationState	TcEventConfirmationState [▶ 81]	获取	本地	返回确认状态。
nTimeCleared	ULINT	获取	本地	返回清除时间。
nTimeConfirmed	ULINT	获取	本地	返回确认时间。
nTimeRaised	ULINT	获取	本地	返回触发时间。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.4.1 清除

此方法将警报状态 [▶ 11]设置为 Not Raised (未触发)。

语法

```
METHOD Clear : HRESULT
VAR_INPUT
    nTimeStamp         : ULINT;
    bResetConfirmation : BOOL;
END_VAR
```

输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
bResetConfirmation	BOOL	如果为 TRUE 且确认状态为 WaitForConfirmation，则确认状态设置为 Reset（重置）。 否则，确认状态不会更改。

返回值

名称	类型	描述
清除	HRESULT	如果方法调用成功，则返回 S_OK。 如果警报未处于 Raised（已触发）状态，则返回 ADS_E_INVALIDSTATE。 否则，返回 HRESULT 作为错误代码。

6.1.4.2 确认

将 WaitingForConfirmation 的确认状态 [▶ 11]设置为 Confirmed（已确认）。

语法

```
METHOD Confirm : HRESULT
VAR_INPUT
    nTimeStamp: ULINT;
END_VAR
```

输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

名称	类型	描述
确认	HRESULT	如果方法调用成功，则返回 S_OK。 如果确认状态不是 WaitConfirmation，则返回 ADS_E_INVALIDSTATE。 否则，返回 HRESULT 作为错误代码。

6.1.4.3 创建

此方法在 EventLogger 中创建警报实例。

语法

```
METHOD Create : HRESULT
    eventClass         : GUID;
    nEventId           : UDINT;
    eSeverity          : TcEventSeverity;
    bWithConfirmation  : BOOL;
    ipSourceInfo       : I_TcSourceInfo;
END_VAR
```

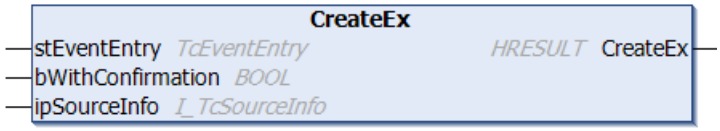
输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
eSeverity	TcEventSeverity [▶ 81]	事件的严重级别。
bWithConfirmation	BOOL	定义警报是否需要强制确认。
ipSourceInfo	I_TcSourceInfo [▶ 79]	接口指向源信息。 如果未传输任何接口指针，则创建默认源信息。

返回值

名称	类型	描述
创建	HRESULT	如果成功创建了新警报，则返回 S_OK。 如果警报已存在，则返回 ERROR_ALREADY_EXISTS。 否则，返回 HRESULT 作为错误代码

6.1.4.4 CreateEx



此方法在 EventLogger 中创建警报实例。

语法

```
METHOD CreateEx : HRESULT
VAR_INPUT
    stEventEntry       : TcEventEntry;
    bWithConfirmation  : BOOL;
    ipSourceInfo       : I_TcSourceInfo;
END_VAR
```

输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	事件定义。
bWithConfirmation	BOOL	定义警报是否需要强制确认。
ipSourceInfo	I_TcSourceInfo [▶ 79]	接口指向源信息。 如果未传输任何接口指针，则创建默认源信息。

返回值

名称	类型	描述
CreateEx	HRESULT	如果成功创建了新警报，则返回 S_OK。 如果警报已存在，则返回 ERROR_ALREADY_EXISTS。 否则，返回 HRESULT 作为错误代码。

6.1.4.5 触发

将警报状态 [▶ 11]设置为 Raised（已触发）。

如果警报需要强制确认，则确认状态会另外设置为 WaitForConfirmation。

语法

```
METHOD Raise : HRESULT
VAR_INPUT
    nTimeStamp : ULINT;
END_VAR
```

输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

名称	类型	描述
触发	HRESULT	如果方法调用成功，则返回 S_OK。 如果警报已处于 Raised（已触发）状态，则返回 ADS_E_INVALIDSTATE。 否则，返回 HRESULT 作为错误代码

6.1.4.6 SetJsonAttribute

此方法设置 JSON 属性。

语法

```
METHOD SetJsonAttribute : HRESULT
VAR_IN_OUT CONSTANT
    sJsonAttribute : STRING;
END_VAR
```

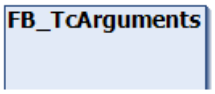
输入

名称	类型	描述
sJsonAttribute	STRING	JSON string

返回值

名称	类型	描述
SetJsonAttribute	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.5 FB_TcArguments



可以使用此函数块定义事件的参数。因此，此函数块可实现 I_TcArguments 接口。

语法

定义：

```
FUNCTION_BLOCK FB_TcArguments IMPLEMENTS I_TcArguments
```

接口

类型	描述
I_TcArguments [▶ 64]	定义参数处理。

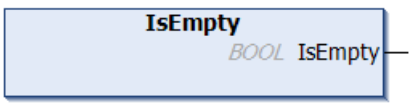
方法

名称	定义位置	描述
AddBlob [▶ 65]	I_TcArguments [▶ 64]	添加二进制数据作为参数。
AddBool [▶ 66]	I_TcArguments [▶ 64]	添加 BOOL 类型的参数。
AddByte [▶ 66]	I_TcArguments [▶ 64]	添加 BYTE 类型的参数。
AddDint [▶ 66]	I_TcArguments [▶ 64]	添加 DINT 类型的参数。
AddDWord [▶ 67]	I_TcArguments [▶ 64]	添加 DWORD 类型的参数。
AddEventReferencId [▶ 67]	I_TcArguments [▶ 64]	将引用添加到另一个事件，作为参数。
AddEventReferenceIdGuid [▶ 68]	I_TcArguments [▶ 64]	将引用添加到另一个事件，作为参数。
AddInt [▶ 68]	I_TcArguments [▶ 64]	添加 INT 类型的参数。
AddLInt [▶ 69]	I_TcArguments [▶ 64]	添加 LINT 类型的参数。
AddLReal [▶ 69]	I_TcArguments [▶ 64]	添加 LREAL 类型的参数。
AddReal [▶ 70]	I_TcArguments [▶ 64]	添加 REAL 类型的参数。
AddSInt [▶ 70]	I_TcArguments [▶ 64]	添加 SINT 类型的参数。
AddString [▶ 71]	I_TcArguments [▶ 64]	添加 STRING 类型的参数。
AddUDint [▶ 71]	I_TcArguments [▶ 64]	添加 UDINT 类型的参数。
AddUInt [▶ 71]	I_TcArguments [▶ 64]	添加 INT 类型的参数。
AddULInt [▶ 72]	I_TcArguments [▶ 64]	添加 ULINT 类型的参数。
AddUSInt [▶ 72]	I_TcArguments [▶ 64]	添加 USINT 类型的参数。
AddWord [▶ 73]	I_TcArguments [▶ 64]	添加 WORD 类型的参数。
AddWString [▶ 73]	I_TcArguments [▶ 64]	添加 WSTRING 类型的参数。
清除 [▶ 74]	I_TcArguments [▶ 64]	删除所有参数。
IsEmpty [▶ 46]	本地	检查是否已添加参数。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.5.1 IsEmpty



此方法检查是否已添加参数。

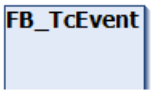
语法

```
METHOD IsEmpty : BOOL
```

返回值

名称	类型	描述
IsEmpty	BOOL	如果未添加任何参数，则返回 TRUE。

6.1.6 FB_TcEvent



此函数块仅提供事件的读取方法和读取属性。

语法

定义：

```
FUNCTION_BLOCK FB_TcEvent EXTENDS FB_TcEventBase IMPLEMENTS I_TcEventBase
```

继承层次结构

FB_TcEventBase [▶ 48]

FB_TcEvent

接口

类型	描述
I_TcEventBase [▶ 74]	定义事件的方法和属性的基本接口。

方法

名称	定义位置	描述
EqualsTo [▶ 49]	从 FB_TcEventBase [▶ 48] 中继承	将事件与其他实例进行比较。
EqualsToEventClass [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件类与其他事件类进行比较。
EqualsToEventEntry [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件定义与其他事件定义进行比较。
EqualsToEventEntryEx [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件定义与其他事件定义进行比较。
GetJsonAttribute [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	返回 Json 属性。
释放 [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	再次释放通过 EventLogger 创建的实例。
RequestEventClassName [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	请求事件类的名称。
RequestEventText [▶ 53]	从 FB_TcEventBase [▶ 48] 中继承	返回事件文本。

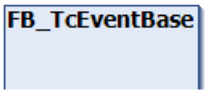
属性

名称	类型	访问	定义位置	描述
eSeverity	TcEventSeverity [▶ 81]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回严重级别。
EventClass	GUID	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件类的 GUID。
ipArguments [▶ 54]	I_TcArguments [▶ 64]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回参数的接口指针。
ipSourceInfo [▶ 54]	I_TcSourceInfo [▶ 79]	获取	从 FB_TcEventBase [▶ 48] 中继承	SourceInfo 在内部创建为默认行为。然后，它包含函数块的符号名称，将 FB_TcMessage 实例化为 SourceName，并将 PLC 的对象 ID 实例化为 SourceID。 如果使用属性“hide (隐藏)”隐藏了 FB_TcMessage 的实例，则无法在内部为默认行为创建符号名称。
nEventId	nEventId	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件 ID。
stEventEntry	TcEventEntry [▶ 80]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件定义。
nTimeStamp	ULINT	获取	本地	返回时间。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.7 FB_TcEventBase



此函数块包含基本实现功能。

语法

定义：

```
FUNCTION_BLOCK FB_TcEventBase
```

方法

名称	定义位置	描述
EqualsTo [▶ 49]	本地	将事件与其他实例进行比较。
EqualsToEventClass [▶ 50]	本地	将事件的事件类与其他事件类进行比较。
EqualsToEventEntry [▶ 50]	本地	将事件的事件定义与其他事件定义进行比较。
EqualsToEventEntryEx [▶ 51]	本地	将事件的事件定义与其他事件定义进行比较。
GetJsonAttribute [▶ 51]	本地	返回 Json 属性。
释放 [▶ 52]	本地	再次释放通过 EventLogger 创建的实例。
RequestEventClassName [▶ 52]	本地	请求事件类的名称。
RequestEventText [▶ 53]	本地	返回事件文本。

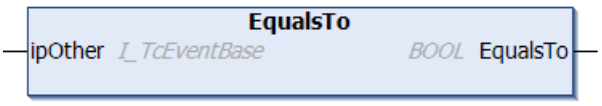
属性

名称	类型	访问	描述
eSeverity	TcEventSeverity [▶ 81]	获取	返回严重级别。
EventClass	GUID	获取	返回事件类的 GUID。
ipArguments [▶ 54]	I_TcArguments [▶ 64]	获取	返回参数的接口指针。
ipSourceInfo [▶ 54]	I_TcSourceInfo [▶ 79]	获取	SourceInfo 在内部创建为默认行为。然后，它包含函数块的符号名称，将 FB_TcMessage 实例化为 SourceName，并将 PLC 的对象 ID 实例化为 SourceID。 如果使用属性“hide (隐藏)”隐藏了 FB_TcMessage 的实例，则无法在内部为默认行为创建符号名称。
nEventId	nEventId	获取	返回事件 ID
stEventEntry	TcEventEntry [▶ 80]	获取	返回事件定义。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.7.1 EqualsTo



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsTo : BOOL
VAR_INPUT
    ipOther : I_TcEventBase;
END_VAR
```

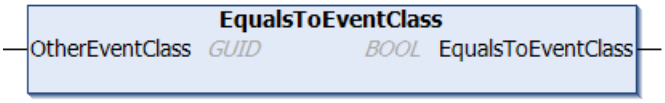
输入

名称	类型	描述
ipOther	I_TcEventBase [▶ 74]	要比较的事件

返回值

名称	类型	描述
EqualsTo	BOOL	如果事件匹配，则返回 TRUE。

6.1.7.2 EqualsToEventClass



此方法执行与输入中指定的另一个事件类的比较。

语法

```
METHOD EqualsToEventClass : BOOL
VAR_INPUT
    OtherEventClass : GUID
END_VAR
```

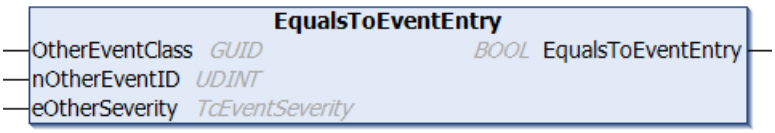
输入

名称	类型	描述
OtherEventClass	GUID	要比较的事件类。

返回值

名称	类型	描述
EqualsToEventClass	BOOL	如果事件类匹配，则返回 TRUE。

6.1.7.3 EqualsToEventEntry



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsToEventEntry : BOOL
VAR_INPUT
    OtherEventClass : GUID;
    nOtherEventID   : UDINT;
    eOtherSeverity  : TcEventSeverity;
END_VAR
```

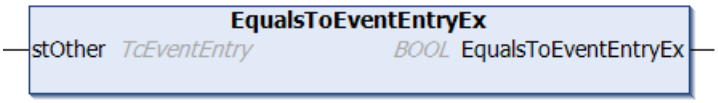
输入

名称	类型	描述
OtherEventClass	GUID	要比较的事件的事件类。
nOtherEventID	UDINT	要比较的事件的事件 ID。
eOtherSeverity	TcEventSeverity [▶ 81]	要比较的事件的事件严重级别。

返回值

名称	类型	描述
EqualsToEventEntry	BOOL	如果事件匹配，则返回 TRUE。

6.1.7.4 EqualsToEventEntryEx



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsToEventEntryEx : BOOL
VAR_INPUT
    stOther : TcEventEntry;
END_VAR
```

输入

名称	类型	描述
stOther	TcEventEntry [▶ 80]	要比较的事件。

返回值

名称	类型	描述
EqualsToEventEntryEx	BOOL	如果事件匹配，则返回 TRUE。

6.1.7.5 GetJsonAttribute

此方法返回 JSON 属性。

语法

```
METHOD GetJsonAttribute : HRESULT
VAR_INPUT
    sJsonAttribute : REFERENCE TO STRING;
    nJsonAttribute : UDINT;
END_VAR
```

输入

名称	类型	描述
sJsonAttribute	引用 STRING	引用 String 类型的变量
nJsonAttribute	UDINT	String 变量的长度

返回值

名称	类型	描述
GetJsonAttribute	HRESULT	如果方法调用成功，则返回 S_OK。 如果变量长度过小，则返回 ERROR_BAD_LENGTH。 否则，返回 HRESULT 作为错误代码。

6.1.7.6 释放

此方法再次释放通过 EventLogger 创建的实例。

语法

```
METHOD Release : HRESULT
```

返回值

名称	类型	描述
释放	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.7.7 RequestEventClassName

此方法返回事件类的名称。

语法

```
METHOD RequestEventClassName : BOOL
VAR_INPUT
    nLangId      : DINT;
    sResult      : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError       : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
sResult	引用 STRING	引用 String 类型的变量
nResultSize	UDINT	String 变量的大小 (字节)

返回值

名称	类型	描述
RequestEventClassName	BOOL	请求终止后立即返回 TRUE。如果异步请求仍处于激活状态，则返回 FALSE。必须调用此方法，直到返回值为 TRUE。

输出

名称	类型	描述
bError	BOOL	如果方法调用成功，则返回 FALSE。如果发生错误，则返回 TRUE。
hrErrorCode	HRESULT	如果方法调用成功，则返回 S_OK。发生错误时输出错误代码。

6.1.7.8 RequestEventText

此方法返回事件文本。

语法

```
METHOD RequestEventText : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
sResult	引用 STRING	引用 String 类型的变量
nResultSize	UDINT	String 变量的大小 (字节)

返回值

名称	类型	描述
RequestEventText	BOOL	请求终止后立即返回 TRUE。如果异步请求仍处于激活状态，则返回 FALSE。必须调用此方法，直到返回值为 TRUE。

输出

名称	类型	描述
bError	BOOL	如果方法调用成功，则返回 FALSE。如果发生错误，则返回 TRUE。
hrErrorCode	HRESULT	如果方法调用成功，则返回 S_OK。发生错误时输出错误代码。

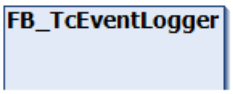
6.1.7.9 ipArguments

```
PROPERTY PUBLIC ipArguments : I_TcArguments
```

6.1.7.10 ipSourceInfo

```
PROPERTY ipSourceInfo : I_TcSourceInfo
```

6.1.8 FB_TcEventLogger



此函数块表示 TwinCAT 3 EventLogger 本身。

语法

定义：

```
FUNCTION_BLOCK FB_TcEventLogger
```

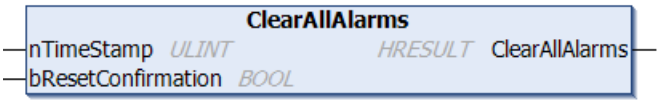
方法

名称	描述
ClearAllAlarms [▶ 55]	为处于 Raised（已触发）状态的所有警报调用 Clear()。
ConfirmAllAlarms [▶ 55]	为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。
GetAlarm [▶ 56]	返回指向现有警报的指针。
GetAlarmEx [▶ 56]	返回指向现有警报的指针。
IsAlarmRaised [▶ 57]	查询警报是否处于 Raised（已触发）状态。
IsAlarmRaisedEx [▶ 57]	查询警报是否处于 Raised（已触发）状态。
SendMessage [▶ 58]	发送消息。
SendMessageEx [▶ 59]	发送消息。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX（x64、x86、ARM）	Tc3_EventLogger

6.1.8.1 ClearAllAlarms



此方法为处于 Raised（已触发）警报状态的所有警报调用 Clear() 方法。

语法

```
METHOD ClearAllAlarms : HRESULT
VAR_INPUT
    nTimeStamp         : ULINT := 0;
    bResetConfirmation : BOOL := FALSE;
END_VAR
```

输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
bResetConfirmation	BOOL	如果为 TRUE 且确认状态为 WaitForConfirmation，则确认状态设置为 Reset（重置）。 否则，确认状态不会更改。

返回值

名称	类型	描述
ClearAllAlarms	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码

6.1.8.2 ConfirmAllAlarms

此方法为确认状态为 WaitForConfirmation 的所有警报调用 Confirm() 方法。

语法

```
METHOD ConfirmAllAlarms : HRESULT
VAR_INPUT
    nTimeStamp : ULINT := 0;
END_VAR
```

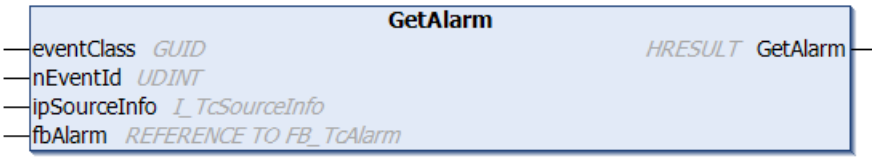
输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

名称	类型	描述
ConfirmAllAlarms	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.8.3 GetAlarm



返回指向现有实例的接口指针。

语法

```
METHOD GetAlarm : HRESULT
VAR_INPUT
    eventClass   : GUID;
    nEventId     : UDINT;
    ipSourceInfo : I_TcSourceInfo := 0;
    fbAlarm      : REFERENCE TO FB_TcAlarm;
END_VAR
```

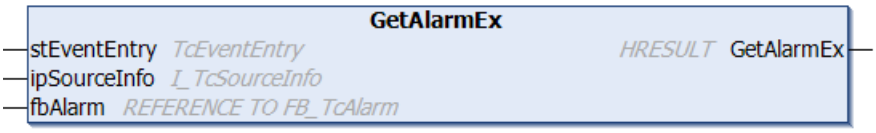
输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。
fbAlarm	引用 FB_TcAlarm [▶ 40]	指向警报。

返回值

名称	类型	描述
GetAlarm	HRESULT	如果未找到实例，则返回 ADS_E_NOTFOUND。 如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.8.4 GetAlarmEx



返回指向现有实例的接口指针。

语法

```
METHOD GetAlarmEx : HRESULT
VAR_INPUT
    stEventEntry : TcEventEntry;
    ipSourceInfo : I_TcSourceInfo := 0; // optional
    fbAlarm      : REFERENCE TO FB_TcAlarm;
END_VAR
```

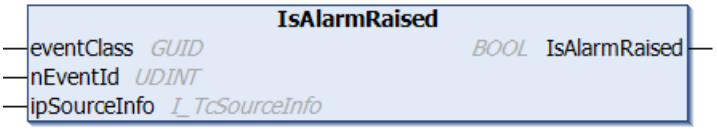
输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	事件定义。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。
fbAlarm	引用 FB_TcAlarm [▶ 40]	指向警报。

返回值

名称	类型	描述
GetAlarmEx	HRESULT	如果未找到实例，则返回 ADS_E_NOTFOUND。 如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.8.5 IsAlarmRaised



此方法查询警报是否处于 Raised（已触发）状态。

语法

```
METHOD IsAlarmRaised : BOOL
VAR_INPUT
    eventClass   : GUID;
    nEventId     : UDINT;
    ipSourceInfo : I_TcSourceInfo := 0;
END_VAR
```

输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。

返回值

名称	类型	描述
IsAlarmRaised	BOOL	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.8.6 IsAlarmRaisedEx

此方法查询警报是否处于 Raised（已触发）状态。

语法

```
METHOD IsAlarmRaisedEx : BOOL
VAR_INPUT
    stEventEntry : TcEventEntry;
    ipSourceInfo : I_TcSourceInfo := 0;
END_VAR
```

输入

名称	类型	描述
stEventEntry	UDINT	事件定义。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。

返回值

名称	类型	描述
IsAlarmRaisedEx	BOOL	如果警报处于已触发状态，则返回 TRUE。

6.1.8.7 SendMessage

此方法发送消息。

语法

```
METHOD SendMessage : HRESULT
VAR_INPUT
    eventClass    : GUID;
    nEventId      : UDINT;
    eSeverity     : TcEventSeverity;
    ipSourceInfo  : I_TcSourceInfo := 0;
    nTimeStamp    : ULINT := 0;
    ipArguments   : I_TcArguments := 0;
END_VAR
```

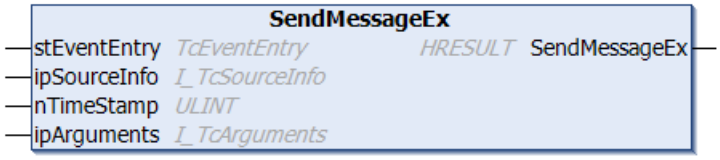
输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
eSeverity	TcEventSeverity [▶ 81]	事件的严重级别。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。
nTimeStamp	ULINT	0：已使用当前时间戳。 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
ipArguments	I_TcArguments [▶ 64]	指向 TcArguments 接口。

返回值

名称	类型	描述
SendMessage	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.8.8 SendMessageEx



此方法发送消息。

语法

```
METHOD SendMessageEx : HRESULT
VAR_INPUT
    stEventEntry : TcEventEntry;
    ipSourceInfo : I_TcSourceInfo := 0;
    nTimeStamp   : ULINT := 0;
    ipArguments  : I_TcArguments := 0;
END_VAR
```

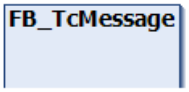
输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	事件定义。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
ipArguments	I_TcArguments [▶ 64]	指向 TcArguments 接口。

返回值

名称	类型	描述
SendMessageEx	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.9 FB_TcMessage



此函数块表示来自 TwinCAT 3 EventLogger 的消息。

语法

定义：

```
FUNCTION_BLOCK FB_TcMessage EXTENDS FB_TcEventBase IMPLEMENTS I_TcMessage
```

继承层次结构

FB_TcEventBase [▶ 48]

FB_TcMessage

接口

类型	描述
I_TcMessage [▶ 79]	提供消息处理的方法和属性。

方法

名称	定义位置	描述
EqualsTo [▶ 49]	从 FB_TcEventBase [▶ 48] 中继承	将事件与其他实例进行比较。
EqualsToEventClass [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件类与其他事件类进行比较。
EqualsToEventEntry [▶ 50]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件定义与其他事件定义进行比较。
EqualsToEventEntryEx [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	将事件的事件定义与其他事件定义进行比较。
GetJsonAttribute [▶ 51]	从 FB_TcEventBase [▶ 48] 中继承	返回 Json 属性。
释放 [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	再次释放通过 EventLogger 创建的实例。
RequestEventClassName [▶ 52]	从 FB_TcEventBase [▶ 48] 中继承	请求事件类的名称。
RequestEventText [▶ 53]	从 FB_TcEventBase [▶ 48] 中继承	返回事件文本。
创建 [▶ 61]	本地	在 EventLogger 中创建消息实例。
CreateEx [▶ 61]	本地	根据事件定义在 EventLogger 中创建消息实例。
SetJsonAttribute [▶ 62]	本地	设置 Json 属性。
发送 [▶ 79]	I_TcMessage [▶ 79]	发送消息。

属性

名称	类型	访问	定义位置	描述
eSeverity	TcEventSeverity [▶ 81]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回严重级别。
EventClass	GUID	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件类的 GUID。
ipArguments [▶ 54]	I_TcArguments [▶ 64]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回参数的接口指针。
ipSourceInfo [▶ 54]	I_TcSourceInfo [▶ 79]	获取	从 FB_TcEventBase [▶ 48] 中继承	SourceInfo 在内部创建为默认行为。然后，它包含函数块的符号名称，将 FB_TcMessage 实例化为 SourceName，并将 PLC 的对象 ID 实例化为 SourceID。 如果使用属性“hide (隐藏)”隐藏了 FB_TcMessage 的实例，则无法在内部为默认行为创建符号名称。
nEventId	nEventId	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件 ID。
stEventEntry	TcEventEntry [▶ 80]	获取	从 FB_TcEventBase [▶ 48] 中继承	返回事件定义。
nTimeSent	ULINT	获取	本地	返回发送时间。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.1.9.1 创建

此方法在 EventLogger 中创建消息实例。

语法

```
METHOD Create : HRESULT
VAR_INPUT
    eventClass    : GUID;
    nEventId      : UDINT;
    eSeverity     : TcEventSeverity;
    ipSourceInfo : I_TcSourceInfo := 0;
END_VAR
```

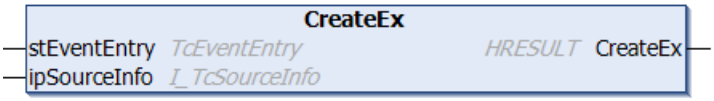
输入

名称	类型	描述
eventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
eSeverity	TcEventSeverity [▶ 81]	定义严重级别。
ipSourceInfo	I_TcSourceInfo [▶ 79]	指向 ITcSourceInfo 接口。

返回值

名称	类型	描述
创建	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.9.2 CreateEx



此方法根据事件定义在 EventLogger 中创建消息实例。

语法

```
METHOD PUBLIC CreateEx : HRESULT
VAR_INPUT
    stEventEntry : TcEventEntry;
    ipSourceInfo : I_TcSourceInfo := 0;
END_VAR
```

输入

名称	类型	描述
stEventEntry	TcEventEntry [▶ 80]	事件定义。
ipSourceInfo	I_TcSourceInfo [▶ 79]	接口指向源信息。 如果未传输任何接口指针，则创建默认源信息。

返回值

名称	类型	描述
CreateEx	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.9.3 SetJsonAttribute

此方法设置 JSON 属性。

语法

```
METHOD SetJsonAttribute : HRESULT
VAR_IN_OUT CONSTANT
    sJsonAttribute : STRING;
END_VAR
```

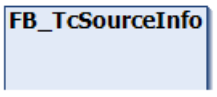
输入

名称	类型	描述
sJsonAttribute	STRING	JSON string

返回值

名称	类型	描述
SetJsonAttribute	HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

6.1.10 FB_TcSourceInfo



可以使用此函数块定义事件的源信息。

语法

定义：

```
FUNCTION_BLOCK FB_TcSourceInfo IMPLEMENTS I_TcSourceInfo
```

接口

类型	描述
I_TcSourceInfo [▶ 79]	提供源信息的读取方法和读取属性。

方法

名称	定义位置	描述
清除 [▶ 63]	本地	重置源信息。
ExtendName [▶ 64]	本地	将传输的字符串附加到名称。
ResetToDefault [▶ 64]	本地	将属性设置为默认值。 sName 以实例化函数块的符号名称初始化。 nId 以 PLC 实例的对象 ID 初始化。 如果使用属性“hide (隐藏)”隐藏了 FB_TcSourceInfo 的实例，则无法在内部为默认行为创建符号名称。
EqualsTo [▶ 80]	I_TcSourceInfo [▶ 79]	将一个实例与另一个实例进行比较。

属性

名称	类型	访问	定义位置	描述
guid	GUID	获取	I_TcSourceInfo [▶ 79]	返回源信息的 GUID。
guid	GUID	SET	本地	将 GUID 设置为源信息。
nId	UDINT	获取	I_TcSourceInfo [▶ 79]	返回源信息 ID。
nId	UDINT	SET	本地	设置源信息 ID。
sName	STRING(ParameterList.cSourceNameSize-1)	获取	I_TcSourceInfo [▶ 79]	返回源信息的名称。
sName	STRING(ParameterList.cSourceNameSize-1)	SET	本地	设置源信息的名称。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

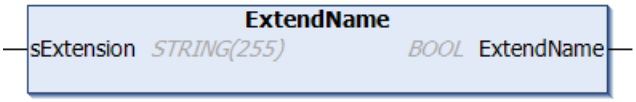
6.1.10.1 清除

此方法重置源信息。

语法

```
METHOD Clear
```

6.1.10.2 ExtendName



此方法扩展名称。

语法

```
METHOD ExtendName : BOOL
VAR_INPUT
    sExtension : STRING(255);
END_VAR
```

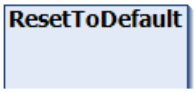
输入

名称	类型	描述
sExtension	STRING(255)	待附加到右侧的文本。

返回值

名称	类型	描述
ExtendName	BOOL	如果串联成功，则返回 TRUE。 如果生成的字符串比输出字符串长且不适合指定的输出缓冲区，则返回 FALSE。此时生成字符串的内存需求将大于输出字符串的内存需求。然后，字符串被截断。

6.1.10.3 ResetToDefault



此方法将源信息设置为默认值。

默认值：

sName 以实例化函数块的符号名称初始化。

nId 以 PLC 实例的对象 ID 初始化。

如果使用属性“hide (隐藏)”隐藏了 FB_TcSourceInfo 的实例，则无法在内部为默认行为创建符号名称。

语法

```
METHOD ResetToDefault
```

6.2 接口

6.2.1 I_TcArguments

此接口定义参数处理方法。

继承层次结构

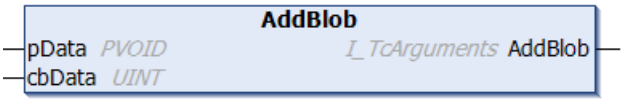
__SYSTEM.IQueryInterface

I_TcArguments

方法

名称	描述
AddBlob [▶ 65]	添加二进制数据作为参数。
AddBool [▶ 66]	添加 BOOL 类型的参数。
AddByte [▶ 66]	添加 BYTE 类型的参数。
AddDint [▶ 66]	添加 DINT 类型的参数。
AddDWord [▶ 67]	添加 DWORD 类型的参数。
AddEventReferencId [▶ 67]	将引用添加到另一个事件，作为参数。
AddEventReferenceIdGuid [▶ 68]	将引用添加到另一个事件，作为参数。
AddInt [▶ 68]	添加 INT 类型的参数。
AddLInt [▶ 69]	添加 LINT 类型的参数。
AddLReal [▶ 69]	添加 LREAL 类型的参数。
AddReal [▶ 70]	添加 REAL 类型的参数。
AddSInt [▶ 70]	添加 SINT 类型的参数。
AddString [▶ 71]	添加 STRING 类型的参数。
AddUDint [▶ 71]	添加 UDINT 类型的参数。
AddUInt [▶ 71]	添加 INT 类型的参数。
AddULInt [▶ 72]	添加 ULINT 类型的参数。
AddUSInt [▶ 72]	添加 USINT 类型的参数。
AddWord [▶ 73]	添加 WORD 类型的参数。
AddWString [▶ 73]	添加 WSTRING 类型的参数。
清除 [▶ 74]	删除所有参数。

6.2.1.1 AddBlob



此方法添加二进制数据作为参数。

语法

```
METHOD AddBlob : I_TcArguments
VAR_INPUT
    pData  : PVOID;
    cbData : UINT;
END_VAR
```

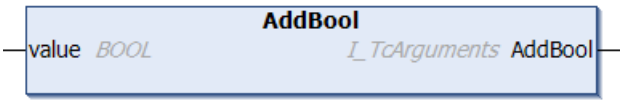
输入

名称	类型	描述
pData	PVOID	指向二进制数据的第一个字节。
cbData	UINT	二进制数据的长度（字节）。

返回值

名称	类型	描述
AddBlob	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.2 AddBool



此方法添加 BOOL 类型的参数。

语法

```
METHOD AddBool : I_TcArguments
VAR_INPUT
    value : BOOL;
END_VAR
```

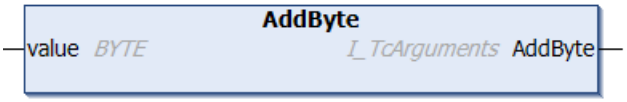
输入

名称	类型	描述
数值	BOOL	要添加的数值。

返回值

名称	类型	描述
AddBool	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.3 AddByte



此方法添加 BYTE 类型的参数。

语法

```
METHOD AddByte : I_TcArguments
VAR_INPUT
    value : BYTE;
END_VAR
```

输入

名称	类型	描述
数值	BYTE	要添加的数值。

返回值

名称	类型	描述
AddByte	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.4 AddDint

此方法添加 DINT 类型的参数。

语法

```
METHOD AddDINT : I_TcArguments
VAR_INPUT
    value : DINT;
END_VAR
```

输入

名称	类型	描述
数值	DINT	要添加的数值。

返回值

名称	类型	描述
AddDINT	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.5 AddDWord

此方法添加 DWORD 类型的参数。

语法

```
METHOD AddDWord : I_TcArguments
VAR_INPUT
    value : DWORD;
END_VAR
```

输入

名称	类型	描述
数值	DWORD	要添加的数值。

返回值

名称	类型	描述
AddDWord	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.6 AddEventReferencId

此方法将引用添加到另一个事件，作为参数。

语法

```
METHOD AddEventReferenceId : I_TcArguments
VAR_INPUT
    nEventId : UDINT;
END_VAR
```

输入

名称	类型	描述
nEventId	UDINT	事件 ID。

返回值

名称	类型	描述
AddEventReferenceId	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.7 AddEventReferenceIdGuid

此方法将引用添加到另一个事件，作为参数。

语法

```
METHOD AddEventReferenceIdGuid : I_TcArguments
VAR_INPUT
    nEventId   : UDINT;
    EventClass : GUID;
END_VAR
```

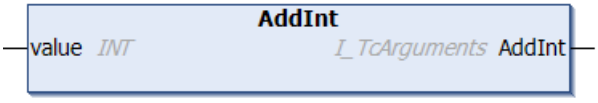
输入

名称	类型	描述
nEventId	UDINT	事件 ID。
EventClass	GUID	事件类的 GUID。

返回值

名称	类型	描述
AddEventReferenceIdGuid	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.8 AddInt



此方法添加 INT 类型的参数。

语法

```
METHOD AddINT : I_TcArguments
VAR_INPUT
    value : INT;
END_VAR
```

输入

名称	类型	描述
数值	INT	要添加的数值。

返回值

名称	类型	描述
AddInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.9 AddLInt

此方法添加 LINT 类型的参数。

语法

```
METHOD AddLInt : I_TcArguments
VAR_INPUT
    value : LINT;
END_VAR
```

输入

名称	类型	描述
数值	LINT	要添加的数值。

返回值

名称	类型	描述
AddLInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.10 AddLReal

此方法添加 LREAL 类型的参数。

语法

```
METHOD AddLReal : I_TcArguments
VAR_INPUT
    value : LREAL;
END_VAR
```

输入

名称	类型	描述
数值	LREAL	要添加的数值。

返回值

名称	类型	描述
AddLReal	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.11 AddReal

此方法添加 REAL 类型的参数。

语法

```
METHOD AddReal : I_TcArguments
VAR_INPUT
    value : REAL;
END_VAR
```

输入

名称	类型	描述
数值	REAL	要添加的数值。

返回值

名称	类型	描述
AddReal	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.12 AddSInt

此方法添加 SINT 类型的参数。

语法

```
METHOD AddSInt : I_TcArguments
VAR_INPUT
    value : SInt;
END_VAR
```

输入

名称	类型	描述
数值	SINT	要添加的数值。

返回值

名称	类型	描述
AddSInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.13 AddString

此方法添加 STRING 类型的参数。

语法

```
METHOD AddString : I_TcArguments
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
```

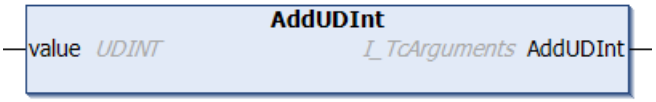
输入

名称	类型	描述
数值	STRING	要添加的数值。

返回值

名称	类型	描述
AddString	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.14 AddUDint



此方法添加 UDINT 类型的参数。

语法

```
METHOD AddUDInt : I_TcArguments
VAR_INPUT
    value : UDINT;
END_VAR
```

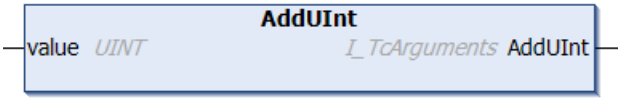
输入

名称	类型	描述
数值	UDINT	要添加的数值。

返回值

名称	类型	描述
AddUDInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.15 AddUInt



此方法添加 INT 类型的参数。

语法

```
METHOD AddUInt : I_TcArguments
VAR_INPUT
    value : UINT;
END_VAR
```

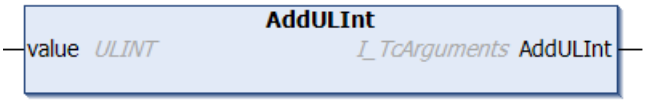
输入

名称	类型	描述
数值	UINT	要添加的数值。

返回值

名称	类型	描述
AddUInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.16 AddULInt



此方法添加 ULINT 类型的参数。

语法

```
METHOD AddULInt : I_TcArguments
VAR_INPUT
    value : ULINT;
END_VAR
```

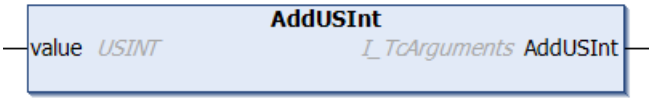
输入

名称	类型	描述
数值	ULINT	要添加的数值。

返回值

名称	类型	描述
AddULInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.17 AddUSInt



此方法添加 USINT 类型的参数。

语法

```
METHOD AddUSInt : I_TcArguments
VAR_INPUT
    value : USINT
END_VAR
```

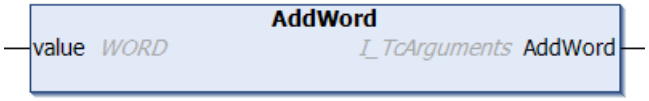
输入

名称	类型	描述
数值	USINT	要添加的数值。

返回值

名称	类型	描述
AddUSInt	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.18 AddWord



此方法添加 WORD 类型的参数。

语法

```
METHOD AddWord : I_TcArguments
VAR_INPUT
    value : WORD;
END_VAR
```

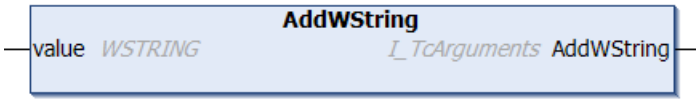
输入

名称	类型	描述
数值	WORD	要添加的数值。

返回值

名称	类型	描述
AddWord	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.19 AddWString



此方法添加 WSTRING 类型的参数。

语法

```
METHOD AddWString : I_TcArguments
VAR_IN_OUT CONSTANT
    value : WSTRING;
END_VAR
```

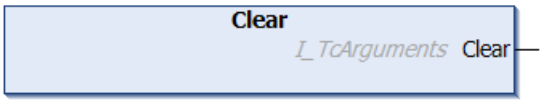
输入

名称	类型	描述
数值	WSTRING	要添加的数值。

返回值

名称	类型	描述
AddWString	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.1.20 清除



此方法删除所有参数。

语法

```
METHOD Clear : I_TcArguments
```

返回值

名称	类型	描述
清除	I_TcArguments [▶ 64]	再次返回 I_TcArgument 指针。

6.2.2 I_TcEventBase

此基本接口定义了事件的方法和属性。

方法

名称	描述
EqualsTo [▶ 75]	将事件与其他实例进行比较。
EqualsToEventClass [▶ 75]	将事件的事件类与其他事件类进行比较。
EqualsToEventEntryEx [▶ 76]	将事件的事件定义与其他事件定义进行比较。
GetJsonAttribute [▶ 77]	返回 Json 属性。
RequestEventClassName [▶ 77]	请求事件类的名称。
RequestEventText [▶ 78]	返回事件文本。

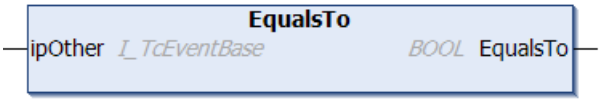
属性

名称	类型	访问	描述
eSeverity	TcEventSeverity [▶ 81]	获取	返回严重级别。
EventClass	GUID	获取	返回事件类的 GUID。
ipSourceInfo	I_TcSourceInfo [▶ 79]	获取	返回指向源定义的指针。
nEventId	UDINT	获取	返回事件 ID。
stEventEntry	TcEventEntry [▶ 80]	获取	返回事件定义。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.2.2.1 EqualsTo



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsTo : BOOL
VAR_INPUT
    ipOther : I_TcEventBase;
END_VAR
```

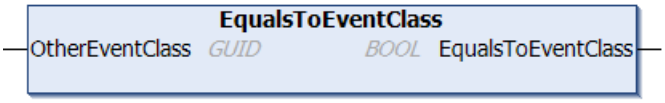
输入

名称	类型	描述
ipOther	I_TcEventBase [▶ 74]	要比较的事件

返回值

名称	类型	描述
EqualsTo	BOOL	如果事件匹配，则返回 TRUE。

6.2.2.2 EqualsToEventClass



此方法执行与输入中指定的另一个事件类的比较。

语法

```
METHOD EqualsToEventClass : BOOL
VAR_INPUT
    OtherEventClass : GUID
END_VAR
```

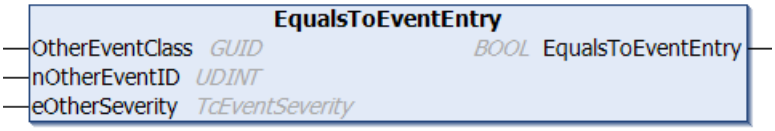
输入

名称	类型	描述
OtherEventClass	GUID	要比较的事件类。

返回值

名称	类型	描述
EqualsToEventClass	BOOL	如果事件类匹配，则返回 TRUE。

6.2.2.3 EqualsToEventEntry



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsToEventEntry : BOOL
VAR_INPUT
    OtherEventClass : GUID;
    nOtherEventID   : UDINT;
    eOtherSeverity  : TcEventSeverity;
END_VAR
```

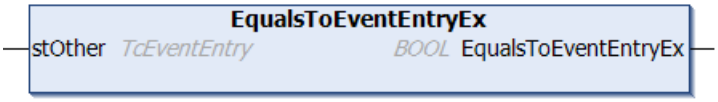
输入

名称	类型	描述
OtherEventClass	GUID	要比较的事件的事件类。
nOtherEventID	UDINT	要比较的事件的事件 ID。
eOtherSeverity	TcEventSeverity [▶ 81]	要比较的事件的事件严重级别。

返回值

名称	类型	描述
EqualsToEventEntry	BOOL	如果事件匹配，则返回 TRUE。

6.2.2.4 EqualsToEventEntryEx



此方法执行与输入中指定的另一个事件的比较。

语法

```
METHOD EqualsToEventEntryEx : BOOL
VAR_INPUT
    stOther : TcEventEntry;
END_VAR
```

输入

名称	类型	描述
stOther	TcEventEntry [▶ 80]	要比较的事件。

返回值

名称	类型	描述
EqualsToEventEntryEx	BOOL	如果事件匹配，则返回 TRUE。

6.2.2.5 GetJsonAttribute

此方法返回 JSON 属性。

语法

```
METHOD GetJsonAttribute : HRESULT
VAR_INPUT
    sJsonAttribute : REFERENCE TO STRING;
    nJsonAttribute : UDINT;
END_VAR
```

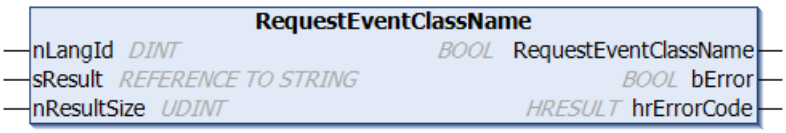
输入

名称	类型	描述
sJsonAttribute	引用 STRING	引用 String 类型的变量
nJsonAttribute	UDINT	String 变量的长度

返回值

名称	类型	描述
GetJsonAttribute	HRESULT	如果方法调用成功，则返回 S_OK。 如果变量长度过小，则返回 ERROR_BAD_LENGTH。 否则，返回 HRESULT 作为错误代码。

6.2.2.6 RequestEventClassName



此方法返回事件类的名称。

语法

```
METHOD RequestEventClassName : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
sResult	引用 STRING	引用 String 类型的变量
nResultSize	UDINT	String 变量的大小 (字节)

返回值

名称	类型	描述
RequestEventClassName	BOOL	请求终止后立即返回 TRUE。如果异步请求仍处于激活状态，则返回 FALSE。必须调用此方法，直到返回值为 TRUE。

输出

名称	类型	描述
bError	BOOL	如果方法调用成功，则返回 FALSE。如果发生错误，则返回 TRUE。
hrErrorCode	HRESULT	如果方法调用成功，则返回 S_OK。发生错误时输出错误代码。

6.2.2.7 RequestEventText

此方法返回事件文本。

语法

```
METHOD RequestEventText : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

输入

名称	类型	描述
nLangId	DINT	指定语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …
sResult	引用 STRING	引用 String 类型的变量
nResultSize	UDINT	String 变量的大小 (字节)

返回值

名称	类型	描述
RequestEventText	BOOL	请求终止后立即返回 TRUE。如果异步请求仍处于激活状态，则返回 FALSE。必须调用此方法，直到返回值为 TRUE。

输出

名称	类型	描述
bError	BOOL	如果方法调用成功，则返回 FALSE。如果发生错误，则返回 TRUE。
hrErrorCode	HRESULT	如果方法调用成功，则返回 S_OK。发生错误时输出错误代码。

6.2.3 I_TcMessage

此接口提供消息处理的方法和属性。

继承层次结构

I_TcEventBase [▶ 74]

I_TcMessage

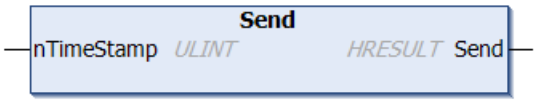
方法

名称	描述
发送 [▶ 79]	发送消息

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.2.3.1 发送



此方法发送消息。

语法

```
METHOD Send : HRESULT
VAR_INPUT
    nTimeStamp: ULINT;
END_VAR
```

输入

名称	类型	描述
nTimeStamp	ULINT	0：已使用当前时间戳 > 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

名称	类型	描述
发送	FB_ HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码

6.2.4 I_TcSourceInfo

此接口定义源信息项的属性。

方法

名称	描述
EqualsTo [▶ 80]	将具有源信息的实例与另一个实例进行比较。

属性

名称	类型	访问	描述
guid	GUID	获取	返回源信息的 GUID。
nId	UDINT	获取	返回源信息 ID。
sName	STRING(ParameterList.cSourceNameSize-1)	获取	返回源信息的名称。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4022.20	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger

6.2.4.1 EqualsTo

此方法将具有源信息的实例与另一个实例进行比较。

语法

```
METHOD EqualsTo : BOOL
VAR_INPUT
    ipOther : I_TcSourceInfo;
END_VAR
```

输入

名称	类型	描述
ipOther	I_TcSourceInfo [▶ 79]	要比较的源信息项

返回值

名称	类型	描述
EqualsTo	BOOL	如果源信息项匹配，则返回 TRUE。

6.3 数据类型

6.3.1 TcEventEntry

通过事件类、事件 ID 和严重级别定义事件。

语法

定义：

```
TYPE TcEventEntry :
STRUCT
    uuidEventClass : GUID;
    nEventId       : UDINT;
    eSeverity      : TcEventSeverity;
END_STRUCT
END_TYPE
```

参数

名称	类型	描述
uuidEventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
eSeverity	TcEventSeverity	事件严重级别定义事件的严重级别，

6.3.2 TcEventSeverity

定义事件的严重级别。

语法

定义：

```
{attribute 'qualified_only'}
TYPE TcEventSeverity : (
    Verbose  := 0,
    Info     := 1,
    Warning  := 2,
    Error    := 3,
    Critical := 4);
END_TYPE
```

参数

	名称	描述
4	Critical	评论
3	Error	错误
2	Warning	警告
1	Info	信息
0	Verbose	扩展输出

6.3.3 TcEventConfirmationState

定义警报的确认状态。

语法

定义：

```
{attribute 'qualified_only'}
TYPE TcEventConfirmationState : (
   NotSupported := 0,
   NotRequired  := 1,
   WaitForConfirmation := 2,
   Confirmed := 3,
   Reset     := 4);
END_TYPE
```

参数

名称	描述
已确认	已确认
NotRequired	在当前状态下无需确认。(警报当前未处于 Raised (已触发) 状态)。
NotSupported (不支持)	未经确认进行了初始化。
Reset (重置)	初始状态
WaitForConfirmation (待确认)	等待确认。

6.4 Global lists (全局列表)

6.4.1 Global_Constants

```
VAR_GLOBAL CONSTANT
    EMPTY_EVENT_CLASS : GUID := (Data1:=16#0, Data2:=16#0, Data3:=16#0, Data4:=[16#0,16#0,16#0,16#0,
16#0,16#0,16#0,16#0]);
    EMPTY_EVENT_ID    : UDINT := 16#0;
    EMPTY_SEVERITY    : TcEventSeverity := TcEventSeverity.Verbose;
    SUCCESS_EVENT     : TcEventEntry := ( uuidEventClass := EMPTY_EVENT_CLASS, nEventID := EMPTY_EVE
NT_ID, eSeverity := EMPTY_SEVERITY );
END_VAR
```

名称	类型	初始值
EMPTY_EVENT_CLASS	GUID	STRUCT(Data1=16#0, Data2:=16#0, Data3:=16#0, Data4:=[16#0,16#0,16#0,16#0,16#0,16#0,16#0,16#0])
EMPTY_EVENT_ID	UDINT	16#0
EMPTY_SEVERITY	TcEventSeverity [▶ 81]	TcEventSeverity.Verbose
SUCCESS_EVENT	TcEventEntry [▶ 80]	STRUCT(uuidEventClass := EMPTY_EVENT_CLASS, nEventID := EMPTY_EVENT_ID, eSeverity := EMPTY_SEVERITY)

6.4.2 GVL

```
{attribute 'qualified_only'}
VAR_GLOBAL
    nLangId_OnlineMonitoring : DINT := 1033;
END_VAR
```

名称	类型	初始值	描述
nLangId_OnlineMonitoring	DINT	1033	在线监控的语言 ID 英语 (en-US) = 1033 德语 (de-DE) = 1031 …

6.4.3 参数列表

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    cSourceNameSize : UDINT(81..Tc2_Utilities.ParameterList.cMaxCharacters) := 256;
END_VAR
```

名称	类型	初始值	描述
cSourceNameSize	UDINT(81..Tc2_Utilities.ParameterList.cMaxCharacters)	256	源名称的大小 (字节)

6.4.4 Global_Version

所有库都有特定版本。此版本可以在 PLC 库存储库等中查看。
全局常量包含库版本信息 (ST_LibVersion 类型):

Global_Version

```
VAR_GLOBAL CONSTANT
    stLibVersion_Tc3_EventLogger : ST_LibVersion;
END_VAR
```

要检查您拥有的版本是否是所需的版本，请使用函数 F_CmpLibVersion (在 Tc2_System 库中定义)。

7 C++ API

7.1 接口

7.1.1 ITcEvent

此接口为 ITcAlarm 和 ITcMessage 提供常用方法。它用于回调接口 ITcAlarmListener 和 ITcMessageListener。

语法

```
TCOM_DECL_INTERFACE("4A9CB0E9-8969-4B85-B567-605110511200", ITcEvent)
```

方法

名称	描述
GetEventClass [▶ 83]	返回事件类的 GUID。
GetEventId [▶ 83]	返回事件 ID。
GetSeverity [▶ 84]	返回事件的严重级别。
GetSourceInfo [▶ 84]	返回 SourceInfo。
GetJsonAttribute [▶ 84]	返回 JSON 属性。
GetText [▶ 85]	异步返回文本。
GetEventClassName [▶ 85]	异步返回事件类名称。

7.1.1.1 GetEventClass

返回事件类的 GUID。

语法

```
virtual HRESULT TCOMAPI GetEventClass (GUID eventClass)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.2 GetEventId

返回事件 ID。

语法

```
virtual HRESULT TCOMAPI GetEventId (UDINT EventId)
```

参数

名称	类型	描述
eventId	引用 UDINT	引用事件 ID。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.3 GetSeverity

返回事件的严重级别。

语法

```
virtual HRESULT TCOMAPI GetSeverity (TcEventSeverity severity)
```

参数

名称	类型	描述
严重级别	引用 TcEventSeverity	引用事件严重级别。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.4 GetSourceInfo

返回 SourceInfo。

语法

```
virtual HRESULT TCOMAPI GetSourceInfo (ITcSourceInfo pipSourceInfo)
```

参数

名称	类型	描述
pipSourceInfo	指向	引用 SourceInfo 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.5 GetJsonAttribute

返回 JSON 属性。

语法

```
virtual HRESULT TCOMAPI GetJsonAttribute (STRING sJsonAttribute, UDINT nJsonAttribute)
```

参数

名称	类型	描述
sJsonAttribute	引用 STRING	引用 JSON string。
nJsonAttribute	引用 UDINT	引用 Json 属性的长度。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.6 GetText

异步返回文本。

语法

```
virtual HRESULT TCOMAPI GetText (DINT nLangId, ITcAsyncStringResult pipResult)
```

参数

名称	类型	描述
nLangId	DINT	所请求语言的语言 ID (LCID)。
pipResult	指向 ITcAsyncStringResult	引用 ITcAsyncStringResult 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.1.7 GetEventClassName

异步返回事件类名称。

语法

```
virtual HRESULT TCOMAPI GetClassName (DINT nLangId, ITcAsyncStringResult pipResult)
```

参数

名称	类型	描述
nLangId	DINT	所请求语言的语言 ID (LCID)。
pipResult	指向 ITcAsyncStringResult	引用 ITcAsyncStringResult 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.2 ITcMessage

此接口表示来自 TwinCAT 3 EventLogger 的消息。

语法

```
TCOM_DECL_INTERFACE("6474ED2C-E483-454E-A67D-233E6D337C08", ITcMessage)
```

方法

名称	描述
SetJsonAttribute [▶ 86]	设置 JSON 属性。
GetArguments [▶ 86]	返回参数的接口指针。
发送 [▶ 86]	发送消息。

7.1.2.1 SetJsonAttribute

设置 JSON 属性。

语法

```
virtual HRESULT TCOMAPI SetJsonAttribute (STINRG sJsonAttribute)
```

参数

名称	类型	描述
sJsonAttribute	STRING	JSON string

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.2.2 GetArguments

返回参数的接口指针。

语法

```
virtual HRESULT TCOMAPI GetArguments (ITcArguments pipArguments)
```

参数

名称	类型	描述
pipArguments	指向	引用 ITcArguments 接口

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.2.3 发送

发送消息。

语法

```
virtual HRESULT TCOMAPI Send (ULINT timeStamp)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3 ITcAlarm

此接口表示来自 TwinCAT 3 EventLogger 的警报。

语法

```
TCOM_DECL_INTERFACE("EC6D4FF7-5805-4DDB-A316-27894E77D644", ITcAlarm)
```

方法

名称	描述
SetJsonAttribute [▶ 87]	设置 JSON 属性。
GetArguments [▶ 87]	返回参数的接口指针。
GetIsRaised [▶ 88]	如果警报处于已触发状态，则返回 TRUE。
触发 [▶ 88]	将警报状态设置为 Raised（已触发）。
清除 [▶ 89]	将警报状态设置为 Not Raised（未触发）。
GetConfirmationState [▶ 89]	返回确认状态。
确认 [▶ 89]	将警报状态设置为 Confirmed（已确认）。

7.1.3.1 SetJsonAttribute

设置 Json 属性。

语法

```
virtual HRESULT TCOMAPI SetJsonAttribute (STRING sJsonAttribute)
```

参数

名称	类型	描述
sJsonAttribute	STRING	JSON string

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.2 GetArguments

返回参数的接口指针。

语法

```
virtual HRESULT TCOMAPI GetArguments (ITcArguments pipArguments)
```

参数

名称	类型	描述
pipArguments	指向	引用 ITcArguments 接口

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.3 GetIsRaised

如果警报处于 Raised（已触发）状态，则在参数 blsRaised 中返回 TRUE。

语法

```
virtual HRESULT TCOMAPI GetIsRaised (BOOL32 bIsRaised)
```

参数

名称	类型	描述
bIsRaised	引用 BOOL32	引用状态。 如果警报处于已触发状态，则返回 TRUE。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.4 触发

将警报状态 [▶ 11]设置为 Raised（已触发）。

如果警报需要强制确认，则确认状态会另外设置为 WaitForConfirmation。

语法

```
virtual HRESULT TCOMAPI Raise (ULINT timeStamp)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.5 清除

将警报状态 [▶ 11] 设置为 Not Raised（未触发）。

语法

```
virtual HRESULT TCOMAPI Clear (ULINT timeStamp, BOOL32 bResetConfirmation)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
bResetConfirmation	BOOL32	如果为 TRUE 且确认状态为 WaitForConfirmation，则确认状态设置为 Reset（重置）。否则，确认状态不会更改。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.6 GetConfirmationState

返回确认状态 [▶ 11]。

语法

```
virtual HRESULT TCOMAPI GetConfirmationState (… state)
```

参数

名称	类型	描述
状态	引用 TcEventConfirmationState	返回确认状态。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.3.7 确认

将 WaitingForConfirmation 的确认状态 [▶ 11]设置为 Confirmed（已确认）。

语法

```
virtual HRESULT TCOMAPI Confirm (ULINT timeStamp)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4 ITcEventLogger

此接口表示 TwinCAT 3 EventLogger 本身。

语法

```
TCOM_DECL_INTERFACE("B2D5D4E2-07F6-44F4-A292-92CA8035AA86", ITcEventLogger)
```

要求包括：

```
#include "TcRouterInterfaces.h"
#include "TcEventLoggerInterfaces.h"
```

方法

名称	描述
CreateMessage [▶ 90]	创建一个实现 ITcMessage 的实例。
CreateAlarm [▶ 91]	创建一个实现 ITcAlarm 的实例。
GetAlarm [▶ 91]	返回指向现有警报的指针。
IsAlarmRaised [▶ 92]	查询警报是否处于 Raised（已触发）状态。
ConfirmAllAlarms [▶ 92]	为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。
ClearAllAlarms [▶ 93]	为处于 Raised（已触发）状态的所有警报调用 Clear()。
SendTcMessage [▶ 93]	发送消息。
AddMessageListener [▶ 94]	添加消息监听器。
RemoveMessageListener [▶ 94]	移除消息监听器。
NotifyMessageListener [▶ 94]	处理消息监听器队列。
AddAlarmListener [▶ 95]	添加警报监听器。
RemoveAlarmListener [▶ 95]	移除警报监听器。
NotifyAlarmListener [▶ 95]	处理警报监听器队列。
GetEventText [▶ 96]	返回事件文本。
GetEventClassName [▶ 96]	返回事件的类名称。
CreateArguments [▶ 96]	创建一个实现 ITcArguments 的实例。

7.1.4.1 CreateMessage

创建一个实现 ITcMessage 的实例。

语法

```
virtual HRESULT TCOMAPI CreateMessage (GUID eventClass, UDINT eventId, GUID severit, ITcSourceInfo i
pSourceInfo, ITcMessage pipMessage)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
严重级别	引用 TcEventSeverity	引用事件严重级别。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。
pipMessage	ITcMessage [▶ 85]	指向 ITcMessage 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.2 CreateAlarm

创建一个实现 ITcAlarm 的实例。

语法

```
virtual HRESULT TCOMAPI CreateAlarm (GUID eventClass, UDINT eventId, GUID severity, BOOL32 bWithConf
irmation, ITcSourceInfo ipSourceInfo, ITcAlarm pipAlarm)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
严重级别	引用 TcEventSeverity	引用事件严重级别。
bWithConfirmation	BOOL32	定义警报是否需要强制确认。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。
pipAlarm	ITcAlarm [▶ 87]	指向 ITcAlarm 指针。

返回值

类型	描述
HRESULT	如果成功创建了新警报，则返回 S_OK。 如果警报已存在，则返回 ERROR_ALREADY_EXISTS。 如果发生错误，则返回 HRESULT 作为错误代码。

7.1.4.3 GetAlarm

返回指向现有实例的接口指针。

语法

```
virtual HRESULT TCOMAPI GetAlarm (GUID eventClass, UDINT eventId, ITcSourceInfo ipSourceInfo, ITcAla
rm pipAlarm)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。
pipAlarm	ITcAlarm [▶ 87]	指向 ITcAlarm 指针。

返回值

类型	描述
HRESULT	如果未找到实例，则返回 ADS_E_NOTFOUND。 如果全部成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.4 IsAlarmRaised

查询警报是否处于 Raised（已触发）状态。

语法

```
virtual HRESULT TCOMAPI IsAlarmRaised (GUID eventClass, UDINT eventId, BOOL32 bIsRaised, ITcSourceInfo ipSourceInfo)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
bIsRaised	引用 BOOL32	引用状态。 如果警报处于已触发状态，则返回 TRUE。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.5 ConfirmAllAlarms

为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。

语法

```
virtual HRESULT TCOMAPI ConfirmAllAlarms (ULINT timeStamp)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.6 ClearAllAlarms

为处于 Raised（已触发）状态的所有警报调用 Clear()。

语法

```
virtual HRESULT TCOMAPI ClearAllAlarms (ULINT timestamp, BOOL32 bResetConfirmation)
```

参数

名称	类型	描述
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
bResetConfirmation	BOOL32	如果为 TRUE 且确认状态为 WaitForConfirmation，则确认状态设置为 Reset（重置）。否则，确认状态不会更改。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.7 SendTcMessage

发送消息。

语法

```
virtual HRESULT TCOMAPI SendTcMessage (GUID eventClass, UDINT eventId, GUID severity, ITcSourceInfo
ipSourceInfo, ULINT timeStamp, ITcArguments ipSerializedArguments)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
严重级别	引用 TcEventSeverity	引用事件严重级别。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。
timeStamp	ULINT	> 0：外部时间戳以 100 纳秒为单位，自 1 月 1 日起，1601 (UTC)。
ipSerializedArguments	ITcArguments	指向 TcArguments 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.8 AddMessageListener

添加消息监听器。

语法

```
virtual HRESULT TCOMAPI AddMessageListener (ITcMessageListener ipListener, ITcEventFilterConfig pipF
ilterConfig)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。
pipFilterConfig	ITcEventFilterConfig	指向 ITcEventFilterConfig 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.9 RemoveMessageListener

移除消息监听器。

语法

```
virtual HRESULT TCOMAPI RemoveMessageListener (ITcMessageListener ipListener)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.10 NotifyMessageListener

处理消息监听器队列。

语法

```
virtual HRESULT TCOMAPI NotifyMessageListener (ITcMessageListener ipListener)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.11 AddAlarmListener

添加警报监听器。

语法

```
virtual HRESULT TCOMAPI AddAlarmListener (ITcMessageListener ipListener, ITcEventFilterConfig pipFilterConfig)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。
pipFilterConfig	ITcEventFilterConfig	指向 ITcEventFilterConfig 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.12 RemoveAlarmListener

移除警报监听器。

语法

```
virtual HRESULT TCOMAPI RemoveAlarmListener (ITcMessageListener ipListener)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.13 NotifyAlarmListener

处理警报监听器队列。

语法

```
virtual HRESULT TCOMAPI NotifyMessageListener (ITcMessageListener ipListener)
```

参数

名称	类型	描述
ipListener	ITcMessageListener	指向 ITcMessageListene 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.14 GetEventText

返回事件文本。

语法

```
virtual HRESULT TCOMAPI GetEventText (GUID eventClass, UDINT eventId, ITcSourceInfo ipSourceInfo, ITcArguments ipArguments, DINT nLangId, ITcAsyncStringResult pipResult)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
eventId	引用 UDINT	引用事件 ID。
ipSourceInfo	ITcSourceInfo	指向 ITcSourceInfo 接口。
ipArguments	ITcArguments	指向 TcArguments 接口。
nLangId	DINT	所请求语言的语言 ID (LCID)。
pipResult	指向 ITcAsyncStringResult	引用 ITcAsyncStringResult 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.15 GetEventClassName

返回事件的类名称。

语法

```
virtual HRESULT TCOMAPI GetEventClassName (GUID eventClass, DINT nLangId, ITcAsyncStringResult pipResult)
```

参数

名称	类型	描述
eventClass	引用 GUID	引用事件类的 GUID。
nLangId	DINT	所请求语言的语言 ID (LCID)。
pipResult	指向 ITcAsyncStringResult	引用 ITcAsyncStringResult 指针。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.1.4.16 CreateArguments

创建一个实现 ITcArguments 的实例。

语法

```
virtual HRESULT TCOMAPI CreateArguments (ITcArguments ipArguments)
```

参数

名称	类型	描述
ipArguments	ITcArguments	指向 TcArguments 接口。

返回值

类型	描述
HRESULT	如果方法调用成功，则返回 S_OK，否则返回 HRESULT 作为错误代码。

7.2 数据类型

7.2.1 TcEventEntry

通过事件类、事件 ID 和严重级别定义事件。

语法

定义：

```
typedef struct
{
    GUID             uuidEventClass;
    UDINT            nEventId;
    TcEventSeverity  eSeverity;
}TcEventEntry;
```

参数

名称	类型	描述
uuidEventClass	GUID	事件类的 GUID。
nEventId	UDINT	事件 ID。
eSeverity	TcEventSeverity	事件严重级别定义事件的严重级别，

7.2.2 TcEventSeverity

定义事件的严重级别。

语法

定义：

```
typedef enum
{
    Verbose   = 0,
    Info      = 1,
    Warning   = 2,
    Error     = 3,
    Critical  = 4
}TcEventSeverity;
```

7.2.3 TcEventConfirmationState

定义警报的确认状态。

语法

定义：

```
typedef enum
{
   NotSupported        = 0,
   NotRequired         = 1,
   WaitForConfirmation = 2,
   Confirmed           = 3,
   Reset               = 4
}TcEventConfirmationState;
```

8 COM API

可以通过 COM 接口从 Windows 系统访问 EventLogger。

8.1 类

8.1.1 TcEventLogger

此类表示与 TwinCAT 3 Eventlogger 的连接。

语法

```
public class: ITcEventLogger2, _ITcEventLoggerEvents
```

构造函数

初始化 TcEventLogger 类的新建实例。

```
public TcEventLoggerClass();
```

调用：

```
TcEventLogger logger : new TcEventLogger();
```

接口

名称	描述
ITcEventLogger2 [▶ 124]	用于向 EventLogger 发送命令的接口。
_ITcEventLoggerEvents [▶ 108]	此接口提供发生事件的消息。

方法

名称	修饰符	返回类型	定义位置	描述
ClearAllAlarms [▶ 124]	public virtual	void	ITcEventLogger / ITcEventLogger2 [▶ 124]	为处于“Raised (已触发)”状态的所有警报调用 Clear()。
ClearLoggedEvents [▶ 124]	public virtual	void	ITcEventLogger / ITcEventLogger2 [▶ 124]	为当前处于“Raised (已触发)”状态的所有事件调用 Clear()。
ConfirmAllAlarms [▶ 125]	public virtual	void	ITcEventLogger / ITcEventLogger2 [▶ 124]	为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。
连接 [▶ 125]	public virtual	void	ITcEventLogger / ITcEventLogger2 [▶ 124]	连接到指定系统上的 TwinCAT 系统的 EventLogger。
断开 [▶ 125]	public virtual	void	ITcEventLogger / ITcEventLogger2 [▶ 124]	取消与 TwinCAT 系统的 EventLogger 的连接。
GetEventClassName [▶ 125]	public virtual	string	ITcEventLogger / ITcEventLogger2 [▶ 124]	返回事件的类名称。
GetLoggedEvents [▶ 125]	public virtual	TcEventLoggerAdsProxyLib.TcLoggedEventCollection	ITcEventLogger / ITcEventLogger2 [▶ 124]	查询缓存中的事件。
ITcEventLogger2_SendTcMessage [▶ 103]	public virtual	void	ITcEventLogger2 [▶ 124]	发送消息。

事件

名称	返回类型	定义位置	描述
AlarmCleared	_ITcEventLoggerEvents_AlarmClearedEventHandler	_ITcEventLoggerEvents [▶ 108]	如果警报更改为“Not Raised (未触发)”状态，则进行调用。
AlarmConfirmed	_ITcEventLoggerEvents_AlarmConfirmedEventHandler	_ITcEventLoggerEvents [▶ 108]	如果警报更改为“Confirmed (已确认)”状态，则进行调用。
AlarmRaised	_ITcEventLoggerEvents_AlarmRaisedEventHandler	_ITcEventLoggerEvents [▶ 108]	如果警报更改为“Raised (已触发)”状态，则进行调用。
MessageSent	_ITcEventLoggerEvents_MessageSentEventHandler	_ITcEventLoggerEvents [▶ 108]	当发送消息时调用。

属性

名称	修饰符	类型	访问	定义位置	描述
ActiveAlarms [▶ 126]	public virtual	TcAlarmCollection	获取	ITcEventLogger/ ITcEventLogger2 [▶ 124]	返回当前激活的所有警报的集合。
IsConnected [▶ 126]	public virtual	bool	获取	ITcEventLogger/ ITcEventLogger2 [▶ 124]	表示通过 Connect 建立的连接。应定期检查。

还请参阅有关此

- ITcEventLogger_ClearAllAlarms [▶ 101]
- ITcEventLogger_ClearLoggedEvents [▶ 101]
- ITcEventLogger_ConfirmAllAlarms [▶ 101]
- ITcEventLogger_Connect [▶ 101]
- ITcEventLogger_Disconnect [▶ 102]
- ITcEventLogger_GetEventClassName [▶ 102]
- ITcEventLogger_GetLoggedEvents [▶ 102]
- ITcEventLogger_ActiveAlarms [▶ 103]
- ITcEventLogger_IsConnected [▶ 103]

8.1.1.1 ITcEventLogger_ClearAllAlarms

此方法将处于“Raised (已触发)”状态的所有警报设置为“Not Raised (未触发)”。

语法

```
public virtual void ITcEventLogger_ClearAllAlarms([bool bResetConfirmation = True])
```

参数

名称	类型	描述
bResetConfirmation	bool	确定是否应触发警报确认。

8.1.1.2 ITcEventLogger_ClearLoggedEvents

此方法清除事件缓存。
当前的警报状态不会由此改变。

语法

```
public virtual void ITcEventLogger_ClearLoggedEvents()
```

8.1.1.3 ITcEventLogger_ConfirmAllAlarms

此方法确认所有必须确认的警报，即那些处于“WaitForConfirmation”状态的警报。

语法

```
public virtual void ITcEventLogger_ConfirmAllAlarms()
```

8.1.1.4 ITcEventLogger_Connect

此方法基于 AmsNetId 将对象与运行时系统上的 EventLogger 连接。

语法

```
public virtual void ITcEventLogger_Connect([string Address = localhost])
```

参数

名称	类型	描述
地址	string	定义 AmsNetId。

8.1.1.5 ITcEventLogger_Disconnect

此方法断开对象与 EventLogger 的连接。

语法

```
public virtual void ITcEventLogger_Disconnect()
```

8.1.1.6 ITcEventLogger_GetEventClassName

此方法返回与指定 EventClass GUID 匹配的 EventClass 名称。

语法

```
public virtual string ITcEventLogger_GetEventClassName(System.Guid EventClass, int nLangId)
```

参数

名称	类型	描述
EventClass	System.Guid	事件类的 GUID。
langId	int	包含要返回的名称的 LangID。

返回值

名称	类型	描述
ITcEventLogger_GetEventClassName	string	EventClass 的名称。

8.1.1.7 ITcEventLogger_GetLoggedEvents

此方法返回存储的最新事件的集合。

语法

```
public virtual TcLoggedEventCollection ITcEventLogger_GetLoggedEvents(uint nMaxEntiries)
```

参数

名称	类型	描述
nMaxEntiries	uint	要返回的最大事件数。

返回值

名称	类型	描述
ITcEventLogger_GetLoggedEvents	TcLoggedEventCollection	最新事件的集合。

8.1.1.8 ITcEventLogger_GetText

此方法返回事件文本。

语法

```
public virtual string ITcEventLogger_GetText(System.Guid EventClass, uint EventId, uint objectId,
TcEventLoggerAdsProxyLib.TcEventArgumentsInfo pArgInfo, System.IntPtr pArgData, int nLangId)
```

参数

名称	类型	描述
EventClass	System.Guid	事件类的 GUID。
EventId	uint	事件 ID
objectId	uint	
pArgInfo	TcEventLoggerAdsProxyLib.TcEventArgumentsInfo	
pArgData	System.IntPtr	
langId	int	包含要返回的名称的 LangID。

返回值

名称	类型	描述
ITcEventLogger_GetText	string	事件文本。

8.1.1.9 ITcEventLogger_ActiveAlarms

该属性框返回当前激活的警报的集合（激活状态为“Raised（已触发）”或“WaitingForConfirmation”）。

语法

```
public virtual TcEventLoggerAdsProxyLib.TcAlarmCollection ITcEventLogger_ActiveAlarms
```

8.1.1.10 ITcEventLogger_IsConnected

该属性指示 TcEventLogger 对象当前是否连接到目标系统。
为了能够对连接断开做出反应，应该定期检查。

语法

```
public virtual bool ITcEventLogger_IsConnected
```

8.1.1.11 ITcEventLogger2_SendTcMessage

此方法发送消息。

语法

```
public virtual void SendTcMessage(System.Guid EventClass, uint EventId,
TcEventLoggerAdsProxyLib.SeverityLevelEnum severity, string JsonAttribute,
TcEventLoggerAdsProxyLib.TcSourceInfo pSourceInfo, TcEventLoggerAdsProxyLib.TcArguments pArguments)
```

参数

名称	类型	描述
EventClass	System.Guid	事件类的 GUID
EventId	uint	事件 ID
严重级别	TcEventLoggerAdsProxyLib.SeverityLevelEnum [▶ 136]	事件的严重级别
JsonAttribute	string	JSON 属性
pSourceInfo	TcEventLoggerAdsProxyLib.TcSourceInfo	引用源信息
pArguments	TcArguments	引用参数。

8.1.2 TcArguments

可以使用此类定义事件的参数。为此实现了 ITcArguments 接口。

语法

```
public class: ITcArguments
```

构造函数

初始化 TcArguments 类的新建实例。

```
public TcArguments();
```

调用：

```
TcArguments args : new TcArguments();
```

接口

名称	描述
ITcArguments	描述参数的接口。

方法

名称	修饰符	返回类型	定义位置	描述
添加 [▶ 104]	public virtual	void	ITcEventLogger/ ITcArguments	添加参数。
AddV [▶ 105]	public virtual	void	ITcEventLogger/ ITcArguments	添加参数数组。
清除 [▶ 105]	public virtual	void	ITcEventLogger/ ITcArguments	删除所有参数。
GetEnumerator [▶ 105]	public virtual	System.Collections .IEnumerator	ITcEventLogger/ ITcArguments	返回参数的枚举。
删除 [▶ 105]	public virtual	void	ITcEventLogger/ ITcArguments	删除参数。
Set [▶ 118]	public virtual	void	ITcEventLogger/ ITcArguments	设置参数。

属性

名称	修饰符	类型	访问	定义位置	描述
计数 [▶ 106]	public virtual	int	获取	ITcEventLogger/ ITcArguments	参数数量。
此 [uint]	public virtual	ITcLoggedEvent	获取	ITcEventLogger/ ITcArguments	参数所关联的事件。

8.1.2.1 添加

此方法可添加参数。

语法

```
public virtual void Add(object Item)
```

参数

名称	类型	描述
项	对象	要添加的参数。

8.1.2.2 AddV

此方法可添加参数数组。

语法

```
public virtual void AddV(params object[] args)
```

参数

名称	类型	描述
args	params object[]	要添加的参数数组。

8.1.2.3 清除

此方法删除所有参数。

语法

```
public virtual void Clear()
```

8.1.2.4 GetEnumerator

此方法返回参数的枚举。

语法

```
public virtual System.Collections.IEnumerator GetEnumerator()
```

返回值

名称	类型	描述
GetEnumerator	System.Collections.IEnumerator	参数的枚举。

8.1.2.5 删除

此方法删除参数。

语法

```
public virtual void Remove(uint Index)
```

参数

名称	类型	描述
索引	uint	要删除的参数索引。

8.1.2.6 Set

此方法设置参数。

语法

```
public virtual void Set(uint Index, object Item)
```

参数

名称	类型	描述
索引	uint	要设置的参数索引。
项	对象	新建参数。

8.1.2.7 计数

该属性可返回数量。

语法

```
public virtual int Count
```

8.1.2.8 此 [uint]

该属性是参数所引用的事件。

语法

```
public virtual TcEventLoggerAdsProxyLib.TcArgumentEntry this[uint Index]
```

8.1.3 TcSourceInfo

此类描述事件的源信息。

语法

```
public class: ITcSourceInfo
```

构造函数

初始化 TcSourceInfo 类的新建实例。

```
public TcArguments();
```

调用：

```
TcSourceInfo souceInfo: new SourceInfo();
```

方法

名称	修饰符	返回类型	定义位置	描述
IsSourceInfoTypeSupported [▶ 107]	public virtual	bool	ITcEventLogger/ITcSourceInfo	源信息类型的查询选项。

属性

名称	修饰符	类型	访问	定义位置	描述
计数 [▶ 107]	public virtual	int	获取	ITcEventLogger/ITcSourceInfo	SourceInfo 类型的数量。
Guid [▶ 107]	public virtual	System.Guid	获取，设置	ITcEventLogger/ITcSourceInfo	源的 GUID。
Id [▶ 107]	public virtual	uint	获取，设置	ITcEventLogger/ITcSourceInfo	源 ID。
名称 [▶ 108]	public virtual	string	获取，设置	ITcEventLogger/ITcSourceInfo	源的名称。

8.1.3.1 GetData

此方法返回源的数据。

语法

```
public virtual void GetData(uint Index, out TcEventLoggerAdsProxyLib.TcSourceInfoTypeEnum pInfoType,
 System.IntPtr pData, out uint cbData)
```

参数

名称	类型	描述
索引	uint	源的索引。
pInfoType	TcEventLoggerAdsProxyLib.TcSourceInfoTypeEnum [▶ 137]	引用类型信息。
pData	System.IntPtr	引用数据。
cbData	uint	数据长度。

8.1.3.2 IsSourceInfoTypeSupported

此方法可用于检查是否已定义源信息类型。

语法

```
public virtual bool IsSourceInfoTypeSupported(TcSourceInfoTypeEnum infoType)
```

参数

名称	类型	描述
infoType	TcSourceInfoTypeEnum [▶ 137]	源查询类型

返回值

名称	类型	描述
IsSourceInfoTypeSupported	bool	有关是否支持的信息。

8.1.3.3 计数

该属性可返回数量。

语法

```
public virtual int Count
```

8.1.3.4 Guid

该属性可返回源的 GUID。

语法

```
public virtual System.Guid Guid
```

8.1.3.5 Id

该属性可返回 Id。

语法

```
public virtual uint Id
```

8.1.3.6 名称

该属性可返回名称。

语法

```
public virtual string Name
```

8.2 接口

8.2.1 _ITcEventLoggerEvents

此接口提供发生事件的消息。

语法

```
public interface _ITcEventLoggerEvents
```

方法

名称	修饰符	返回类型	描述
AlarmCleared [▶ 108]	public virtual	void	如果警报更改为“Not Raised (未触发)”状态，则进行调用。
AlarmConfirmed [▶ 108]	public virtual	void	如果警报更改为“Confirmed (已确认)”状态，则进行调用。
AlarmRaised [▶ 109]	public virtual	void	如果警报更改为“Raised (已触发)”状态，则进行调用。
MessageSend [▶ 109]	public virtual	void	当发送消息时调用。

8.2.1.1 AlarmCleared

如果警报更改为“Not Raised (未触发)”状态，则进行调用。

语法

```
public virtual void AlarmCleared(TcAlarm evtObj, bool bRemove)
```

参数

名称	类型	描述
evtObj	TcAlarm	警报
bRemove	bool	TRUE = 警报处于“Not Raised (未触发)”状态而不是“Wait for Confirmation (等待确认)”状态。 FALSE = 警报仍在等待确认。

8.2.1.2 AlarmConfirmed

如果警报更改为“Confirmed (已确认)”状态，则进行调用。

语法

```
public virtual void AlarmConfirmed(TcAlarm evtObj, bool bRemove)
```

参数

名称	类型	描述
evtObj	TcAlarm	警报
bRemove	bool	如果警报未处于“Raised（已触发）”状态，则为 TRUE。 如果警报处于“Raised（已触发）”状态，则为 FALSE。

8.2.1.3 AlarmRaised

如果警报更改为“Raised（已触发）”状态，则进行调用。

语法

```
public virtual void AlarmRaised(TcAlarm evtObj)
```

参数

名称	类型	描述
evtObj	TcAlarm	警报

8.2.1.4 MessageSend

当发送消息时调用。

语法

```
void MessageSent(TcMessage evtObj)
```

参数

名称	类型	描述
evtObj	TcMessage	消息

8.2.2 ITcAlarm3

此接口表示警报。

语法

```
public interface ITcAlarm3
```

方法

名称	修饰符	返回类型	描述
确认 [▶ 110]	public virtual	void	确认警报。
GetCauseRemedy [▶ 112]	public virtual	TcCauseRemedyCollection	返回有关原因和纠正措施的信息（如果已对其进行定义）。
GetDetails [▶ 112]	public virtual	TcDetailCollection	返回详细信息。
GetEventClassName [▶ 111]	public virtual	string	返回事件类名称。
GetText [▶ 111]	public virtual	string	返回事件文本，包括参数。
IsSourceInfoTypeSupported [▶ 112]	public virtual	bool	源信息类型的查询选项。

属性

名称	修饰符	类型	访问	描述
ConfirmationState [▶ 112]	public virtual	ConfirmationStateEnum	获取	返回确认状态 [▶ 136]。
EventClass [▶ 113]	public virtual	System.Guid	获取	返回 EventClass GUID。
EventId [▶ 113]	public virtual	uint	获取	返回事件 ID。
EventType [▶ 113]	public virtual	EventTypeEnum	获取	返回事件类型 [▶ 136]。
FileTimeCleared [▶ 113]	public virtual	long	获取	警报更改为 Cleared (已清除) 状态时的时间戳。
FileTimeConfirmed [▶ 113]	public virtual	long	获取	警报已确认时的时间戳。
FileTimeRaised [▶ 113]	public virtual	long	获取	警报更改为 Raised (已触发) 状态时的时间戳。
IsRaised [▶ 113]	public virtual	bool	获取	指示警报是否已更改为 Raised (已触发) 状态。
JsonAttribute [▶ 114]	public virtual	string	获取	JSON 属性。
SeverityLevel [▶ 114]	public virtual	SeverityLevelEnum	获取	严重级别 [▶ 136]。
SourceGuid [▶ 114]	public virtual	System.Guid	获取	返回源的 GUID。
SourceId [▶ 114]	public virtual	uint	获取	返回源 ID。
SourceName [▶ 114]	public virtual	string	获取	返回源的名称。
TimeCleared [▶ 114]	public virtual	System.DateTime	获取	警报更改为 Cleared (已清除) 状态时的时间戳。
TimeConfirmed [▶ 114]	public virtual	System.DateTime	获取	警报已确认时的时间戳。
TimeRaised [▶ 115]	public virtual	Sytem.DateTime	获取	警报更改为 Raised (已触发) 状态时的时间戳。

8.2.2.1 确认

此方法可确认警报。

语法

```
public virtual void Confirm()
```

8.2.2.2 GetArgumentData

此方法返回参数数据。

语法

```
public virtual void GetArgumentData(ref System.IntPtr ppArgData, uint size)
```

参数

名称	类型	描述
ppArgData	ref System.IntPtr	引用数据。
大小	Uint	大小

8.2.2.3 GetArgumentInfo

此方法返回参数的（类型）信息。

语法

```
public virtual void GetArgumentInfo(ref TcEventArgumentsInfo pArgInfo)
```

参数

名称	类型	描述
pArgInfo	ref TcEventArgumentsInfo	引用所提供的信息。

8.2.2.4 GetEventClassName

此方法返回 EventClass 名称。

语法

```
public virtual string GetEventClassName(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetEventClassName	string	EventClass 的名称

8.2.2.5 GetText

此方法返回事件文本，包括参数。

语法

```
public virtual string GetText(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetText	string	事件文本

8.2.2.6 IsSourceInfoTypeSupported

此方法可用于检查是否已定义源信息类型。

语法

```
public virtual bool IsSourceInfoTypeSupported(TcSourceInfoTypeEnum infoType)
```

参数

名称	类型	描述
infoType	TcSourceInfoTypeEnum [▶ 137]	源查询类型

返回值

名称	类型	描述
IsSourceInfoTypeSupported	bool	有关是否支持的信息。

8.2.2.7 GetCauseRemedy

如果已进行定义，此方法将返回原因/纠正措施信息。

语法

```
public virtual TcCauseRemedyCollection GetCauseRemedy(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID。

返回值

名称	类型	描述
GetCauseRemedy	TcCauseRemedyCollection	原因/纠正措施信息集合。

8.2.2.8 ConfirmationState

该属性可返回 Confirmation State (确认状态)。

语法

```
Public virtual ConfirmationStateEnum ConfirmationState
```

8.2.2.9 GetDetails

此方法返回详细信息。

语法

```
public virtual TcDetailCollection GetDetails(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetDetails	TcDetailCollection	详细信息集合

8.2.2.10 EventClass

该属性可返回 EventClass GUID。

语法

```
public virtual System.Guid EventClass
```

8.2.2.11 EventId

该属性可返回事件 Id。

语法

```
public virtual uint EventId
```

8.2.2.12 EventType

该属性可返回事件类型。

语法

```
public virtual EventTypeEnum EventType
```

8.2.2.13 FileTimeCleared

该属性可返回警报更改为 Cleared（已清除）状态时的时间戳。

语法

```
public virtual long FileTimeCleared
```

8.2.2.14 FileTimeConfirmed

该属性可返回警报更改为 Confirmed（已确认）状态时的时间戳。

语法

```
public virtual long FileTimeConfirmed
```

8.2.2.15 FileTimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual long FileTimeRaised
```

8.2.2.16 IsRaised

该属性指示警报是否已更改为 Raised（已触发）状态。

语法

```
public virtual bool IsRaised
```

8.2.2.17 JsonAttribute

该属性可返回 JSON 属性。

语法

```
public virtual string JsonAttribute
```

8.2.2.18 SeverityLevel

该属性可返回严重级别。

语法

```
public virtual TcEventLoggerAdsProxyLib.SeverityLevelEnum SeverityLevel
```

8.2.2.19 SourceGuid

该属性可返回源的 GUID。

语法

```
public virtual System.Guid SourceGuid
```

8.2.2.20 SourceId

该属性可返回源 ID。

语法

```
public virtual uint SourceId
```

8.2.2.21 SourceName

该属性可返回源的名称。

语法

```
public virtual string SourceName
```

8.2.2.22 TimeCleared

该属性可返回警报更改为 Cleared（已清除）状态时的时间戳。

语法

```
public virtual System.DateTime TimeCleared
```

8.2.2.23 TimeConfirmed

该属性可返回警报更改为 Confirmed（已确认）状态时的时间戳。

语法

```
public virtual System.DateTime TimeConfirmed
```

8.2.2.24 TimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual System.DateTime TimeRaised
```

8.2.3 ITcArgumentEntry

此接口描述参数。

语法

```
public interface ITcArgumentEntry
```

方法

名称	修饰符	返回类型	描述
获取 [▶ 115]	public virtual	动态	返回动态类型的值。
GetBoolean [▶ 115]	public virtual	bool	返回布尔值。
GetDouble [▶ 116]	public virtual	void	返回 Double 值。
GetFloat [▶ 116]	public virtual	float	返回浮点型的值。
GetInt16 [▶ 116]	public virtual	short	返回 Short 值。
GetInt32 [▶ 117]	public virtual	int	返回 Int 值。
GetInt64 [▶ 117]	public virtual	long	返回 Long 值。
GetInt8 [▶ 117]	public virtual	sbyte	返回 SByte 值。
GetString [▶ 117]	public virtual	string	返回 String 值。
GetUInt16 [▶ 118]	public virtual	ushort	返回 UShort 值。
GetUInt32 [▶ 118]	public virtual	uint	返回 UInt 值。
GetUInt64 [▶ 118]	public virtual	ulong	返回 ULong 值。
GetUInt8 [▶ 118]	public virtual	byte	返回 Byte 值。
Set [▶ 118]	public virtual	void	设置值。

还请参阅有关此

GetData [▶ 107]

8.2.3.1 获取

此方法返回动态类型的值。

语法

```
public virtual Object Get()
```

返回值

名称	类型	描述
获取	对象	Return 是与参数相对应的类型的对象。

8.2.3.2 GetBoolean

此方法返回布尔值。

语法

```
public virtual bool GetBoolean()
```

返回值

名称	类型	描述
GetBoolean	bool	Return

8.2.3.3 GetData

此方法返回数据引用值。

语法

```
public virtual void GetData(out TcEventArgumentTypeEnum pInfoType, System.IntPtr pData, out uint
cbData)
```

参数

名称	类型	描述
pInfoType	TcEventArgumentTypeEnum [▶ 137]	引用数据类型。
pData	System.IntPtr	引用数据。
cbData	uint	数据长度。

8.2.3.4 GetDouble

此方法返回 Double 值。

语法

```
public virtual double GetDouble
```

返回值

名称	类型	描述
GetDouble	double	Return

8.2.3.5 GetFloat

此方法返回 Float 值。

语法

```
public virtual float GetFloat()
```

返回值

名称	类型	描述
GetFloat	float	Return

8.2.3.6 GetInt16

此方法返回 Short 值。

语法

```
public virtual short GetInt16()
```

返回值

名称	类型	描述
GetInt16	short	Return

8.2.3.7 GetInt32

此方法返回 Int 值。

语法

```
public virtual int GetInt32()
```

返回值

名称	类型	描述
GetInt32	int	Return

8.2.3.8 GetInt64

此方法返回 Long 值。

语法

```
public virtual long GetInt64()
```

返回值

名称	类型	描述
GetInt64	long	Return

8.2.3.9 GetInt8

此方法返回 SByte 值。

语法

```
public virtual sbyte GetInt8()
```

返回值

名称	类型	描述
GetInt8	sbyte	Return

8.2.3.10 GetString

此方法返回 String 值。

语法

```
public virtual string GetString()
```

返回值

名称	类型	描述
GetString	string	Return

8.2.3.11 GetUInt16

此方法返回 UShort 值。

语法

```
public virtual ushort GetUInt16()
```

返回值

名称	类型	描述
GetUInt16	ushort	Return

8.2.3.12 GetUInt32

此方法返回 UInt 值。

语法

```
public virtual uint GetUInt32()
```

返回值

名称	类型	描述
GetUInt32	uint	Return

8.2.3.13 GetUInt64

此方法返回 ULong 值。

语法

```
public virtual ulong GetUInt64()
```

返回值

名称	类型	描述
GetUInt64	ulong	Return

8.2.3.14 GetUInt8

此方法返回 Byte 值。

语法

```
public virtual byte GetUInt8()
```

返回值

名称	类型	描述
GetUInt8	byte	Return

8.2.3.15 Set

此方法可设置值。

语法

```
public virtual void Set(object Item)
```

参数

名称	类型	描述
项	对象	新建参数。与参数相对应的类型的对象。

8.2.4 ITcCauseRemedy

此接口描述事件的原因/纠正措施信息。

语法

```
public interface ITcCauseRemedy
```

属性

名称	修饰符	类型	访问	描述
原因 [▶ 119]	public virtual	string	获取	原因。
Id [▶ 119]	public virtual	uint	获取	ID。
纠正措施 [▶ 119]	public virtual	string	获取	纠正措施。

8.2.4.1 原因

该属性可返回原因。

语法

```
public virtual string Cause
```

8.2.4.2 Id

该属性可返回 Id。

语法

```
public virtual uint Id
```

8.2.4.3 纠正措施

该属性可返回纠正措施。

语法

```
public virtual string Remedy
```

8.2.5 ITcDetail

此接口描述事件的详细信息。

语法

```
public interface ITcDetail
```

属性

名称	修饰符	类型	访问	描述
注释 [▶ 120]	public virtual	string	获取	注释
名称 [▶ 120]	public virtual	string	获取	名称
文本 [▶ 120]	public virtual	string	获取	文本

8.2.5.1 注释

该属性可返回详细信息的注释。

语法

```
public virtual string Comment
```

8.2.5.2 名称

该属性可返回名称。

语法

```
public virtual string Name
```

8.2.5.3 文本

该属性可返回详细信息的文本。

语法

```
public virtual string text
```

8.2.6 ITcEvent

此接口描述事件。此事件可以是消息，也可以是警报。

语法

```
public interface ITcEvent
```

方法

名称	修饰符	返回类型	描述
GetArgumentData [▶ 121]	public virtual	void	返回参数的数据。
GetArgumentInfo [▶ 121]	public virtual	void	返回参数的（类型）信息。
GetEventClassName [▶ 121]	public virtual	string	返回事件类名称。
GetText [▶ 122]	public virtual	string	返回事件文本，包括参数。
IsSourceInfoTypeSupported [▶ 122]	public virtual	bool	源信息类型的查询选项。

属性

名称	修饰符	类型	访问	描述
EventClass [▶ 122]	public virtual	System.Guid	获取	返回 EventClass GUID。
EventId [▶ 122]	public virtual	uint	获取	返回事件 ID。
EventType [▶ 123]	public virtual	EventTypeEnum	获取	返回事件类型 [▶ 136]。
FileTimeRaised [▶ 123]	public virtual	long	获取	发送事件时的时间戳。
SeverityLevel [▶ 123]	public virtual	SeverityLevelEnum	获取	严重级别 [▶ 136]。
SourceGuid [▶ 123]	public virtual	System.Guid	获取	返回源的 GUID。
SourceId [▶ 123]	public virtual	uint	获取	返回源 ID。
SourceName [▶ 123]	public virtual	string	获取	返回源的名称。
TimeRaised [▶ 123]	public virtual	System.DateTime	获取	发送事件时的时间戳。

8.2.6.1 GetArgumentData

此方法返回参数数据。

语法

```
public virtual void GetArgumentData(ref System.IntPtr ppArgData, uint size)
```

参数

名称	类型	描述
ppArgData	ref System.IntPtr	引用数据。
大小	Uint	大小

8.2.6.2 GetArgumentInfo

此方法返回参数的（类型）信息。

语法

```
public virtual void GetArgumentInfo(ref TcEventArgumentsInfo pArgInfo)
```

参数

名称	类型	描述
pArgInfo	ref TcEventArgumentsInfo	引用所提供的信息。

8.2.6.3 GetEventClassName

此方法返回 EventClass 名称。

语法

```
public virtual string GetEventClassName(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetEventClassName	string	EventClass 的名称

8.2.6.4 GetText

此方法返回事件文本，包括参数。

语法

```
public virtual string GetText(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetText	string	事件文本

8.2.6.5 IsSourceInfoTypeSupported

此方法可用于检查是否已定义源信息类型。

语法

```
public virtual bool IsSourceInfoTypeSupported(TcSourceInfoTypeEnum infoType)
```

参数

名称	类型	描述
infoType	TcSourceInfoTypeEnum [▶ 137]	源查询类型

返回值

名称	类型	描述
IsSourceInfoTypeSupported	bool	有关是否支持的信息。

8.2.6.6 EventClass

该属性可返回 EventClass GUID。

语法

```
public virtual System.Guid EventClass
```

8.2.6.7 EventId

该属性可返回事件 Id。

语法

```
public virtual uint EventId
```

8.2.6.8 EventType

该属性可返回事件类型。

语法

```
public virtual EventTypeEnum EventType
```

8.2.6.9 FileTimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual long FileTimeRaised
```

8.2.6.10 SeverityLevel

该属性可返回严重级别。

语法

```
public virtual TcEventLoggerAdsProxyLib.SeverityLevelEnum SeverityLevel
```

8.2.6.11 SourceGuid

该属性可返回源的 GUID。

语法

```
public virtual System.Guid SourceGuid
```

8.2.6.12 SourceId

该属性可返回源 ID。

语法

```
public virtual uint SourceId
```

8.2.6.13 SourceName

该属性可返回源的名称。

语法

```
public virtual string SourceName
```

8.2.6.14 TimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual System.DateTime TimeRaised
```

8.2.7 ITcEventLogger2

用于向 EventLogger 发送命令的接口。

语法

```
public interface ITcEventLogger2
```

方法

名称	修饰符	返回类型	描述
ClearAllAlarms [▶ 124]	public virtual	void	为处于“Raised (已触发)”状态的所有警报调用 Clear()。
ClearLoggedEvents [▶ 124]	public virtual	void	清除缓存。
ConfirmAllAlarms [▶ 125]	public virtual	void	为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。
连接 [▶ 125]	public virtual	void	连接到指定系统上的 TwinCAT 系统的 EventLogger。
断开 [▶ 125]	public virtual	void	取消与 TwinCAT 系统的 EventLogger 的连接。
GetEventClassName [▶ 125]	public virtual	string	返回事件的类名称。
GetLoggedEvents [▶ 125]	public virtual	TcLoggedEventCollection	查询缓存中的事件。

属性

名称	修饰符	类型	访问	描述
ActiveAlarms [▶ 126]	public virtual	TcAlarmCollection	获取	返回当前激活的所有警报的集合。
IsConnected [▶ 126]	public virtual	bool	获取	表示通过 Connect 建立的连接。应定期检查。

还请参阅有关此

ITcEventLogger2_SendTcMessage [▶ 126]

8.2.7.1 ClearAllAlarms

为处于“Raised (已触发)”状态的所有警报调用 Clear()。

语法

```
public virtual void ClearAllAlarms([bool bResetConfirmation = True])
```

参数

名称	类型	描述
bResetConfirmation	bool	指示是否还应将警报更改为“Confirmed (已确认)”状态。

8.2.7.2 ClearLoggedEvents

清除缓存。

语法

```
Public virtual void ClearLoggedEvents()
```

8.2.7.3 ConfirmAllAlarms

为确认状态为 WaitForConfirmation 的所有警报调用 Confirm()。

语法

```
public virtual void ConfirmAllAlarms()
```

8.2.7.4 连接

连接到指定系统上的 TwinCAT 系统的 EventLogger。

语法

```
public virtual void Connect([string Address = localhost])
```

参数

名称	类型	描述
地址	string	AMS Net ID

8.2.7.5 断开

取消与 TwinCAT 系统的 EventLogger 的连接。

语法

```
public virtual void Disconnect()
```

8.2.7.6 GetEventClassName

返回事件的类名称。

语法

```
public virtual string GetEventClassName(System.Guid EventClass, int nLangId)
```

参数

名称	类型	描述
langId	int	语言的 LangID
EventClass	System.Guid	事件类的 GUID

返回值

名称	类型	描述
GetEventClassName	string	

8.2.7.7 GetLoggedEvents

查询缓存中的事件。

语法

```
public virtual TcEventLoggerAdsProxyLib.TcLoggedEventCollection GetLoggedEvents(uint nMaxEntiries)
```

参数

名称	类型	描述
nMaxEntiries	uint	要返回的事件数。

返回值

名称	类型	描述
GetLoggedEvents	TcEventLoggerAdsProxyLib.TcLoggedEventCollection	

8.2.7.8 ActiveAlarms

返回当前激活的所有警报的集合。

语法

```
public virtual TcEventLoggerAdsProxyLib.TcAlarmCollection ActiveAlarms
```

8.2.7.9 IsConnected

表示通过 Connect 建立的连接。应定期检查。

语法

```
public virtual bool IsConnected
```

8.2.7.10 ITcEventLogger2_SendTcMessage

此方法发送消息。

语法

```
public virtual void SendTcMessage(System.Guid EventClass, uint EventId,
TcEventLoggerAdsProxyLib.SeverityLevelEnum severity, string JsonAttribute,
TcEventLoggerAdsProxyLib.TcSourceInfo pSourceInfo, TcEventLoggerAdsProxyLib.TcArguments pArguments)
```

参数

名称	类型	描述
EventClass	System.Guid	事件类的 GUID
EventId	uint	事件 ID
严重级别	TcEventLoggerAdsProxyLib.SeverityLevelEnum [▶ 136]	事件的严重级别
JsonAttribute	string	JSON 属性
pSourceInfo	TcEventLoggerAdsProxyLib.TcSourceInfo	引用源信息
pArguments	TcArguments	引用参数。

8.2.8 ITcLoggedEvent4

此接口描述存储的事件。此事件可以是消息，也可以是警报。

语法

```
public interface ITcLoggedEvent4
```

方法

名称	修饰符	返回类型	描述
GetArgumentData [▶ 127]	public virtual	void	返回参数的数据。
GetArgumentInfo [▶ 128]	public virtual	void	返回参数的（类型）信息。
GetCauseRemedy [▶ 128]	public virtual	TcCauseRemedyCollection	返回原因/纠正措施信息。
GetDetails [▶ 128]	public virtual	TcDetailCollection	返回详细信息。
GetEventClassName [▶ 129]	public virtual	string	返回事件类名称。
GetText [▶ 129]	public virtual	string	返回事件文本，包括参数。
IsSourceInfoTypeSupported [▶ 129]	public virtual	bool	源信息类型的查询选项。

属性

名称	修饰符	类型	访问	描述
EventClass [▶ 129]	public virtual	System.Guid	获取	返回 EventClass GUID。
EventId [▶ 130]	public virtual	uint	获取	返回事件 ID。
EventType [▶ 130]	public virtual	EventTypeEnum	获取	返回事件类型 [▶ 136]。
FileTimeCleared [▶ 130]	public virtual	long	获取	仅警报：警报更改为 Cleared（已清除）状态时的时间戳。
FileTimeConfirmed [▶ 130]	public virtual	long	获取	仅警报：警报已确认时的时间戳。
FileTimeRaised [▶ 130]	public virtual	long	获取	发送事件/警报更改为 Raised（已触发）状态时的时间戳。
JsonAttribute [▶ 130]	public virtual	string	获取	JSON 属性。
SeverityLevel [▶ 130]	public virtual	SeverityLevelEnum	获取	严重级别 [▶ 136]。
SourceGuid [▶ 131]	public virtual	System.Guid	获取	返回源的 GUID。
SourceId [▶ 131]	public virtual	uint	获取	返回源 ID。
SourceName [▶ 131]	public virtual	string	获取	返回源的名称。
TimeCleared [▶ 131]	public virtual	System.DateTime	获取	仅警报：警报更改为 Cleared（已清除）状态时的时间戳。
TimeConfirmed [▶ 131]	public virtual	System.DateTime	获取	仅警报：警报已确认时的时间戳。
TimeRaised [▶ 131]	public virtual	System.DateTime	获取	发送事件/警报更改为 Raised（已触发）状态时的时间戳。
WithConfirmation [▶ 131]	public virtual	bool	获取	仅警报：需要确认。

8.2.8.1 GetArgumentData

此方法返回参数数据。

语法

```
public virtual void GetArgumentData(ref System.IntPtr ppArgData, uint size)
```

参数

名称	类型	描述
ppArgData	ref System.IntPtr	引用数据。
大小	Uint	大小

8.2.8.2 GetArgumentInfo

此方法返回参数的（类型）信息。

语法

```
public virtual void GetArgumentInfo(ref TcEventArgumentsInfo pArgInfo)
```

参数

名称	类型	描述
pArgInfo	ref TcEventArgumentsInfo	引用所提供的信息。

8.2.8.3 GetCauseRemedy

如果已进行定义，此方法将返回原因/纠正措施信息。

语法

```
public virtual TcCauseRemedyCollection GetCauseRemedy(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID。

返回值

名称	类型	描述
GetCauseRemedy	TcCauseRemedyCollection	原因/纠正措施信息集合。

8.2.8.4 GetDetails

此方法返回详细信息。

语法

```
public virtual TcDetailCollection GetDetails(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetDetails	TcDetailCollection	详细信息集合

8.2.8.5 GetEventClassName

此方法返回 EventClass 名称。

语法

```
public virtual string GetEventClassName(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetEventClassName	string	EventClass 的名称

8.2.8.6 GetText

此方法返回事件文本，包括参数。

语法

```
public virtual string GetText(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetText	string	事件文本

8.2.8.7 IsSourceInfoTypeSupported

此方法可用于检查是否已定义源信息类型。

语法

```
public virtual bool IsSourceInfoTypeSupported(TcSourceInfoTypeEnum infoType)
```

参数

名称	类型	描述
infoType	TcSourceInfoTypeEnum [▶ 137]	源查询类型

返回值

名称	类型	描述
IsSourceInfoTypeSupported	bool	有关是否支持的信息。

8.2.8.8 EventClass

该属性可返回 EventClass GUID。

语法

```
public virtual System.Guid EventClass
```

8.2.8.9 EventId

该属性可返回事件 Id。

语法

```
public virtual uint EventId
```

8.2.8.10 EventType

该属性可返回事件类型。

语法

```
public virtual EventTypeEnum EventType
```

8.2.8.11 FileTimeCleared

该属性可返回警报更改为 Cleared（已清除）状态时的时间戳。

语法

```
public virtual long FileTimeCleared
```

8.2.8.12 FileTimeConfirmed

该属性可返回警报更改为 Confirmed（已确认）状态时的时间戳。

语法

```
public virtual long FileTimeConfirmed
```

8.2.8.13 FileTimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual long FileTimeRaised
```

8.2.8.14 JsonAttribute

该属性可返回 JSON 属性。

语法

```
public virtual string JsonAttribute
```

8.2.8.15 SeverityLevel

该属性可返回严重级别。

语法

```
public virtual TcEventLoggerAdsProxyLib.SeverityLevelEnum SeverityLevel
```

8.2.8.16 SourceGuid

该属性可返回源的 GUID。

语法

```
public virtual System.Guid SourceGuid
```

8.2.8.17 SourceId

该属性可返回源 ID。

语法

```
public virtual uint SourceId
```

8.2.8.18 SourceName

该属性可返回源的名称。

语法

```
public virtual string SourceName
```

8.2.8.19 TimeCleared

该属性可返回警报更改为 Cleared（已清除）状态时的时间戳。

语法

```
public virtual System.DateTime TimeCleared
```

8.2.8.20 TimeConfirmed

该属性可返回警报更改为 Confirmed（已确认）状态时的时间戳。

语法

```
public virtual System.DateTime TimeConfirmed
```

8.2.8.21 TimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual System.DateTime TimeRaised
```

8.2.8.22 WithConfirmation

该属性描述是否需要确认。

语法

```
public virtual bool WithConfirmation
```

8.2.9 ITcMessage3

此接口表示消息。

语法

```
public interface ITcMessage
```

方法

名称	修饰符	返回类型	描述
GetArgumentData [▶ 132]	public virtual	void	返回参数的数据。
GetArgumentInfo [▶ 133]	public virtual	void	返回参数的（类型）信息。
GetCauseRemedy [▶ 133]	public virtual	TcCauseRemedyCollection	返回原因/纠正措施信息。
GetDetails [▶ 133]	public virtual	TcDetailCollection	返回详细信息。
GetEventClassName [▶ 133]	public virtual	string	返回 EventClass 名称。
GetText [▶ 134]	public virtual	string	返回事件文本，包括参数。
IsSourceInfoTypeSupported [▶ 134]	public virtual	bool	源信息类型的查询选项。

属性

名称	修饰符	类型	访问	描述
EventClass [▶ 134]	public virtual	System.Guid	获取	返回 EventClass GUID。
EventId [▶ 134]	public virtual	uint	获取	返回事件 ID。
EventType [▶ 135]	public virtual	EventTypeEnum	获取	返回事件类型 [▶ 136]。
FileTimeRaised [▶ 135]	public virtual	long	获取	发送消息时的时间戳。
JsonAttribute [▶ 135]	public virtual	string	获取	JSON 属性。
SeverityLevel [▶ 135]	public virtual	SeverityLevelEnum	获取	严重级别 [▶ 136]。
SourceGuid [▶ 135]	public virtual	System.Guid	获取	返回源的 GUID。
SourceId [▶ 135]	public virtual	uint	获取	返回源 ID。
SourceName [▶ 135]	public virtual	string	获取	返回源的名称。
TimeRaised [▶ 136]	public virtual	System.DateTime	获取	发送消息时的时间戳。

8.2.9.1 GetArgumentData

此方法返回参数数据。

语法

```
public virtual void GetArgumentData(ref System.IntPtr ppArgData, uint size)
```

参数

名称	类型	描述
ppArgData	ref System.IntPtr	引用数据。
大小	Uint	大小

8.2.9.2 GetArgumentInfo

此方法返回参数的（类型）信息。

语法

```
public virtual void GetArgumentInfo(ref TcEventArgumentsInfo pArgInfo)
```

参数

名称	类型	描述
pArgInfo	ref TcEventArgumentsInfo	引用所提供的信息。

8.2.9.3 GetCauseRemedy

如果已进行定义，此方法将返回原因/纠正措施信息。

语法

```
public virtual TcCauseRemedyCollection GetCauseRemedy(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID。

返回值

名称	类型	描述
GetCauseRemedy	TcCauseRemedyCollection	原因/纠正措施信息集合。

8.2.9.4 GetDetails

此方法返回详细信息。

语法

```
public virtual TcDetailCollection GetDetails(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetDetails	TcDetailCollection	详细信息集合

8.2.9.5 GetEventClassName

此方法返回 EventClass 名称。

语法

```
public virtual string GetEventClassName(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetEventClassName	string	EventClass 的名称

8.2.9.6 GetText

此方法返回事件文本，包括参数。

语法

```
public virtual string GetText(int langId)
```

参数

名称	类型	描述
langId	int	语言的 LangID

返回值

名称	类型	描述
GetText	string	事件文本

8.2.9.7 IsSourceInfoTypeSupported

此方法可用于检查是否已定义源信息类型。

语法

```
public virtual bool IsSourceInfoTypeSupported(TcSourceInfoTypeEnum infoType)
```

参数

名称	类型	描述
infoType	TcSourceInfoTypeEnum [▶ 137]	源查询类型

返回值

名称	类型	描述
IsSourceInfoTypeSupported	bool	有关是否支持的信息。

8.2.9.8 EventClass

该属性可返回 EventClass GUID。

语法

```
public virtual System.Guid EventClass
```

8.2.9.9 EventId

该属性可返回事件 Id。

语法

```
public virtual uint EventId
```

8.2.9.10 EventType

该属性可返回事件类型。

语法

```
public virtual EventTypeEnum EventType
```

8.2.9.11 FileTimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual long FileTimeRaised
```

8.2.9.12 JsonAttribute

该属性可返回 JSON 属性。

语法

```
public virtual string JsonAttribute
```

8.2.9.13 SeverityLevel

该属性可返回严重级别。

语法

```
public virtual TcEventLoggerAdsProxyLib.SeverityLevelEnum SeverityLevel
```

8.2.9.14 SourceGuid

该属性可返回源的 GUID。

语法

```
public virtual System.Guid SourceGuid
```

8.2.9.15 SourceId

该属性可返回源 ID。

语法

```
public virtual uint SourceId
```

8.2.9.16 SourceName

该属性可返回源的名称。

语法

```
public virtual string SourceName
```

8.2.9.17 TimeRaised

该属性可返回警报更改为 Raised（已触发）状态时的时间戳。

语法

```
public virtual System.DateTime TimeRaised
```

8.3 数据类型

8.3.1 ConfirmationStateEnum

定义警报的确认状态。

语法

```
public enum ConfirmationStateEnum
{
    Confirmed,
    NotRequired,
    NotSupported,
    Reset,
    WaitForConfirmation
}
```

参数

名称	描述
已确认	已确认
NotRequired	在当前状态下无需确认。(警报当前未处于 Raised (已触发) 状态)。
NotSupported (不支持)	未经确认进行了初始化。
Reset (重置)	初始状态
WaitForConfirmation (待确认)	等待确认。

8.3.2 EventTypeEnum

类型定义 TcEvent 是警报类型还是消息类型。

语法

```
public enum EventTypeEnum
{
    Alarm,
    Message
}
```

参数

名称	描述
警报	TcEvent 的类型为 TcAlarm。
消息	TcEvent 的类型为 TcMessage。

8.3.3 SeverityLevelEnum

此枚举定义事件的“严重级别”。这是一个有序列表。

语法

```
public enum SeverityLevelEnum
{
    Critical,
    Error,
    Warning,
    Info,
    Verbose
}
```

参数

	名称	描述
4	Critical	评论
3	Error	错误
2	Warning	警告
1	Info	信息
0	Verbose	扩展输出

8.3.4 TcEventArgumentTypeEnum

对于 TcArgument 是什么类型的类型定义。

语法

```
public enum TcEventArgumentTypeEnum
{
    Blob,
    Boolean,
    Char,
    Double,
    E_AdsnotificationStream,
    EventReference,
    ExternalTimeStamp,
    Float,
    FormatString,
    Int16,
    Int32,
    Int64,
    Int8,
    StringType,
    UInt16,
    UInt32,
    UInt64,
    UInt8,
    Undefinied,
    UTF8EncodedString,
    WChar,
    WStringType
}
```

8.3.5 TcSourceInfoTypeEnum

定义在 TcSourceInfo 中标识哪个条目。

语法

```
public enum TcSourceInfoTypeEnum
{
    SourceGuid,
    SourceId,
    SourceName
}
```

参数

名称	描述
SourceGuid	TcSourceInfo 的源 GUID。
SourceId	TcSourceInfo 的源 ID。例如，TcCOM 对象 ID。
SourceName	TcSourceInfo 的 SourceName。例如，PLC 中的实例路径。

9 示例

9.1 PLC

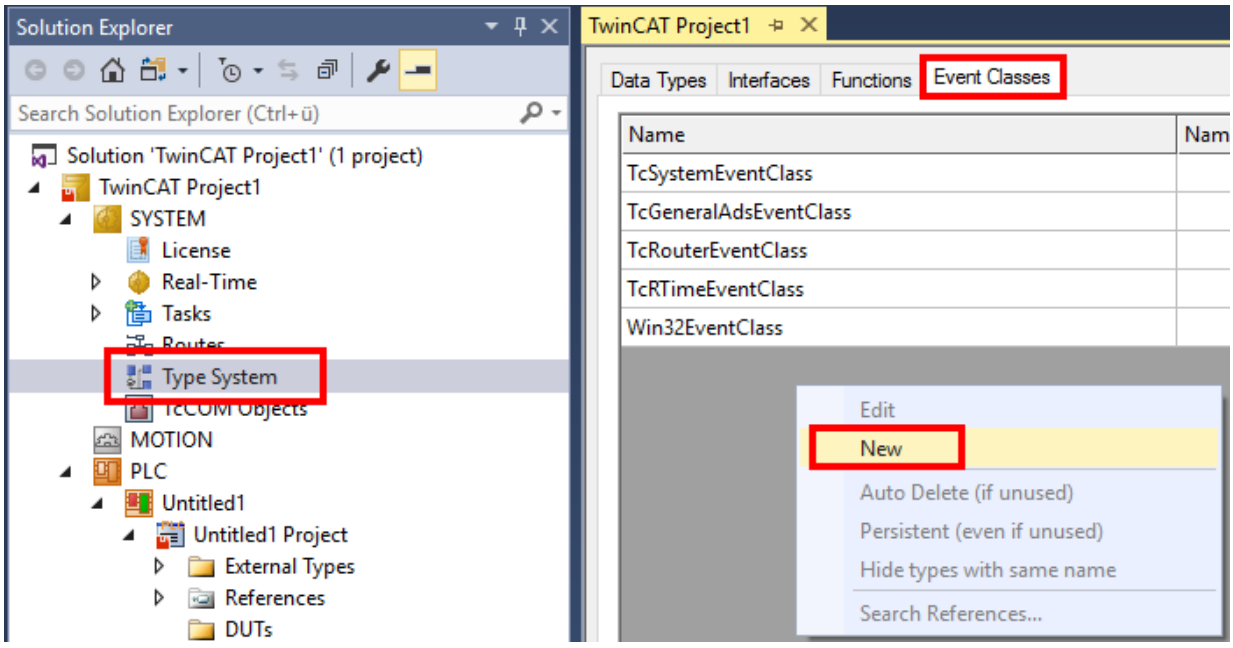
9.1.1 教程

该教程说明从一个空的 TwinCAT 项目到已调度消息的工作步骤。它描述了工作序列中技术简介 [▶ 10]部分所述的 TwinCAT 3 EventLogger 的属性。

在 TwinCAT 类型系统中创建事件类

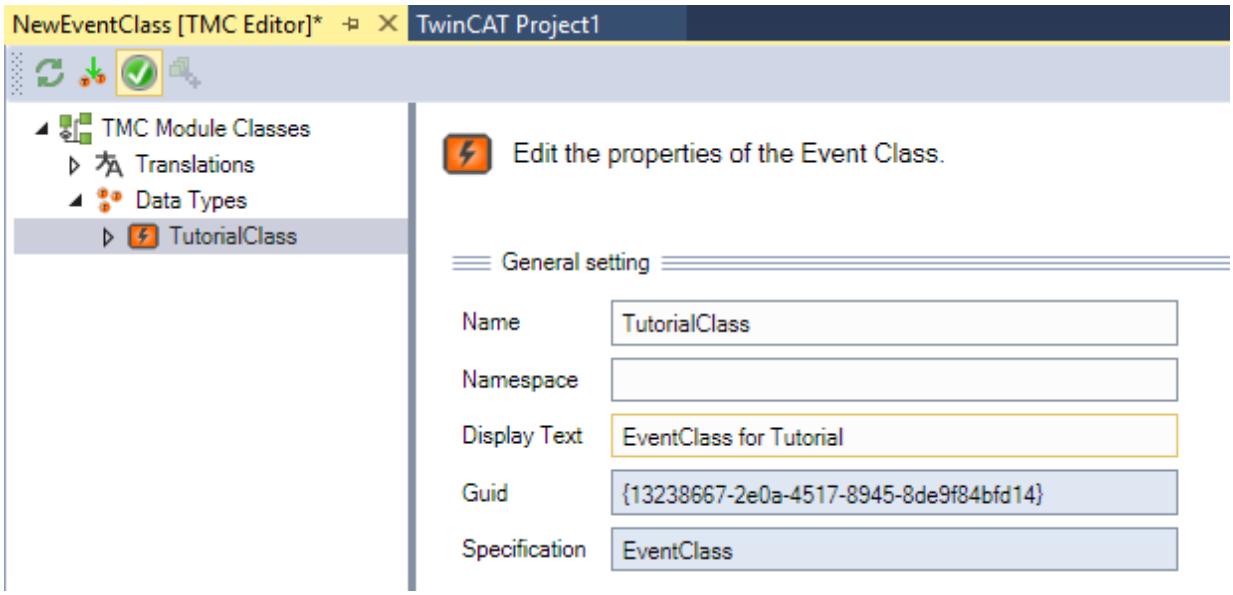
✓ 存在标准 TwinCAT PLC 项目。

1. 双击 SYSTEM 子树中的 **Type System (类型系统)**，在编辑器中选择 **Event Classes (事件类)** 选项卡，然后将其打开。打开上下文菜单，然后选择 **New (新建)** 命令。

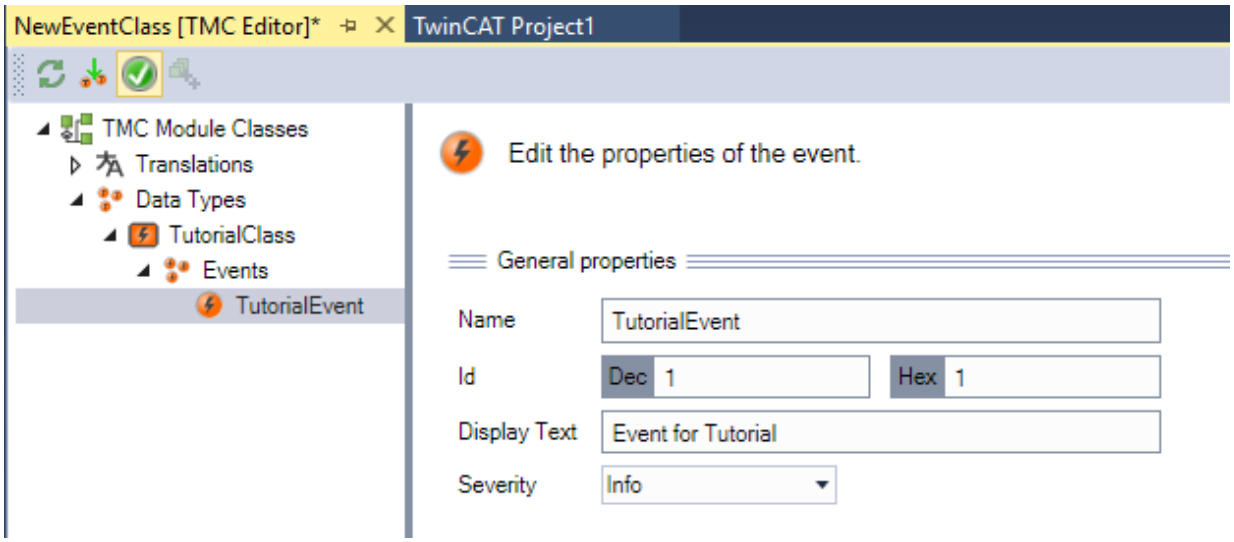


⇨ TMC 编辑器打开。

2. 为事件类命名，然后输入显示文本。



3. 已在事件类下创建了一个事件。为事件命名，然后输入显示文本和严重级别。



4. 保存并关闭事件类（如果适用）。

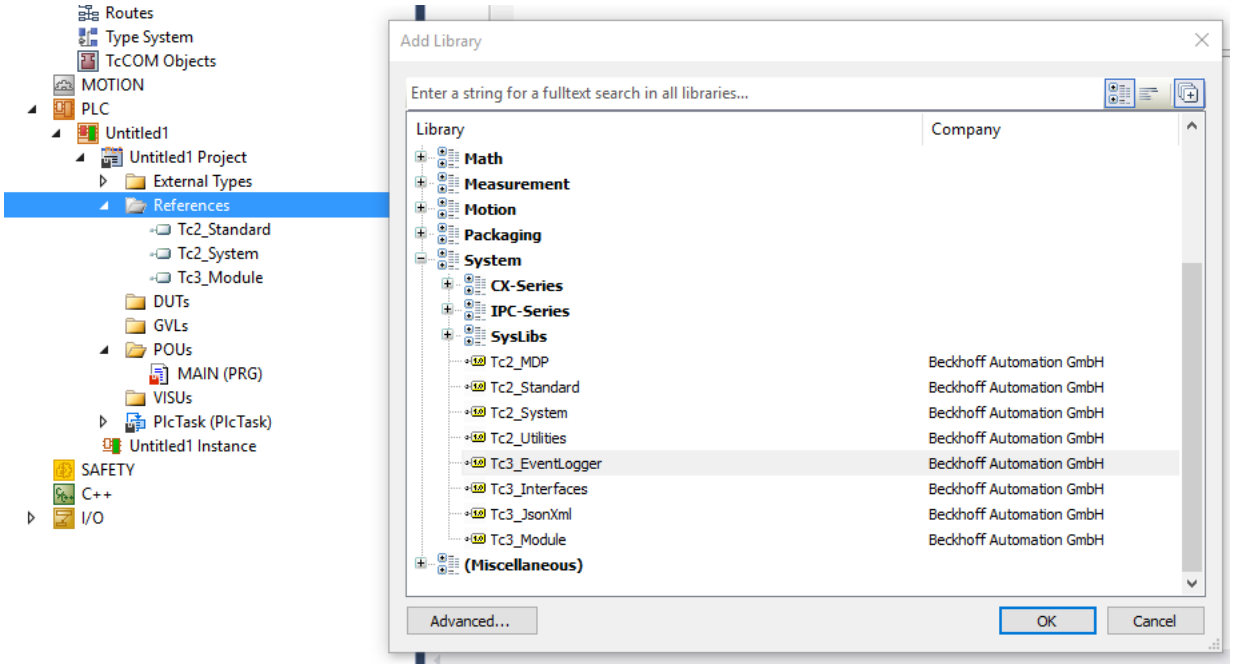
   ⇨ 源代码在 PLC 中提供，可通过 TC_EVENTS 符号访问。

添加 TC3_EventLogger 库

5. 在 **References（引用）** 对象的上下文菜单中选择 **Add Library（添加库）** 命令。

   ⇨ **Add Library（添加库）** 对话框打开。

6. 选择库并确认对话框。



⇨ 该库已添加到 PLC 项目。

创建 PLC 程序

1. 通过双击在编辑器中打开 PLC 项目的 MAIN 程序。
2. 声明并初始化变量 bInit 和 bSend，并声明函数块 FB_TcMessage 的实例：

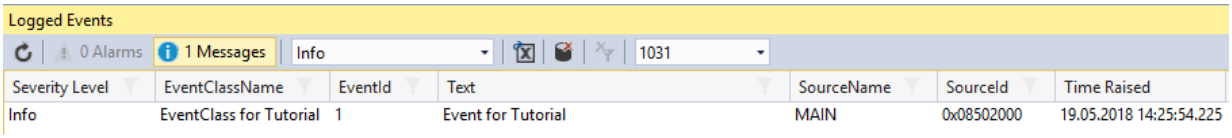
```
PROGRAM MAIN
VAR
        bInit  :  BOOL  :=  TRUE;
        bSend  :  BOOL  :=  TRUE;

        fbMsg  :  FB_TcMessage;
END_VAR
```

3. 在代码中执行所示发送过程。该消息通过 CreateEx 方法初始化一次。由于初始化需要动态资源，因此不应循环进行。随后使用 Send 方法发送初始化的消息。

```
IF  bInit  THEN
        bInit  :=  FALSE;
        fbMsg.CreateEx(TC_EVENTS.TutorialClass.TutorialEvent,  0);
END_IF

IF  bSend  THEN
        bSend  :=  FALSE;
        fbMsg.Send(0);
END_IF
```

4. 创建 PLC 项目并启动 PLC。

⇨ 结果显示在 TwinCAT 3 Engineering 中的 LoggedEvents 窗口中。



9.1.2 示例：ResultMessage

该示例说明如何使用带函数块的 TwinCAT 3 EventLogger。首先，它演示了如何利用函数块上的输出来将事件信息用作扩展返回。其次，它演示了如何进行参数化，以便在某些情况下仅通过 TwinCAT 3 EventLogger 输出消息。

下载：https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/5288319115.zip

该示例包含两个函数块：

- FB_MathCalculation：此函数块提供了两种方法和两个属性，它们始终在输出 ipResultMessage 上输出消息，并在超过跟踪级别时通过 EventLogger 发送消息。
  - Addition() 方法：将两个数相加，并在发生溢出的情况下发送消息
  - Divison() 方法：检查后将两个数相除。在被 0 除时发送消息。
  - 属性 bTraceLevelDefault：指示是在函数块上本地察看跟踪级别，还是使用存在于示例中的 GVL 中的跟踪级别库。
  - 属性 eTraceLevel：如果严重级别大于或等于此属性，则这些方法仅通过 EventLogger 发送消息。
- FB_Control：此函数块显示如何在另一个函数块中使用 FB_MathCalculation 函数块。因此，FB_Control 的 Execute 方法使用 FB_MathCalculation.Divison() 并将消息本身作为错误代码进行进一步处理。

9.1.3 示例：监听器

此示例说明如何使用与消息和警报相关的 TwinCAT 3 EventLogger。同时，消息的接收显示在第二个项目中。

下载：https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/18014403797798923.zip

发布服务器项目

单一 BOOL 变量在发布服务器项目中用作触发器：

- bSendMessage 用于发送消息。
- bRaiseAlarm 用于设置警报。
- bClearAlarm 用于取消警报。
- bConfirmAlarm 用于确认警报。

另外，还有一个选项用于设置 JSON 属性，以便与两条消息一起发送。

监听器项目

监听器项目包含函数块 FB_Listener，它可扩展包含于 Tc3_EventLogger 中的 FB_ListenerBase 函数块。函数块可实现用于接收消息的函数：

- OnMessageSent：发送消息后，EventLogger 将调用此方法作为回调。此方法对消息数进行计数。
- OnAlarmRaised/OnAlarmCleared/OnAlarmConfirmed：如果警报状态更改，则 EventLogger 将调用此方法作为回调。这些方法计算状态更改的次数。
- 为了启动消息的接收，在此函数块上实现 Execute 方法，还同时获取最后接收到的消息文本。
- 函数块 FB_ListenerTest 使用 FB_Listener。在此过程中，它会注册要接收一次的事件类。

要求

开发环境	目标平台	要包括的 PLC 库
TwinCAT v3.1.4024.0	PC 或 CX (x64、x86、ARM)	Tc3_EventLogger (>= v3.1.21.0)

9.2 C++

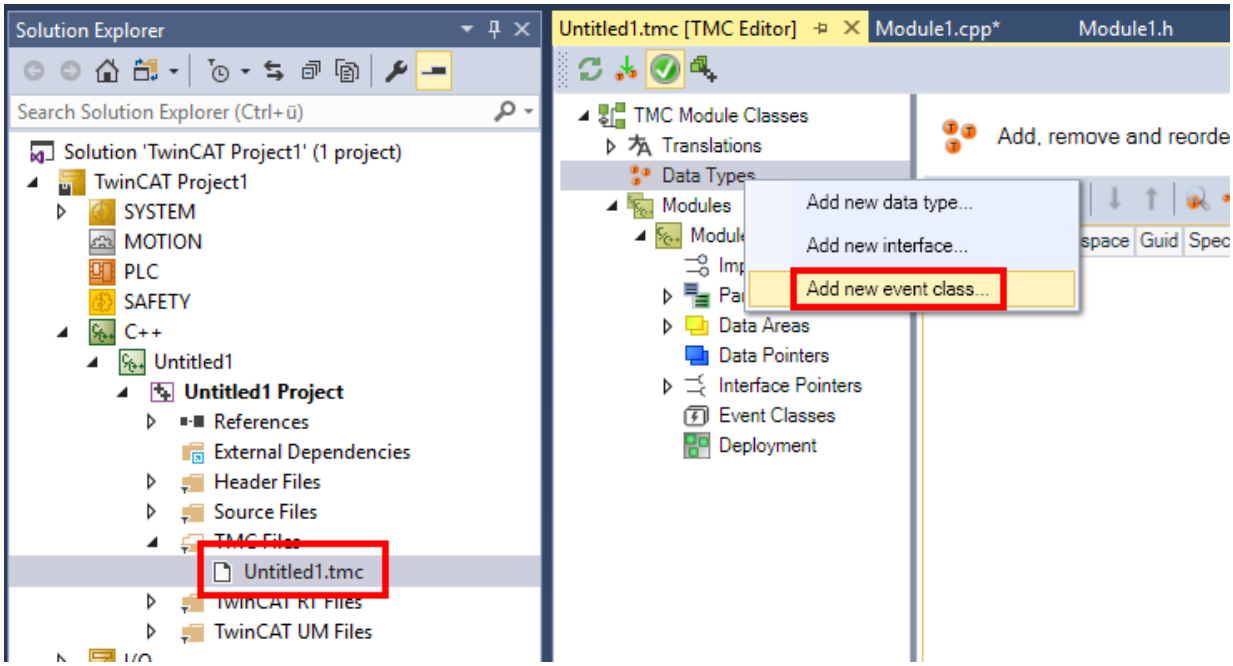
9.2.1 教程

该教程说明从一个空的 TwinCAT 项目到已调度消息的工作步骤。它描述了工作序列中技术简介 [▶ 10]部分所述的 TwinCAT 3 EventLogger 的属性。

在 C++ 项目中，以 TMC 文件的数据类型创建事件类

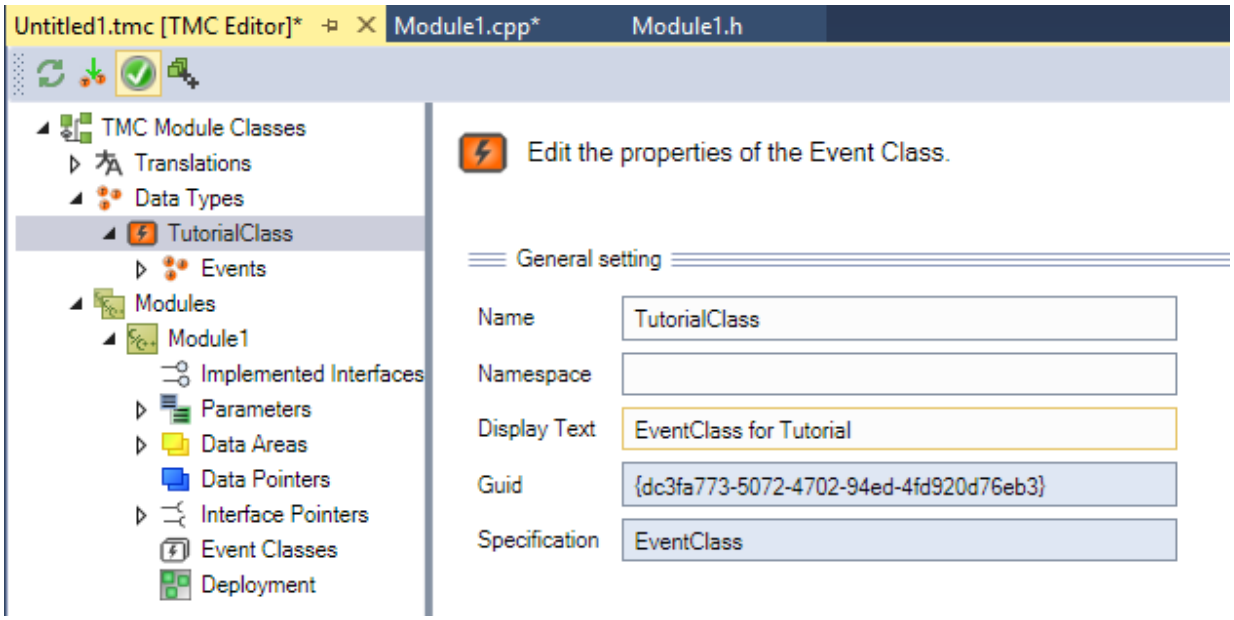
✓ 通过向导“TwinCAT Module Class with Cyclic IO”的模块新建 TwinCAT C++ 项目。

1. 双击 C++ 项目中的 TMC 文件，打开 TMC 编辑器。从 **Data Types（数据类型）** 的上下文菜单中选择命令 **Add new event class…（添加新建事件类...）**。

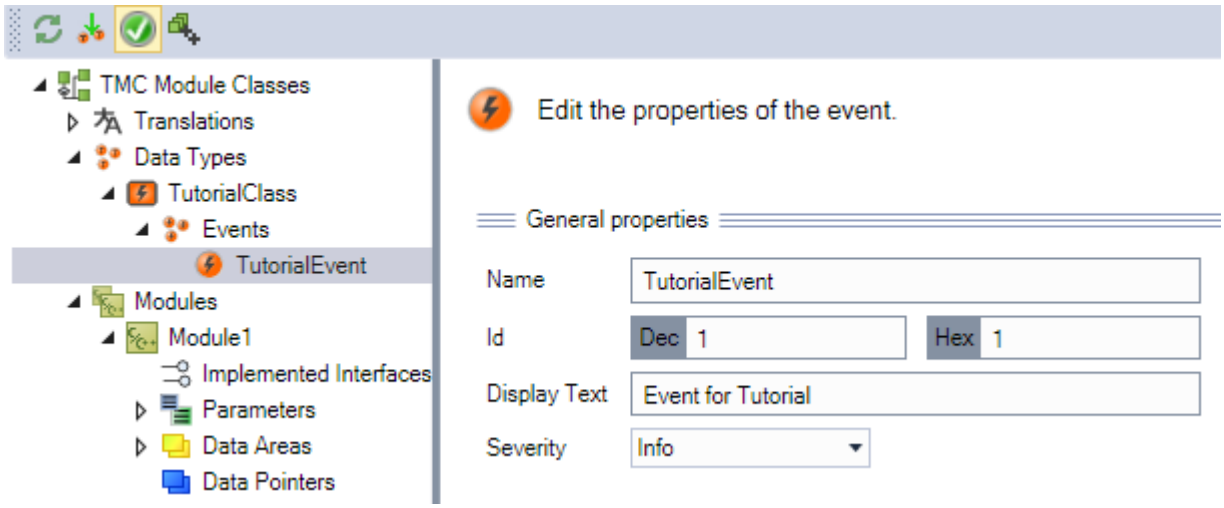


⇨ TMC 编辑器打开事件类。

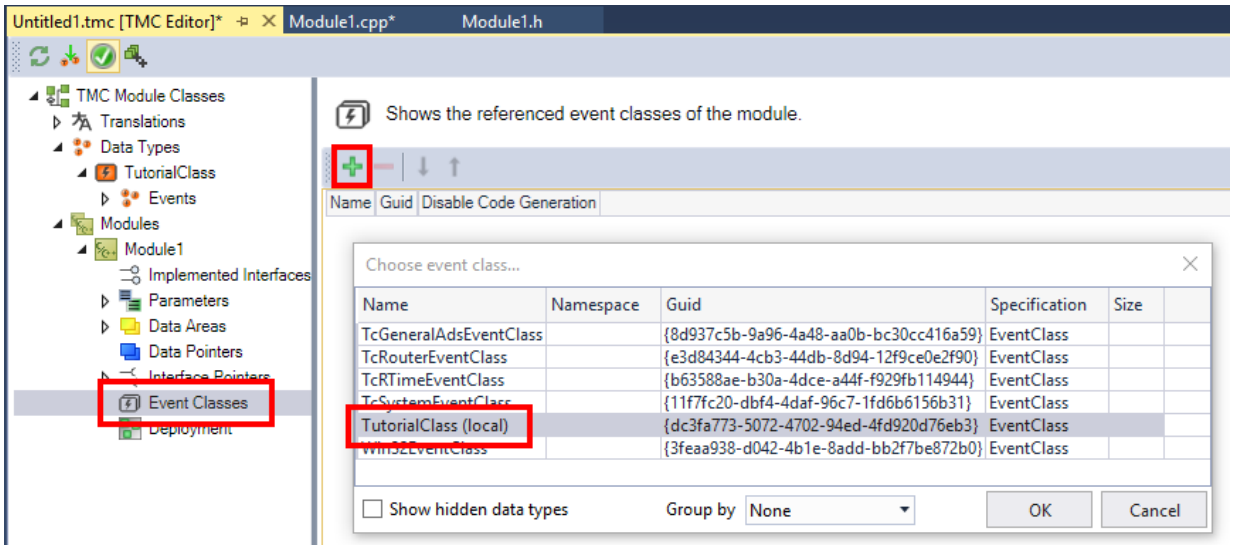
2. 为事件类命名，然后选择输入显示文本。



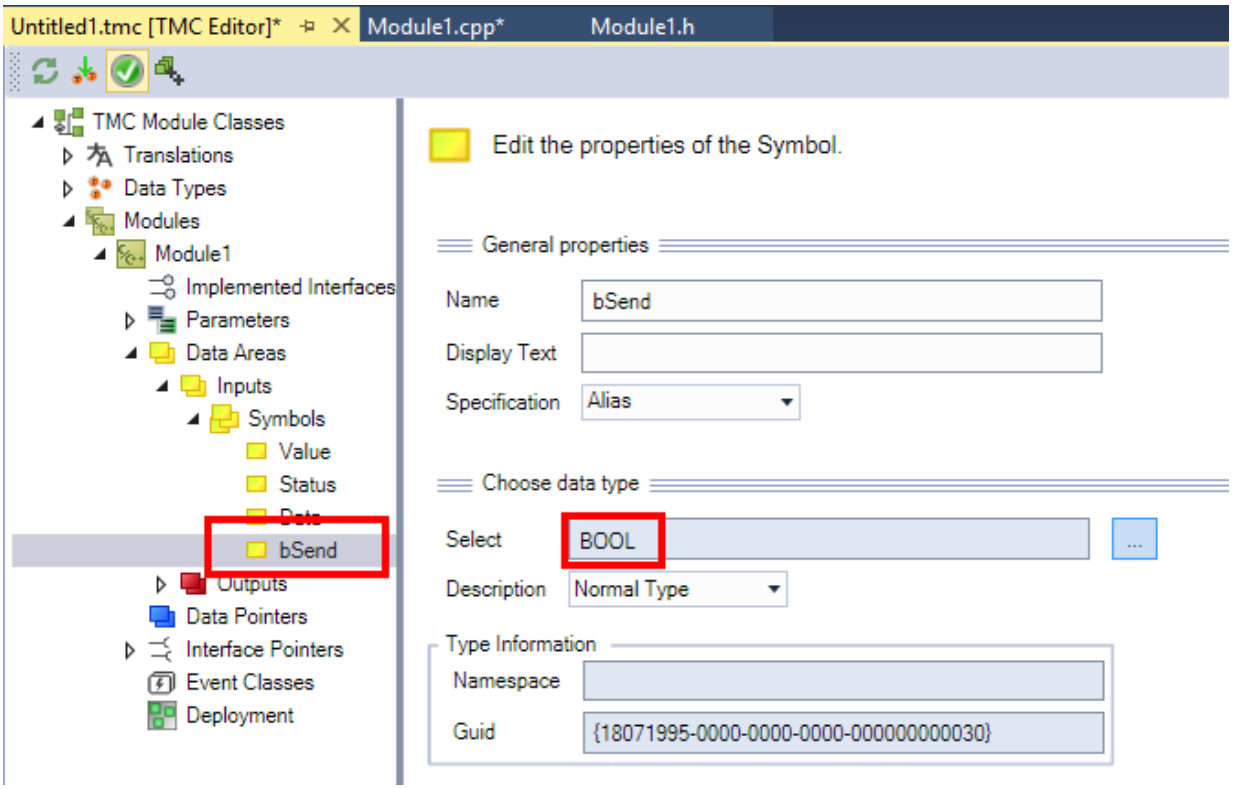
3. 已在事件类下创建了一个事件。为事件命名，然后输入显示文本和严重级别。



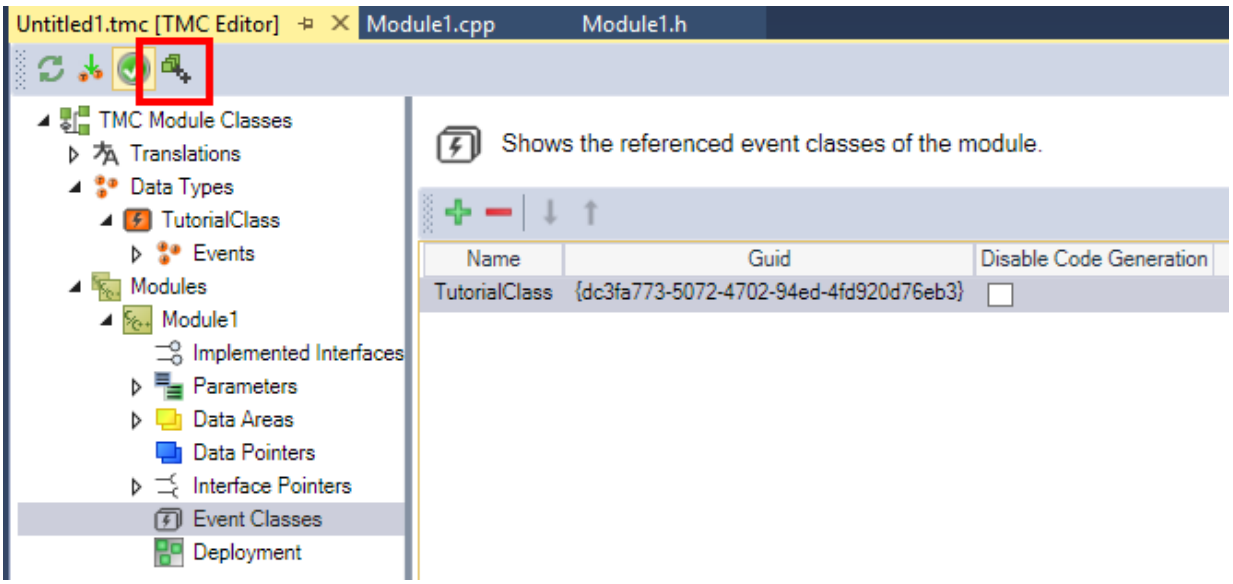
4. 在先前创建的模块中使用事件类。



5. 此外，创建输入 bSend，以便在上升边缘发送事件。



6. 生成模块的源代码。



创建 C++ 程序

7. 将标头添加到 C++ 程序中的 Untitled1Interfaces.h 中：

```
#include "TcRouterInterfaces.h"
#include "TcEventLoggerInterfaces.h"
```

8. 您在 Module1.h 中需要以下声明：

```
UINT m_counter;
ITcEventLoggerPtr m_spEventLogger;
ITcMessagePtr m_spMessage;
BOOL m_OldSend;
```

9. 在 Module1.cpp 中的模块构造函数中初始化以下值：

```
CModule1::CModule1()
        : m_Trace(m_TraceLevelMax, m_spSrv)
, m_counter(0)
{
```

```
///<AutoGeneratedContent id="MemberInitialization">
        m_TraceLevelMax = tlAlways;
        memset(&m_Parameter, 0, sizeof(m_Parameter));
        memset(&m_Inputs, 0, sizeof(m_Inputs));
        memset(&m_Outputs, 0, sizeof(m_Outputs));
///</AutoGeneratedContent>
        m_spEventLogger = 0;
        m_spMessage = 0;
        m_OldSend = FALSE;
        m_Inputs.bSend = TRUE;
}
```

10. 另外，在方法 SetObjStateSO() 中执行以下初始化：

```
// TODO: Add any additional initialization
m_spEventLogger.SetOID(OID_TCEVENTLOGGER);
hr = FAILED(hr) ? hr : m_spSrv->TcQuerySmartObjectInterface(m_spEventLogger);
hr = FAILED(hr) ? hr : m_spEventLogger-
>CreateMessage(TcEvents::TutorialClass::EventClass, TcEvents::TutorialClass::TutorialEvent.nEventId, TcEvents::Tut
orialClass::TutorialEvent.eSeverity, 0, &m_spMessage);
```

11. 如果方法 AddModuleToCaller() 调用失败，则执行反初始化：

```
// Cleanup if transition failed at some stage
if ( FAILED(hr) )
{
        RemoveModuleFromCaller();
        m_spEventLogger = NULL;
        m_spMessage = NULL;
}
```

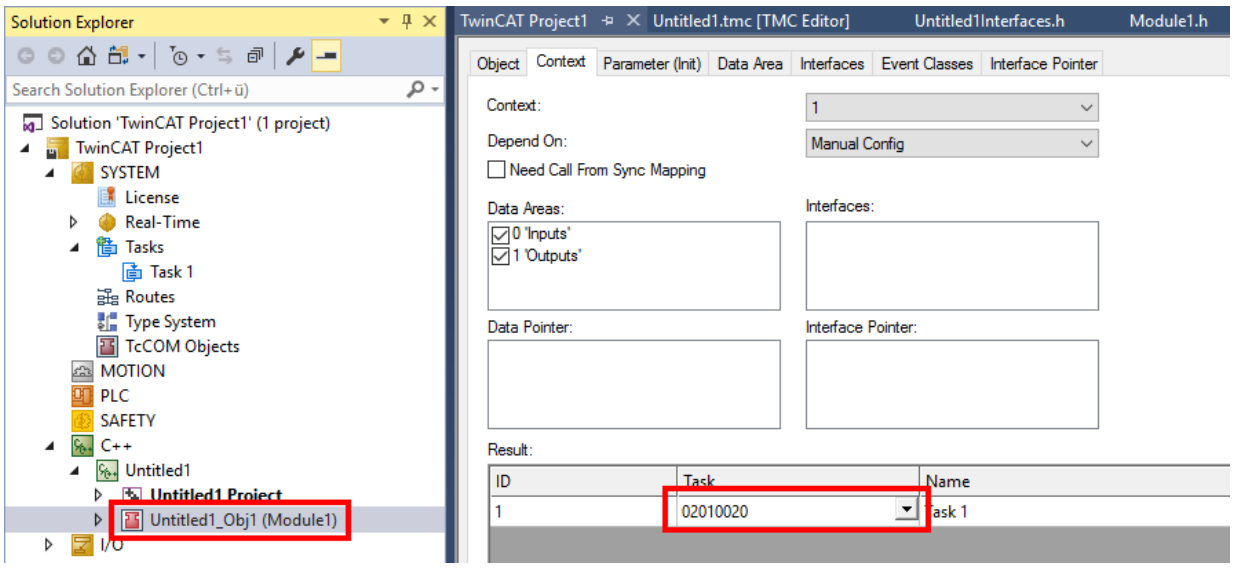
12. 在方法 SetObjStateOS() 中执行初始化：

```
// TODO: Add any additional deinitialization
m_spEventLogger = NULL;
m_spMessage = NULL;
```

13. 通过发送消息来扩展模块的循环代码：

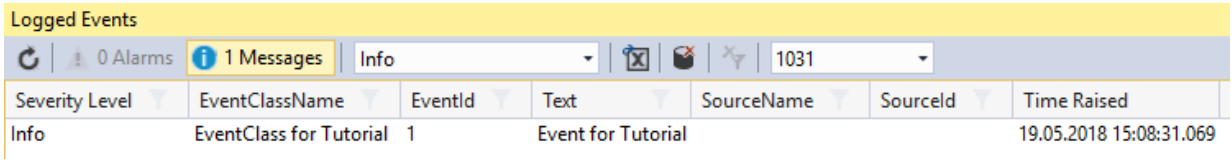
```
// TODO: Replace the sample with your cyclic code
if (m_Inputs.bSend && ! m_OldSend) // raising edge
{
     m_spMessage->Send(0);
}
m_OldSend = m_Inputs.bSend;
```

14. 创建一个模块实例并将其与任务链接。



15. 创建 C++ 项目并启动 TwinCAT。

⇨ 结果显示在 TwinCAT 3 Engineering 中的 LoggedEvents 窗口中。



9.2.2 示例：启动-停止

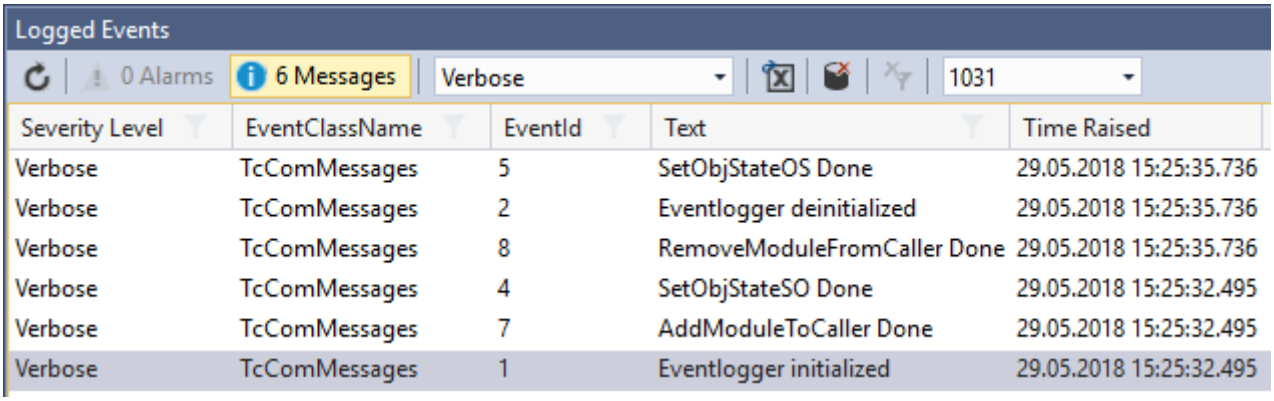
该示例说明在启动和停止 TwinCAT 时如何使用 TwinCAT 3 EventLogger。消息与映射 C++ 模块不同状态的参数一起使用。

下载：https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/5288321291.zip

该消息在状态机的方法中使用：

- SetObjStatePS()：TwinCAT 正在启动 EventLogger。因此，无法在这种状态下使用。
- SetObjStateSO()：此处，对 EventLogger 的引用通过 TcQuerySmartObjectInterface 和 CreateMessage() 启动的消息获得。调度相应的消息。使用 AddModuleToCaller() 调度消息。在转换结束时，还会在此处调度一条消息。
- CycleUpdate()：在此示例中，未调度任何消息。因此，模块对 OP 状态的行为等同于“Cyclic IO”模块的行为。
- SetObjStateOS()：使用 RemoveModuleFromCaller() 调度消息。在转换结束时，还会在此处调度一条消息。此后，对消息和 EventLogger 的引用将设置为零，并通过消息进行通知。
- SetObjStateSP()：TwinCAT 关闭了 EventLogger。因此，无法在这种状态下使用。

生成以下消息：



9.2.3 示例：监听器

此示例说明如何使用与消息和警报相关的 TwinCAT 3 EventLogger。

它由可以发送消息和警报的模块以及接收这些消息的模块组成。

下载：https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/9007204543064459.zip

发布服务器模块

单一 BOOL 变量在发布服务器模块中用作输入数据区域的触发器：

- bSendMessage 用于发送消息
- bRaiseAlarm 用于设置警报
- bClearAlarm 用于取消警报
- bConfirmAlarm 用于确认警报

另外，还有一个选项用于设置和移除 JSON 属性，以便与两条消息一起发送。

监听器模块

监听器模块同时实现 ITCMessageListener 和 ITcAlarmListener 接口。因此，EventLogger 调用指定方法作为回调。

- OnMessageSent：发送消息后，EventLogger 将调用此方法作为回调。此方法对消息数进行计数。

- OnAlarmRaised/OnAlarmCleared/OnAlarmConfirmed：如果警报状态更改，则 EventLogger 将调用此方法作为回调。这些方法计算状态更改的次数。

因此，监听器模块在方法 SetObjStateSO() 中启动时，就通过 EventLogger 针对相应的事件类进行注册。

9.3 COM

您可以使用 COM 接口开发自己的程序，以对事件做出反应。

由于这是一个 COM 接口，因此可以在 Windows 下通过不同的编程语言进行使用。

- https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/7726430987.zip
- https://infosys.beckhoff.com/content/1033/tc3_eventlogger/Resources/zip/7726428683.zip

每个示例都显示了如何接收 EventLogger 事件。在该示例中，输出在控制台上执行。

有关功能的精确描述，可以在 COM API [▶ 99] 中找到。

10 附录

10.1 ADS 返回代码

错误代码分组：0x000 [▶ 148]...、0x500 [▶ 148]...、0x700 [▶ 149]...、0x1000 [▶ 151]...

全局错误代码

Hex	Dec	HRESULT	名称	描述
0x0	0	0x9811 0000	ERR_NOERROR	无错误。
0x1	1	0x9811 0001	ERR_INTERNAL	内部错误。
0x2	2	0x9811 0002	ERR_NORTIME	不具有实时性。
0x3	3	0x9811 0003	ERR_ALLOCLOCKEDMEM	分配已锁定 - 内存错误。
0x4	4	0x9811 0004	ERR_INSERTMAILBOX	邮箱已满 - 无法发送 ADS 消息。减少每个周期的 ADS 消息数量将有所帮助。
0x5	5	0x9811 0005	ERR_WRONGRECEIVEHMSG	HMSG 错误。
0x6	6	0x9811 0006	ERR_TARGETPORTNOTFOUND	未找到目标端口 - ADS 服务器未启动或无法访问。
0x7	7	0x9811 0007	ERR_TARGETMACHINENOTFOUND	未找到目标计算机 - 未找到 AMS 路由。
0x8	8	0x9811 0008	ERR_UNKNOWNCMDID	未知命令 ID。
0x9	9	0x9811 0009	ERR_BADTASKID	任务 ID 无效。
0xA	10	0x9811 000A	ERR_NOIO	无 IO。
0xB	11	0x9811 000B	ERR_UNKNOWNAMSCMD	未知 AMS 命令。
0xC	12	0x9811 000C	ERR_WIN32ERROR	Win32 错误。
0xD	13	0x9811 000D	ERR_PORTNOTCONNECTED	端口未连接。
0xE	14	0x9811 000E	ERR_INVALIDAMSLENGTH	AMS 长度无效。
0xF	15	0x9811 000F	ERR_INVALIDAMSNETID	AMS Net ID 无效。
0x10	16	0x9811 0010	ERR_LOWINSTLEVEL	安装级别低 - TwinCAT 2 授权错误。
0x11	17	0x9811 0011	ERR_NODEBUGINTAVAILABLE	无调试可用。
0x12	18	0x9811 0012	ERR_PORTDISABLED	端口已禁用 - TwinCAT 系统服务未启动。
0x13	19	0x9811 0013	ERR_PORTALREADYCONNECTED	端口已连接。
0x14	20	0x9811 0014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 错误。
0x15	21	0x9811 0015	ERR_AMSSYNC_TIMEOUT	AMS Sync 超时。
0x16	22	0x9811 0016	ERR_AMSSYNC_AMSERROR	AMS Sync 错误。
0x17	23	0x9811 0017	ERR_AMSSYNC_NOINDEXINMAP	没有适用于 AMS Sync 的索引图。
0x18	24	0x9811 0018	ERR_INVALIDAMSPORT	AMS 端口无效。
0x19	25	0x9811 0019	ERR_NOMEMORY	无内存。
0x1A	26	0x9811 001A	ERR_TCPSEND	TCP 发送错误。
0x1B	27	0x9811 001B	ERR_HOSTUNREACHABLE	主机无法访问。
0x1C	28	0x9811 001C	ERR_INVALIDAMSFRAGMENT	AMS 片段无效。
0x1D	29	0x9811 001D	ERR_TLSSEND	TLS 发送错误 - ADS 安全连接失败。
0x1E	30	0x9811 001E	ERR_ACCESSDENIED	拒绝访问 - 拒绝 ADS 安全访问。

路由器错误代码

Hex	Dec	HRESULT	名称	描述
0x500	1280	0x9811 0500	ROUTERERR_NOLOCKEDMEMORY	无法分配锁定的内存。
0x501	1281	0x9811 0501	ROUTERERR_RESIZEMEMORY	路由器内存大小无法更改。
0x502	1282	0x9811 0502	ROUTERERR_MAILBOXFULL	邮箱已达到最大消息数。
0x503	1283	0x9811 0503	ROUTERERR_DEBUGBOXFULL	调试邮箱已达到最大消息数。
0x504	1284	0x9811 0504	ROUTERERR_UNKNOWNPORTTYPE	端口类型未知。
0x505	1285	0x9811 0505	ROUTERERR_NOTINITIALIZED	路由器未初始化。
0x506	1286	0x9811 0506	ROUTERERR_PORTALREADYINUSE	端口号已分配。
0x507	1287	0x9811 0507	ROUTERERR_NOTREGISTERED	端口未注册。
0x508	1288	0x9811 0508	ROUTERERR_NOMOREQUEUES	已达到最大端口数。
0x509	1289	0x9811 0509	ROUTERERR_INVALIDPORT	端口无效。
0x50A	1290	0x9811 050A	ROUTERERR_NOTACTIVATED	路由器未激活。
0x50B	1291	0x9811 050B	ROUTERERR_FRAGMENTBOXFULL	邮箱已达到最大片段消息数。
0x50C	1292	0x9811 050C	ROUTERERR_FRAGMENTTIMEOUT	发生片段超时。
0x50D	1293	0x9811 050D	ROUTERERR_TOBEREMOVED	端口已移除。

一般性 ADS 错误代码

Hex	Dec	HRESULT	名称	描述
0x700	1792	0x9811 0700	ADSERR_DEVICE_ERROR	一般性设备错误。
0x701	1793	0x9811 0701	ADSERR_DEVICE_SRVNOTSUPP	服务器不支持该服务。
0x702	1794	0x9811 0702	ADSERR_DEVICE_INVALIDGRP	索引组无效。
0x703	1795	0x9811 0703	ADSERR_DEVICE_INVALIDOFFSET	索引偏移量无效。
0x704	1796	0x9811 0704	ADSERR_DEVICE_INVALIDACCESS	不允许读取或写入。
0x705	1797	0x9811 0705	ADSERR_DEVICE_INVALIDSIZE	参数大小不正确。
0x706	1798	0x9811 0706	ADSERR_DEVICE_INVALIDDATA	参数值无效。
0x707	1799	0x9811 0707	ADSERR_DEVICE_NOTREADY	设备尚未准备好运行。
0x708	1800	0x9811 0708	ADSERR_DEVICE_BUSY	设备正忙。
0x709	1801	0x9811 0709	ADSERR_DEVICE_INVALIDCONTEXT	操作系统上下文无效。这可能是由于在不同的任务中使用 ADS 函数块造成的。可以通过在 PLC 中进行多任务同步解决这个问题。
0x70A	1802	0x9811 070A	ADSERR_DEVICE_NOMEMORY	内存不足。
0x70B	1803	0x9811 070B	ADSERR_DEVICE_INVALIDPARM	参数值无效。
0x70C	1804	0x9811 070C	ADSERR_DEVICE_NOTFOUND	未找到（文件，...）。
0x70D	1805	0x9811 070D	ADSERR_DEVICE_SYNTAX	文件或命令中的语法错误。
0x70E	1806	0x9811 070E	ADSERR_DEVICE_INCOMPATIBLE	对象不匹配。
0x70F	1807	0x9811 070F	ADSERR_DEVICE_EXISTS	对象已存在。
0x710	1808	0x9811 0710	ADSERR_DEVICE_SYMBOLNOTFOUND	符号未找到。
0x711	1809	0x9811 0711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	符号版本无效。这可能是由于在线更改造成的。创建新句柄。
0x712	1810	0x9811 0712	ADSERR_DEVICE_INVALIDSTATE	设备处于无效状态。
0x713	1811	0x9811 0713	ADSERR_DEVICE_TRANSMODENOTSUPP	不支持 AdsTransMode。
0x714	1812	0x9811 0714	ADSERR_DEVICE_NOTIFYHNDINVALID	通知句柄无效。
0x715	1813	0x9811 0715	ADSERR_DEVICE_CLIENTUNKNOWN	通知客户端未注册。
0x716	1814	0x9811 0716	ADSERR_DEVICE_NOMOREHDLS	无进一步的通知句柄。
0x717	1815	0x9811 0717	ADSERR_DEVICE_INVALIDWATCHSIZE	通知大小过大。
0x718	1816	0x9811 0718	ADSERR_DEVICE_NOTINIT	设备未初始化。
0x719	1817	0x9811 0719	ADSERR_DEVICE_TIMEOUT	设备超时。
0x71A	1818	0x9811 071A	ADSERR_DEVICE_NOINTERFACE	接口查询失败。
0x71B	1819	0x9811 071B	ADSERR_DEVICE_INVALIDINTERFACE	请求的接口错误。
0x71C	1820	0x9811 071C	ADSERR_DEVICE_INVALIDCLSID	类 ID 无效。
0x71D	1821	0x9811 071D	ADSERR_DEVICE_INVALIDOBJID	对象 ID 无效。
0x71E	1822	0x9811 071E	ADSERR_DEVICE_PENDING	请求挂起。
0x71F	1823	0x9811 071F	ADSERR_DEVICE_ABORTED	请求已中止。
0x720	1824	0x9811 0720	ADSERR_DEVICE_WARNING	信号警告。
0x721	1825	0x9811 0721	ADSERR_DEVICE_INVALIDARRAYIDX	数组索引无效。
0x722	1826	0x9811 0722	ADSERR_DEVICE_SYMBOLNOTACTIVE	符号未激活。
0x723	1827	0x9811 0723	ADSERR_DEVICE_ACCESSDENIED	拒绝访问。
0x724	1828	0x9811 0724	ADSERR_DEVICE_LICENSENOTFOUND	缺少授权。
0x725	1829	0x9811 0725	ADSERR_DEVICE_LICENSEEXPIRED	授权已过期。
0x726	1830	0x9811 0726	ADSERR_DEVICE_LICENSEEXCEEDED	超出授权。
0x727	1831	0x9811 0727	ADSERR_DEVICE_LICENSEINVALID	授权无效。
0x728	1832	0x9811 0728	ADSERR_DEVICE_LICENSESYSTEMID	系统 ID 授权无效。
0x729	1833	0x9811 0729	ADSERR_DEVICE_LICENSENOTIMELIMIT	授权不受时间限制。
0x72A	1834	0x9811 072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	授权问题：未来的时间。
0x72B	1835	0x9811 072B	ADSERR_DEVICE_LICENSETIMETOLONG	授权期限太长。
0x72C	1836	0x9811 072C	ADSERR_DEVICE_EXCEPTION	系统启动异常。
0x72D	1837	0x9811 072D	ADSERR_DEVICE_LICENSEDUPLICATED	授权文件读取了两次。
0x72E	1838	0x9811 072E	ADSERR_DEVICE_SIGNATUREINVALID	签名无效。
0x72F	1839	0x9811 072F	ADSERR_DEVICE_CERTIFICATEINVALID	证书无效。
0x730	1840	0x9811 0730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	未从 OEM 获知公开密钥。
0x731	1841	0x9811 0731	ADSERR_DEVICE_LICENSERESTRICTED	此系统 ID 授权无效。
0x732	1842	0x9811 0732	ADSERR_DEVICE_LICENSEDEMODENIED	演示授权已禁止。
0x733	1843	0x9811 0733	ADSERR_DEVICE_INVALIDFNCID	无效函数 ID。
0x734	1844	0x9811 0734	ADSERR_DEVICE_OUTOFRANGE	超出有效范围。
0x735	1845	0x9811 0735	ADSERR_DEVICE_INVALIDALIGNMENT	无效对齐。
0x736	1846	0x9811 0736	ADSERR_DEVICE_LICENSEPLATFORM	无效平台级别。

Hex	Dec	HRESULT	名称	描述
0x737	1847	0x9811 0737	ADSERR_DEVICE_FORWARD_PL	上下文 - 转到被动级别。
0x738	1848	0x9811 0738	ADSERR_DEVICE_FORWARD_DL	上下文 - 转到调度级别。
0x739	1849	0x9811 0739	ADSERR_DEVICE_FORWARD_RT	上下文 - 转到实时。
0x740	1856	0x9811 0740	ADSERR_CLIENT_ERROR	客户端错误。
0x741	1857	0x9811 0741	ADSERR_CLIENT_INVALIDPARM	服务包含无效参数。
0x742	1858	0x9811 0742	ADSERR_CLIENT_LISTEMPTY	轮询列表为空。
0x743	1859	0x9811 0743	ADSERR_CLIENT_VARUSED	Var 连接已在使用中。
0x744	1860	0x9811 0744	ADSERR_CLIENT_DUPLINVOKEID	调用的 ID 已在使用中。
0x745	1861	0x9811 0745	ADSERR_CLIENT_SYNCTIMEOUT	已发生超时 - 远程终端在指定的 ADS 超时中未响应。远程终端的路由设置可能配置不正确。
0x746	1862	0x9811 0746	ADSERR_CLIENT_W32ERROR	Win32 子系统中发生错误。
0x747	1863	0x9811 0747	ADSERR_CLIENT_TIMEOUTINVALID	客户端超时值无效。
0x748	1864	0x9811 0748	ADSERR_CLIENT_PORTNOTOPEN	端口未打开。
0x749	1865	0x9811 0749	ADSERR_CLIENT_NOAMSADDR	无 AMS 地址。
0x750	1872	0x9811 0750	ADSERR_CLIENT_SYNCINTERNAL	Ads sync 中发生内部错误。
0x751	1873	0x9811 0751	ADSERR_CLIENT_ADDHASH	哈希表溢出。
0x752	1874	0x9811 0752	ADSERR_CLIENT_REMOVEHASH	未在表格中找到密钥。
0x753	1875	0x9811 0753	ADSERR_CLIENT_NOMORESYM	缓存中没有符号。
0x754	1876	0x9811 0754	ADSERR_CLIENT_SYNCRESINVALID	收到无效响应。
0x755	1877	0x9811 0755	ADSERR_CLIENT_SYNCPORTLOCKED	同步端口已锁定。

RTime 错误代码

Hex	Dec	HRESULT	名称	描述
0x1000	4096	0x9811 1000	RTERR_INTERNAL	实时系统中发生内部错误。
0x1001	4097	0x9811 1001	RTERR_BADTIMERPERIODS	计时器值无效。
0x1002	4098	0x9811 1002	RTERR_INVALIDTASKPTR	任务指向无效值 0（零）。
0x1003	4099	0x9811 1003	RTERR_INVALIDSTACKPTR	任务堆栈指向无效值 0。
0x1004	4100	0x9811 1004	RTERR_PRIOEXISTS	请求任务优先级已分配。
0x1005	4101	0x9811 1005	RTERR_NOMORETCB	没有可用的空闲 TCB（任务控制块）。TCB 的最大数量为 64。
0x1006	4102	0x9811 1006	RTERR_NOMORESEMAS	没有可用的空闲信号。信号的最大数量为 64。
0x1007	4103	0x9811 1007	RTERR_NOMOREQUEUES	队列中没有可用的空闲空间。队列中的最大位置数为 64。
0x100D	4109	0x9811 100D	RTERR_EXTIRQALREADYDEF	已应用外部同步中断。
0x100E	4110	0x9811 100E	RTERR_EXTIRQNOTDEF	未应用外部同步中断。
0x100F	4111	0x9811 100F	RTERR_EXTIRQINSTALLFAILED	外部同步中断应用失败
0x1010	4112	0x9811 1010	RTERR_IRQLNOTLESSOREQUAL	在错误的上下文中调用服务函数
0x1017	4119	0x9811 1017	RTERR_VMXNOTSUPPORTED	不支持 Intel VT-x 扩展。
0x1018	4120	0x9811 1018	RTERR_VMXDISABLED	BIOS 中未启用 Intel VT-x 扩展。
0x1019	4121	0x9811 1019	RTERR_VMXCONTROLSMISSING	Intel VT-x 扩展中缺少函数。
0x101A	4122	0x9811 101A	RTERR_VMXENABLEFAILS	Intel VT-x 激活失败。

TCP Winsock 错误代码

Hex	Dec	名称	描述
0x274C	10060	WSAETIMEDOUT	发生连接超时。 连接错误，因为远程终端在特定时间后未正确响应，或者所连接的主机未响应，因此无法维持连接。
0x274D	10061	WSAECONNREFUSED	拒绝连接。 无法建立连接，因为目标计算机明确拒绝了该连接。此错误通常由尝试连接到外部主机上处于非活动状态的服务（即没有运行服务器应用程序的服务）导致。
0x2751	10065	WSAEHOSTUNREACH	没有到主机的路由 套接字操作引用了不可用的主机。
更多 Winsock 错误代码：Win32 错误代码			

10.2 支持和服务

倍福公司及其合作伙伴在世界各地提供全面的支持与服务，对与倍福产品和系统解决方案相关的所有问题提供快速有效的帮助。

倍福分公司和代表处

有关倍福产品的当地支持和服务方面的信息，请联系倍福分公司或代表处！

可以在网页：
http://www.beckhoff.com 中找到世界各地的倍福分公司和代表处的地址

您还可以在该网页找到更多倍福组件的文档。

倍福公司总部

德国倍福自动化有限公司

Huelshorstweg 20
33415 Verl
Germany

电话：	+49(0)5246/963-0
传真：	+49(0)5246/963-198
电子邮箱：	info@beckhoff.com

倍福支持

支持服务为您提供全面的技术援助，不仅帮助您应用各种倍福产品，还提供其他广泛的服务：

- 提供支持
- 设计、编程并调试复杂自动化系统
- 以及倍福系统组件广泛的培训计划

热线电话：	+49(0)5246/963-157
传真：	+49(0)5246/963-9157
电子邮箱：	support@beckhoff.com

倍福服务

倍福服务中心为您提供一切售后服务：

- 现场服务
- 维修服务
- 备件服务
- 热线服务

热线电话：	+49(0)5246/963-460
传真：	+49(0)5246/963-479
电子邮箱：	service@beckhoff.com

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
+49 5246 9630
info@beckhoff.com
www.beckhoff.com