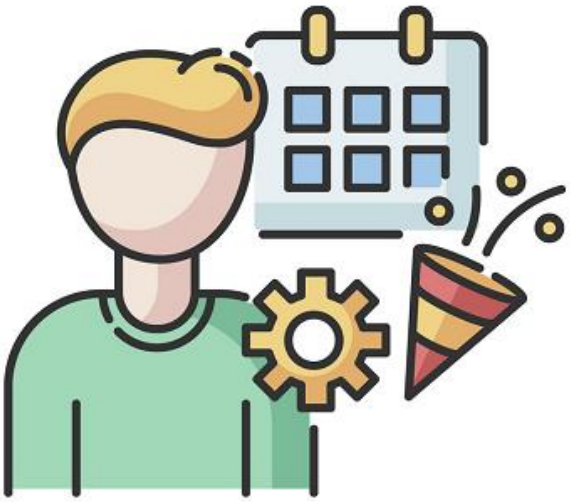
Использование событий в CODESYS V3



06.08.2023
версия 1.0

Оглавление

Введение

Система исполнения CODESYS (часто называемая также «рантайм») имеет модульную архитектуру и состоит из отдельных компонентов. В процессе работы эти компоненты должны получать информацию о работе друг друга. Для этого в рантайме реализован механизм событий, функционирование которого обеспечивается компонентом **CmpEventMgr**. Примерами событий рантайма являются:

- запуск, остановка и сброс пользовательского приложения (а также факт загрузки нового приложения, онлайн-изменения существующего приложения, попадание в точку остановка и т. п.);
- подключение/отключение клиента web-визуализации;
- возникновение исключение в процессе работы приложения;
- и т. д. (см. полный список в приложении А).

События позволяют разработчику компонента предусмотреть реакцию на изменение состояния других компонентов – например, очистить ресурсы (закрыть файлы, сокеты, COM-порты, освободить динамически выделенную память и т. д.) в случае сброса или удаления приложения.

В этом документе описан принцип реализации событий в среде CODESYS V3.5 и способы работы с ними.

Автор: Евгений Кислов

1. Основная информация о событиях

Обработкой событий занимается компонент **CmpEventMgr**.

С точки зрения работы с событиями существует две роли – отправитель (provider) и получатель (consumer). Компонент может совмещать эти роли, являясь одновременно и отправителем, и получателем событий. Отправка событий является «безадресной» – то есть компонент-отправитель обозначает факт того, что событие произошло, а компоненты-получатели, подписавшиеся на данное событие, уведомляются об этом и могут отреагировать на это в своем коде.

Отправителем и получателем может быть любой компонент, а также пользовательское приложение.

Отправитель может:

- создать (create) событие – обозначить потенциальную возможность наступления данного события;
- отправить (post) событие – обозначить факт того, что событие произошло;
- удалить (delete) событие – обозначить потерю возможности наступления данного события.

Получатель может:

- открыть (open) событие – проверить факт создания данного события;
- зарегистрировать callback (register) для события – указать, какой программный объект должен быть вызван при наступлении события;
- дерегистрировать callback (unregister) для события – указать, что обработка данного события больше не требуется;
- закрыть (close) событие – очистить ресурсы, выделяемые рантаймом для обработки события. Обычно выполняется после дерегистрации callback'а.

Таким образом, обработка событий на стороне получателя реализована по технологии callback. Получатель проверяет существование интересующих его событий (они существуют, если были созданы – (create) – компонентами-отправителями) и, если они существуют, регистрирует для каждого события callback-объект, который будет однократно вызван системой исполнения при каждой отправке события. Callback-объектом может быть:

- функция (реализованная на языках МЭК или ANSI C, если компонент реализуется на этом языке);
- метод функционального блока;
- метод класса (реализованный на C++; только для C++ версии рантайма).

В системе исполнения событие описывается структурой **EventParam**:

```
typedef struct
{
 EVENTID EventId;
 CMPID CmpIdProvider;
 unsigned short usParamId;
 unsigned short usVersion;
 void *pParameter;
 void *pUserParameter;
} EventParam;
```

EventId – это идентификатор события. Он представляет собой значение типа DWORD, в котором старшее слово – это код класса события, а младшее – код события. Классы позволяют разделять события по «критичности». В рантайме определены следующие классы событий:

```
EVTCLASS_NONE (класс отсутствует или задан некорректно)
EVTCLASS_ALL (специальный класс, используется только для фильтрации; событие не может
иметь такой класс)
EVTCLASS_INFO
EVTCLASS_WARNING
EVTCLASS_ERROR
EVTCLASS_EXCEPTION
EVTCLASS_INTERRUPT (события о прерываниях)
EVTCLASS_VENDOR_SPEC (начиная с этого кода производители контроллеров могут создавать
собственные классы событий)
```

Код события является уникальным в пределах компонента, но в разных компонентах могут существовать (и существуют) события с одинаковыми кодами.

Поэтому однозначно идентифицировать событие можно только по связке полей **EventId** и **CmpIdProvider**, где **CmpIdProvider** – уникальный идентификатор компонента в системе исполнения.

При получении события компоненту-получателя нужен его контекст. Например, произошло исключение. Какое именно? Или произошел сброс приложения – «горячий», «холодный» или заводской? Для предоставления контекста в состав структуры события входит так называемый «параметр события». Обычно параметр события представляет собой структуру.

- **usParamId** – это идентификатор параметра.
- **usVersion** – версия параметра (по мере доработки компонента-отправителя структура его события может дополняться новыми полями, и по ее версии компонент-получатель может определить, какой именно набор полей стоит ожидать).
- **pParameter** – указатель на структуру параметра.

Кроме того, компонент-получатель может при регистрации callback-объекта передать ему указатель на «пользовательский параметр» (**pUserParameter**). При получении события в коде callback-объекта можно будет получить доступ к значению по этому указателю – в частности, это позволяет учесть при обработке события состояние компонента-получателя. Callback-объект должен заранее знать тип пользовательского параметра, чтобы корректно его обработать.

2. Системные события

2.1. Основная информация

Для ряда наиболее важных для разработчика событий среда CODESYS предоставляет упрощенный механизм настройки с помощью вкладки **Системные события** компонента **Конфигурация задач**. Для создания callback-функции нужно нажать кнопку **Добавить обработчик события**. Для каждого события можно добавить только одну callback-функцию.

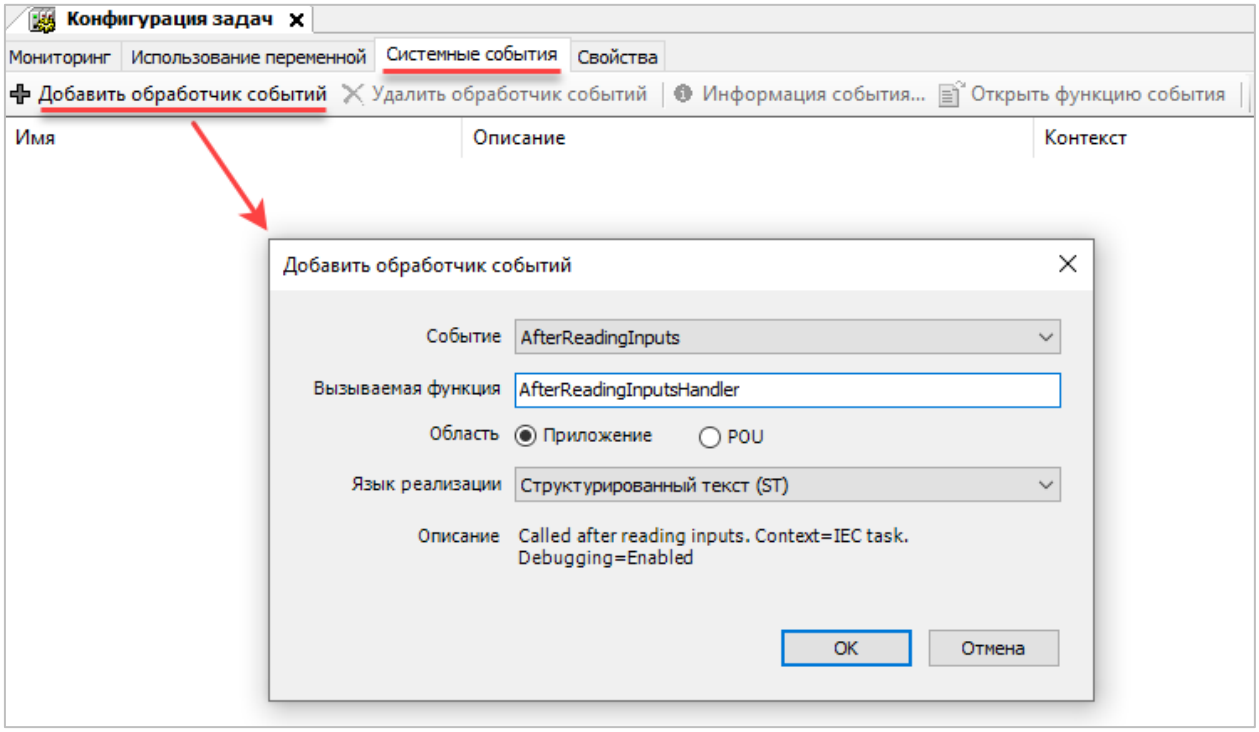


Рис. 2.1. Добавление обработчика событий

- **Событие** – в этом поле выбирается событие, для которого будет создана callback-функция. См. список системных событий в п. 2.2;
- **Вызываемая функция** – в этом поле указывается имя callback-функции, которая будет автоматически создана после нажатия на кнопку ОК (т. е. не надо пытаться указать имя одной из уже существующих функций проекта);
- **Область** – если выбран вариант **Приложение**, то callback-функция будет создана на вкладке **Устройства**. Если выбран вариант **POU**, то callback-функция будет создана на вкладке **POU**;
- **Язык реализации** – язык реализации callback-функции.

Созданная функция будет иметь одну VAR_IN_OUT-переменную, соответствующую параметру события (для разных событий тип переменной будет разным).

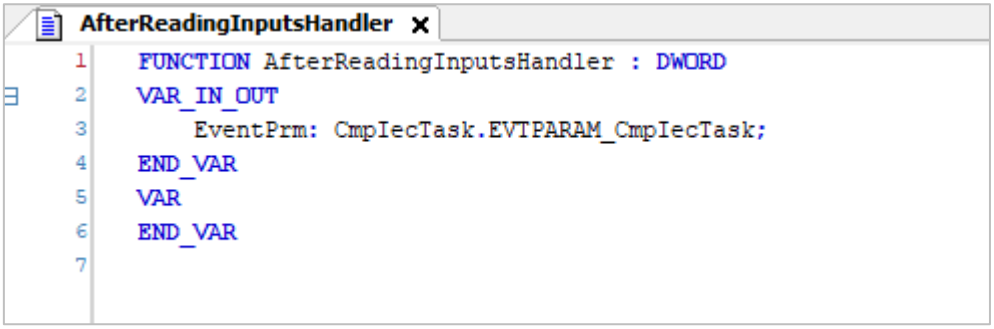


Рис. 2.2. Сигнатура функции-обработчика

С помощью кнопки **Информация события** можно посмотреть ID компонента события, ID самого события, а также ID, версию и названия структуры параметра события (см. информацию в п. 1). Нажатие на кнопку **Открыть функцию события** приводит к открытию в редакторе программирования callback-функции данного события.

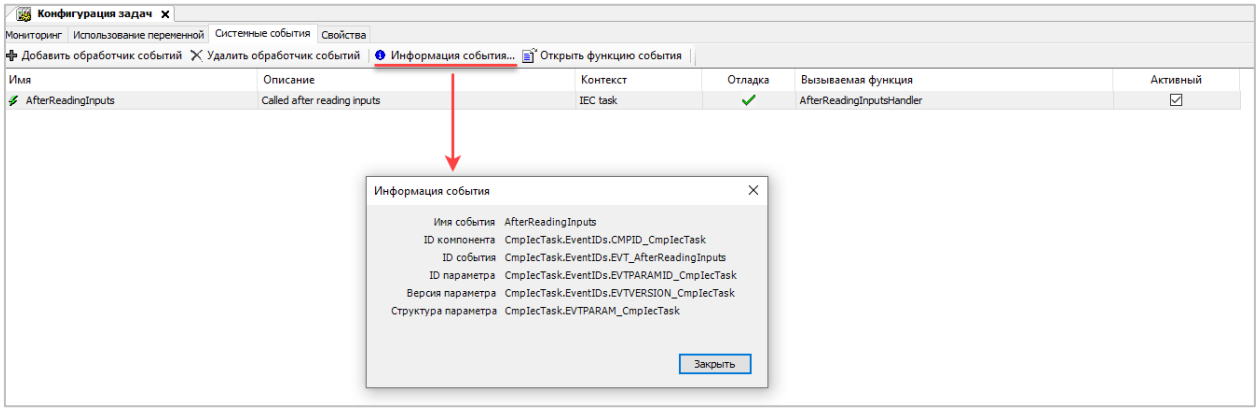


Рис. 2.3. Информация о системном событии

В столбце **Контекст** отображается название задачи, в контексте которой может быть сформировано данное событие. Наличие галочки в столбце **Отладка** означает, что можно провести отладку callback-функции данного события – например, установить в ней точку остановка. Если снять галочку в столбце **Активный** – то данная callback-функция не будет вызываться при возникновении события.

На вкладке **Системные события** можно добавить callback-функции для событий трех компонентов: **CmpIecTask**, **CmpApp** и **ComponentManager**. Более подробная информация о системных событиях приведена в следующих подпунктах.

2.2. Список системных событий

- AfterReadingInputs;
- BeforeReadingInputs;
- AfterWritingOutputs;
- BeforeWritingOutputs;
- CodeInitDone;
- DebugLoop;
- DownloadDone;
- Exception;
- Login;
- Logout;
- OnlineChangeDone;
- PrepareDownload;
- PrepareExit;
- PrepareExitComm;
- PrepareExitTasks;
- PrepareOnlineChange;
- PrepareReset;
- PrepareShutdown;
- PrepareStart;
- PrepareStop;
- ResetDone;
- StartDone;
- StopDone.

2.2.1. AfterReadingInputs

Компонент-отправитель	CmplecTask
Момент отправки события	После чтения входов каждой МЭК-задачи
Контекст	МЭК-задача
Возможность отладки	+
ID компонента	CmplecTask.EventIDs.CMPID_CmplecTask
ID события	CmplecTask.EventIDs.EVT_AfterReadingInputs
ID параметра	CmplecTask.EventIDs.EVTPARAMID_CmplecTask
Версия параметра	CmplecTask.EventIDs.EVTVERSION_CmplecTask
Структура параметра	CmplecTask.EVTPARAM_CmplecTask

2.2.2. BeforeReadingInputs

Компонент-отправитель	CmplecTask
Момент отправки события	Перед чтением входов каждой МЭК-задачи
Контекст	МЭК-задача
Возможность отладки	+
ID компонента	CmplecTask.EventIDs.CMPID_CmplecTask
ID события	CmplecTask.EventIDs.EVT_BeforeReadingInputs
ID параметра	CmplecTask.EventIDs.EVTPARAMID_CmplecTask
Версия параметра	CmplecTask.EventIDs.EVTVERSION_CmplecTask
Структура параметра	CmplecTask.EVTPARAM_CmplecTask

2.2.3. AfterWritingOutputs

Компонент-отправитель	CmplecTask
Момент отправки события	После записи выходов каждой МЭК-задачи
Контекст	МЭК-задача
Возможность отладки	+
ID компонента	CmplecTask.EventIDs.CMPID_CmplecTask
ID события	CmplecTask.EventIDs.EVT_AfterWritingOutputs
ID параметра	CmplecTask.EventIDs.EVTPARAMID_CmplecTask
Версия параметра	CmplecTask.EventIDs.EVTVERSION_CmplecTask
Структура параметра	CmplecTask.EVTPARAM_CmplecTask

2.2.4. BeforeWritingOutputs

Компонент-отправитель	CmpIecTask
Момент отправки события	Перед записью выходов каждой МЭК-задачи
Контекст	МЭК-задача
Возможность отладки	+
ID компонента	CmpIecTask.EventIDs.CMPID_CmpIecTask
ID события	CmpIecTask.EventIDs.EVT_BeforeWritingOutputs
ID параметра	CmpIecTask.EventIDs.EVTPARAMID_CmpIecTask
Версия параметра	CmpIecTask.EventIDs.EVTVERSION_CmpIecTask
Структура параметра	CmpIecTask.EVTPARAM_CmpIecTask

2.2.5. CodInitDone

Компонент-отправитель	CmpApp
Момент отправки события	После инициализации кода приложения в процессе онлайн-изменения (online-change)
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_CodeInitDone
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.6. DebugLoop

Компонент-отправитель	CmpIecTask
Момент отправки события	Генерируется каждый цикл, пока приложение остановлено в точке останова
Контекст	МЭК-задача, в которой сработала точка останова
Возможность отладки	-
ID компонента	CmpIecTask.EventIDs.CMPID_CmpIecTask
ID события	CmpIecTask.EventIDs.EVT_IecTaskDebugLoop
ID параметра	CmpIecTask.EventIDs.EVTPARAMID_CmpIecTask
Версия параметра	CmpIecTask.EventIDs.EVTVERSION_CmpIecTask
Структура параметра	CmpIecTask.EVTPARAM_CmpIecTask

2.2.7. DownloadDone

Компонент-отправитель	CmpApp
Момент отправки события	После загрузки приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_DownloadDone
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.8. Exception

Компонент-отправитель	CmpApp
Момент отправки события	При возникновении исключения
Контекст	МЭК-задача или задача рантайма
Возможность отладки	Зависит от задачи
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_CmpApp_Exception
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppException
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppException
Структура параметра	CmpApp.EVTPARAM_CmpAppException

Более подробную информацию об исключениях см. в п. 5.1.

2.2.9. Login

Компонент-отправитель	CmpApp
Момент отправки события	При подключении к контроллеру из среды CODESYS
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_Login
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppComm
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppComm
Структура параметра	CmpApp.EVTPARAM_CmpAppComm

2.2.10. Logout

Компонент-отправитель	CmpApp
Момент отправки события	При отключении от контроллера из среды CODESYS
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_Logout
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppComm
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppComm
Структура параметра	CmpApp.EVTPARAM_CmpAppComm

2.2.11. OnlineChangeDone

Компонент-отправитель	CmpApp
Момент отправки события	После завершения онлайн-изменения (online-change)
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_OnlineChangeDone
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.12. PrepareDownload

Компонент-отправитель	CmpApp
Момент отправки события	Перед загрузкой приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareDownload
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.13. PrepareExit

Компонент-отправитель	CmpApp
Момент отправки события	Перед завершением работы приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareExit
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.14. PrepareExitComm

Компонент-отправитель	Component Manager
Момент отправки события	Перед завершением работы коммуникационных серверов рантайма (в процессе завершения работы рантайма)
Контекст	Основной рабочий цикл рантайма
Возможность отладки	-
ID компонента	Component_Manager.ComponentID.CMPID_CmpMgr
ID события	Component_Manager.EventIDs.EVT_CmpMgr_PrepareExitComm
ID параметра	Component_Manager.EventIDs.EVTPARAMID_CmpMgr_Shutdown
Версия параметра	Component_Manager.EventIDs.EVTVERSION_CmpMgr_Shutdown
Структура параметра	Component_Manager.EVTPARAM_CmpMgr_Shutdown

2.2.15. PrepareExitTasks

Компонент-отправитель	Component Manager
Момент отправки события	Перед завершением работы задач (в процессе завершения работы рантайма)
Контекст	Основной рабочий цикл рантайма
Возможность отладки	-
ID компонента	Component_Manager.ComponentID.CMPID_CmpMgr
ID события	Component_Manager.EventIDs.EVT_CmpMgr_PrepareExitTasks
ID параметра	Component_Manager.EventIDs.EVTPARAMID_CmpMgr_Shutdown
Версия параметра	Component_Manager.EventIDs.EVTVERSION_CmpMgr_Shutdown
Структура параметра	Component_Manager.EVTPARAM_CmpMgr_Shutdown

2.2.16. PrepareOnlineChange

Компонент-отправитель	CmpApp
Момент отправки события	Перед выполнением онлайн-изменения (online-change)
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareOnlineChange
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.17. PrepareReset

Компонент-отправитель	CmpApp
Момент отправки события	Перед выполнением сброса приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareReset
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppReset
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppReset
Структура параметра	CmpApp.EVTPARAM_CmpAppReset

2.2.18. PrepareShutdown

Компонент-отправитель	Component Manager
Момент отправки события	Перед завершением работы рантайма
Контекст	Основной рабочий цикл рантайма
Возможность отладки	-
ID компонента	Component_Manager.ComponentID.CMPID_CmpMgr
ID события	Component_Manager.EventIDs.EVT_CmpMgr_PrepareShutdown
ID параметра	Component_Manager.EventIDs.EVTPARAMID_CmpMgr_Shutdown
Версия параметра	Component_Manager.EventIDs.EVTVERSION_CmpMgr_Shutdown
Структура параметра	Component_Manager.EVTPARAM_CmpMgr_Shutdown

2.2.19. PrepareStart

Компонент-отправитель	CmpApp
Момент отправки события	Перед запуском приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareStart
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.20. PrepareStop

Компонент-отправитель	CmpApp
Момент отправки события	Перед остановкой приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareStop
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppStop
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppStop
Структура параметра	CmpApp.EVTPARAM_CmpAppStop

2.2.21. ResetDone

Компонент-отправитель	CmpApp
Момент отправки события	После сброса приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareReset
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppReset
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppReset
Структура параметра	CmpApp.EVTPARAM_CmpAppReset

2.2.22. StartDone

Компонент-отправитель	CmpApp
Момент отправки события	После запуска приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareStart
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpApp
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpApp
Структура параметра	CmpApp.EVTPARAM_CmpApp

2.2.23. StopDone

Компонент-отправитель	CmpApp
Момент отправки события	После остановки приложения
Контекст	Задача коммуникации рантайма
Возможность отладки	-
ID компонента	CmpApp.EventIDs.CMPID_CmpApp
ID события	CmpApp.EventIDs.EVT_PrepareStop
ID параметра	CmpApp.EventIDs.EVTPARAMID_CmpAppStop
Версия параметра	CmpApp.EventIDs.EVTVERSION_CmpAppStop
Структура параметра	CmpApp.EVTPARAM_CmpAppStop

3. Библиотека CmpEventMgr

3.1. Основная информация

Библиотека CmpEventMgr представляет собой «обертку» над одноименным компонентом рантайма и позволяет работать с событиями в коде программы. По сравнению с механизмом системных событий, который был рассмотрен в п. 2, данная библиотека имеет следующие преимущества:

- ее можно использовать для обработки событий в коде библиотек (в библиотеке компонент **Конфигурация задач** отсутствует и, соответственно, нельзя добавить обработчик системного события);
- с ее помощью можно обработать любое событие любого компонента рантайма, а не только системные события из п. 2.2;
- с ее помощью можно зарегистрировать в качестве callback-объекта для события не только функцию, но и метод функционального блока;
- с ее помощью можно не только обрабатывать события через callback-объекты, но и формировать события, которые могут быть нужны другим компонентам.

Библиотека **CmpEventMgr** включает в себя две вложенные библиотеки:

- **CmpEventMgr Interfaces** – содержит «интерфейсную» часть библиотеки (типы данных, глобальные константы, интерфейс);
- **CmpEventMgr Implementation** – содержит реализацию библиотеки (функции).

3.2. Библиотека CmpEventMgr Interfaces

3.2.1. Структуры EventParam и EventParam2

Структура **EventParam2** описывает событие. Более подробно назначение полей структуры поясняется в п. 1.

Табл. 3.2.1. Описание полей структуры **EventParam2**

Поле структуры	Тип	Описание
EventId	DWORD	Идентификатор события. Старшее слово – код класса события, младшее – код события
CmpIdProvider	DWORD	Идентификатор компонента, являющегося отправителем данного события
usParamId	WORD	Идентификатор параметра события
usVersion	WORD	Версия параметра события
pParameter	POINTER TO BYTE	Указатель на параметр события
pUserParameter	POINTER TO BYTE	Указатель на данные приложения пользователя, которые будут переданы в вызове callback-объекта, зарегистрированного для данного события

В библиотеке также присутствует структура **EventParam**, отличающаяся от **EventParam2** лишь отсутствием поля **pUserParameter** (она использовалась в старых версиях библиотеки и оставлена для обратной совместимости).

3.2.2. Список глобальных констант Constants

Табл. 3.2.2. Описание констант списка глобальных констант **Constants**

Константа	Тип	Значение	Описание
EVENT_CALLBACKS_NO_LIMIT	UDINT	16#FFFFFFFF	Может использоваться в функции EventCreate2 в качестве значения входа **nCallbacksPossible**. Означает, что для данного события может быть зарегистрировано неограниченное число callback-объектов
CMPID_CmpEventMgr	UDINT	16#5B	Идентификатор компонента **CmpEventMgr**. Необходим для возможности обработки событий данного компонента
ITFID_ICmpEventCallback	UDINT	16#25	Идентификатор интерфейса ICmpEventCallback

3.2.3. Список глобальных констант EventClass

Константы этого списка используются для описания класса события, характеризующего его «критичность». Применяются в функции EventCreateEventID (вход **Class**) и EventGetClass (выход функции).

Табл. 3.2.3. Описание констант списка глобальных констант **EventClass**

Константа	Тип	Значение	Описание
EVTCLASS_NONE	UDINT	16#0	Некорректный класс события (или класс отсутствует). Не может быть использован при создании события[1]
EVTCLASS_ALL	UDINT	16#FFFF	Фильтр для всех событий. Не может быть использован при создании события
EVTCLASS_INFO	UDINT	16#1	Класс «информационное сообщение»
EVTCLASS_WARNING	UDINT	16#2	Класс «предупреждение»
EVTCLASS_ERROR	UDINT	16#4	Класс «ошибка»
EVTCLASS_EXCEPTION	UDINT	16#8	Класс «исключение»
EVTCLASS_VENDOR_SPEC	UDINT	16#1000	Начиная с этого кода производители контроллеров могут создавать собственные классы событий

[1] Но, строго говоря, в рантайме есть одно событие такого класса – **EVT_CmpNSSOemCallback** (формируется компонентом **CmpNameServiceServer**)

3.2.4. Интерфейс ICmpEventCallback

Если в качестве callback-объекта используется метод функционального блока (см. функции EventRegisterCallback и EventRegisterCallback2) – то этот блок должен реализовывать (**IMPLEMENTS**) интерфейс **ICmpEventCallback**. Этот интерфейс содержит единственный метод с названием **EventCallback** – именно он будет однократно вызван при наступлении события. Единственным входом этого метода является указатель на структуру события. Метод возвращает код ошибки из библиотеки CmpErrors.

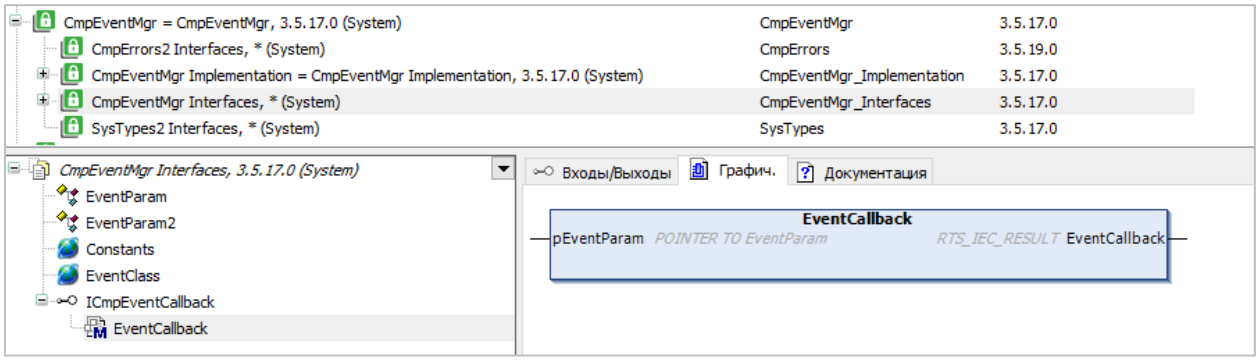


Рис. 3.2.4.1. Сигнатура метода **EventCallback**

3.3. Библиотека CmpEventMgr Implementation

3.3.1. Функция EventCreateEventID

Вызывает: отправитель события

Функция **EventCreateEventID** используется для формирования идентификатора события на основе его кода (**Event**) и класса (**Class**). Код ошибки возвращается через вход-выход **Result** (см. константы библиотеки CmpErrors).

Рис. 3.3.1. Внешний вид функции **EventCreateEventID** на языке CFC

Табл. 3.3.1. Описание переменных функции **EventCreateEventID**

Переменная	Тип	Описание
Входы (VAR INPUT)		
Event	UINT	Код события
Class	UINT	Класс события (см. EventClass)
Входы-выходы (VAR_IN_OUT)		
Result	RTS_IEC_RESULT	Результат выполнения функции
Выход		
EventCreateEventID	UDINT	Идентификатор события

3.3.2. Функция EventCreate, EventCreate2

Вызывает: отправитель события

Функция **EventCreate2** используется для создания события с идентификатором **EventId** для компонента с идентификатором **CmpIdProvider**, и получения его дескриптора. Если данное событие уже было создано, то возвращается его дескриптор. Созданное событие можно впоследствии «отправить» (сгенерировать) с помощью функции EventPost или EventPost2. Вход **nCallbacksPossible** позволяет ограничить максимальное число объектов для данного события (EVENT_CALLBACKS_NO_LIMIT – без ограничений). Код ошибки возвращается через вход-выход **Result** (см. константы библиотеки CmpErrors).

Функция **EventCreate** отличается от **EventCreate2** только отсутствием входа **nCallbacksPossible**.

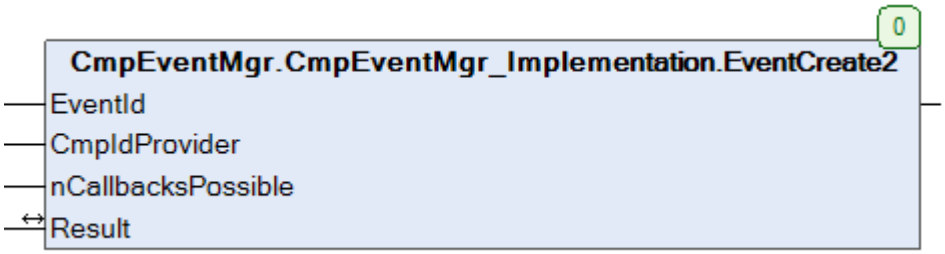


Рис. 3.3.2. Внешний вид функции **EventCreate2** на языке CFC

Табл. 3.3.2. Описание переменных функции **EventCreate2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
EventId	UDINT	Идентификатор события
CmpIdProvider	UDINT	Идентификатор компонента-отправителя
nCallbacksPossible	UDINT	Максимально возможное количество callback-объектов для данного события (если используется константа EVENT_CALLBACKS_NO_LIMIT – то ограничений нет)
Входы-выходы (VAR_IN_OUT)		
Result	RTS_IEC_RESULT	Результат выполнения функции
Выход		
EventCreate2	RTS_IEC_HANDLE	Дескриптор события

3.3.3. Функция EventPost, EventPost2

Вызывает: отправитель события

Функция **EventPost2** используется для «отправки» (формирования) события с дескриптором **hEvent** и структурой, размещенной по указателю **pEventParam**. Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

Функция **EventPost2** отличается от **EventPost** только типом входа **pEventParam** – у неё он POINTER TO EventParam, а не POINTER TO EventParam2.

Рис. 3.3.3. Внешний вид функции **EventPost2** на языке CFC

Табл. 3.3.3. Описание переменных функции **EventPost2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
pEventParam	POINTER TO EventParam2	Указатель на структуру события
Выход		
EventPost2	RTS_IEC_RESULT	Результат выполнения функции

3.3.4. Функция EventPostByEvent, EventPostByEvent2

Вызывает: отправитель события

Функция **EventPostByEvent2** используется для «отправки» (формирования) события с идентификатором **EventId** от компонента с идентификатором **CmpIdProvider** и структурой, размещенной по указателю **pEventParam**. В отличие от функции EventPost2 – в данном случае не требуется дескриптор события (то есть не нужно предварительно вызывать функцию EventCreate2). Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

Функция **EventPostByEvent2** отличается от **EventPostByEvent** только типом входа **pEventParam** – у неё он POINTER TO EventParam, а не POINTER TO EventParam2.

Рис. 3.3.4. Внешний вид функции **EventPostByEvent2** на языке CFC

Табл. 3.3.4. Описание переменных функции **EventPostByEvent2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
EventId	UDINT	Идентификатор события
CmpIdProvider	UDINT	Идентификатор компонента-отправителя
pEventParam	POINTER TO EventParam2	Указатель на структуру события
Выход		
EventPostByEvent2	RTS_IEC_RESULT	Результат выполнения функции

3.3.5. Функция EventDelete2

Вызывает: отправитель события

Функция **EventDelete2** используется для удаления события с дескриптором **hEvent**, если оно больше не требуется компоненту-отправителю. Код ошибки возвращается выход функции (см. константы библиотеки CmpErrors).

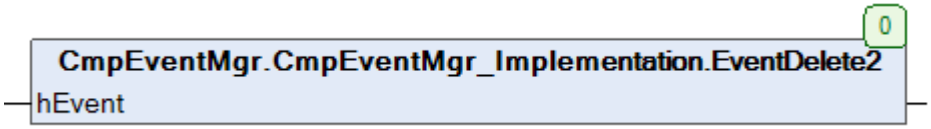


Рис. 3.3.6. Внешний вид функции **EventDelete2** на языке CFC

Табл. 3.3.6. Описание переменных функции **EventDelete2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
Выход		
EventDelete2	RTS_IEC_RESULT	Результат выполнения функции

3.3.6. Функция EventOpen

Вызывает: получатель события

Функция **EventOpen** используется для проверки существования события (т. е. того, что компонент-отправитель создал его с помощью функции EventCreate или EventCreate2) с идентификатором **EventId** для компонента с идентификатором **CmpIdProider**, и получения его дескриптора. Код ошибки возвращается через вход-выход **Result** (см. константы библиотеки CmpErrors).

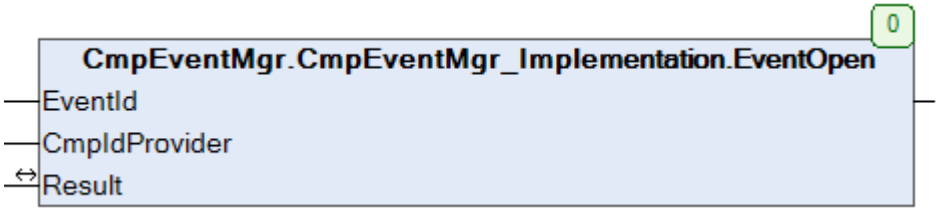


Рис. 3.3.6. Внешний вид функции **EventOpen** на языке CFC

Табл. 3.3.6. Описание переменных функции **EvenOpen**

Переменная	Тип	Описание
Входы (VAR INPUT)		
EventId	UDINT	Идентификатор события
CmpIdProvider	UDINT	Идентификатор компонента-отправителя
Входы-выходы (VAR_IN_OUT)		
Result	RTS_IEC_RESULT	Результат выполнения функции
Выход		
EventOpen	RTS_IEC_HANDLE	Дескриптор события

3.3.7. Функция EventRegisterCallback, EventRegisterCallback2

Вызывает: получатель события

Функция **EventRegisterCallback2** используется для регистрации в качестве callback-объекта события с дескриптором **hEvent** метода функционального блока. Функциональный блок должен реализовывать (**IMPLEMENTS**) интерфейс ICmpEventCallback. На вход функции **pICallback** следует передать экземпляр данного блока. На вход **pUserParameter** передается указатель на переменную приложения пользователя, который будет передан в callback-метод **EventCallback** при его вызове (в одноименное поле структуры EventParam2). Это позволяет учесть в коде метода текущее состояние приложения в момент возникновения события. Код ошибки возвращается через вход-выход **Result** (см. константы библиотеки CmpErrors). Функция возвращает дескриптор интерфейса, который может быть потом использован для дерегистрации callback'а в вызове функции EventUnregisterCallback.

Функция **EventRegisterCallback** отличается от **EventRegisterCallback2** только отсутствием входа **pUserParameter**.

Рис. 3.3.7. Внешний вид функции **EventRegisterCallback2** на языке CFC

Табл. 3.3.7. Описание переменных функции **EventRegisterCallback2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
piCallback	ICmpEventCallback	Экземпляр функционального блока, метод которого будет использоваться в качестве обработчика события (см. описание перед таблицей)
pUserParameter	POINTER TO BYTE	Указатель на переменную приложения, который будет доступен в callback-методе
Входы-выходы (VAR_IN_OUT)		
Result	RTS_IEC_RESULT	Результат выполнения функции
Выход		
EventRegisterCallback2	RTS_IEC_HANDLE	Дескриптор интерфейса

3.3.8. Функция EventRegisterCallbackFunction, EventRegisterCallbackFunction2

Вызывает: получатель события

Функция **EventRegisterCallback2** используется для регистрации в качестве callback-объекта события с дескриптором **hEvent** функции. Сигнатура callback-функции должно соответствовать сигнатуре метода **EventCallback** интерфейса ICmpEventCallback. Указатель на callback-функцию (полученный с помощью оператора **ADR**) должен быть передан на вход **pfCallbackFunction**. На вход **pUserParameter** передается указатель на переменную приложения пользователя, который будет передан в callback-функцию при ее вызове (в одноименное поле структуры EventParam2). Это позволяет учесть в коде функции текущее состояние приложения в момент возникновения события. Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

Функция **EventRegisterCallbackFunction** отличается от **EventRegisterCallbackFunction2** только отсутствием входа **pUserParameter**.

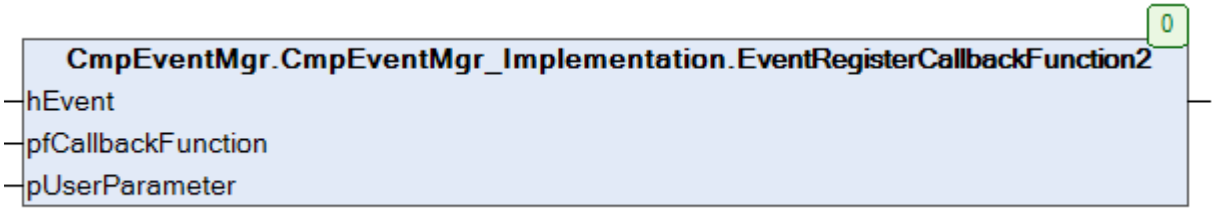


Рис. 3.3.8. Внешний вид функции **EventRegisterCallback2** на языке CFC

Табл. 3.3.8. Описание переменных функции **EventRegisterCallback2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
pfCallbackFunction	POINTER TO BYTE	Указатель на callback-функцию
pUserParameter	POINTER TO BYTE	Указатель на переменную приложения, который будет доступен в callback-функции
Выход		
EventRegisterFunctionCallback2	RTS_IEC_RESULT	Результат выполнения функции

3.3.9. Функция EventUnregisterCallback

Вызывает: получатель события

Функция **EventUnregisterCallback** используется для дерегистрации callback-метода события с дескриптором **hEvent**, зарегистрированного в вызове функции EventRegisterCallback или EventRegisterCallback2. На вход **hInterface** следует передать дескриптор интерфейса, полученный в результате вызова функции **EventRegisterCallback** или **EventRegisterCallback2**. Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

Рис. 3.3.9. Внешний вид функции **EventUnregisterCallback** на языке CFC

Табл. 3.3.9. Описание переменных функции **EventUnregisterCallback**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
hInterface	RTS_IEC_HANDLE	Дескриптор интерфейса
Выход		
EventUnregisterCallback	RTS_IEC_RESULT	Результат выполнения функции

3.3.10. Функция EventUnregisterCallbackFunction, EventUnregisterCallbackFunction2

Вызывает: получатель события

Функция **EventUnregisterCallbackFunction2** используется для дерегистрации callback-функции события с дескриптором **hEvent**, зарегистрированного в вызове функции EventRegisterCallbackFunction2. На входы **pfCallbackFunction** и **pUserParameter** следует передать те же самые значения, которые предавались в вызове функции **EventRegisterCallbackFunction2**. Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

Функция **EventUnregisterCallbackFunction** отличается от **EventUnregisterCallbackFunction2** только отсутствием входа **pUserParameter** и используется для дерегистрации callback-функции, зарегистрированной в вызове функции EventRegisterCallbackFunction.

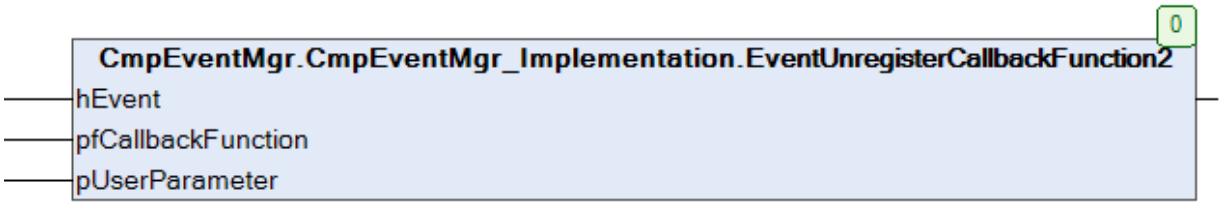


Рис. 3.3.10. Внешний вид функции **EventUnregisterCallbackFunction2** на языке CFC

Табл. 3.3.10. Описание переменных функции **EventUnegisterCallbackFunction2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
pfCallbackFunction	POINTER TO BYTE	Указатель на callback-функцию
pUserParameter	POINTER TO BYTE	Указатель на переменную приложения, который был доступен в callback-функции
Выход		
EventUnregisterCallbackFunction2	RTS_IEC_RESULT	Результат выполнения функции

3.3.11. Функция EventRegisteredCallbacks

Функция **EventRegisteredCallbacks** возвращает число callback-объектов, зарегистрированных для события с дескриптором **hEvent**. Код ошибки возвращается через вход-выход **Result** (см. константы библиотеки CmpErrors).

Рис. 3.3.11. Внешний вид функции **EventRegisteredCallbacks** на языке CFC

Табл. 3.3.11. Описание переменных функции **EventRegisteredCallbacks**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
Входы-выходы (VAR_IN_OUT)		
Result	RTS_IEC_RESULT	Результат выполнения функции
Выход		
EventRegisteredCallbacks	UDINT	Число зарегистрированных callback-объектов

3.3.12. Функция EventGetClass

Функция **EventGetClass** возвращает класс события, извлеченный из идентификатора события **EventId**.

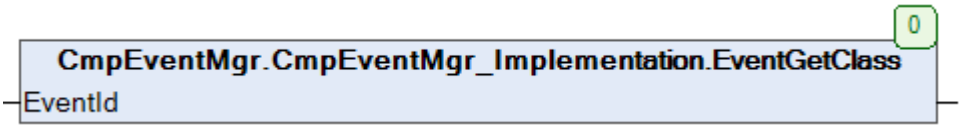


Рис. 3.3.12. Внешний вид функции **EventGetClass** на языке CFC

Табл. 3.3.12. Описание переменных функции **EventGetClass**

Переменная	Тип	Описание
Входы (VAR INPUT)		
EventId	UDINT	Идентификатор события
Выход		
EventGetClass	UINT	Класс события (см. EventClass)

3.3.13. Функция EventGetEvent

Функция **EventGetEvent** возвращает код события, извлеченный из идентификатора события **EventId**.

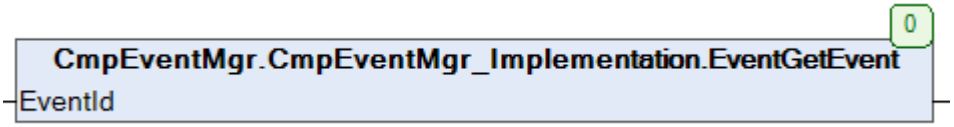


Рис. 3.3.13. Внешний вид функции **EventGetEvent** на языке CFC

Табл. 3.3.13. Описание переменных функции **EventGetEvent**

Переменная	Тип	Описание
Входы (VAR INPUT)		
EventId	UDINT	Идентификатор события
Выход		
EventGetEvent	UINT	Код события

3.3.14. Функция EventClose2

Функция **EventClose2** используется для закрытия события с дескриптором **hEvent** после дерегистрации его callback-объекта. Код ошибки возвращается через выход функции (см. константы библиотеки CmpErrors).

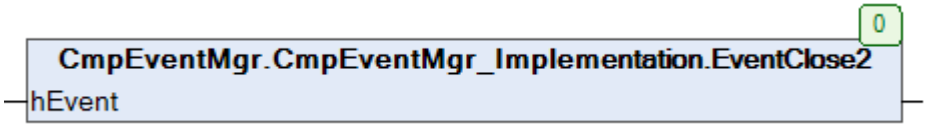


Рис. 3.3.14. Внешний вид функции **EventClose2** на языке CFC

Табл. 3.3.14. Описание переменных функции **EventClose2**

Переменная	Тип	Описание
Входы (VAR INPUT)		
hEvent	RTS_IEC_HANDLE	Дескриптор события
Выход		
EventClose2	RTS_IEC_RESULT	Результат выполнения функции

4. Пример работы с событиями

Пример создан в среде **CODESYS V3.5 SP17 Patch 3** для виртуального **контроллера CODESYS Control Win V3.** Он иллюстрирует информацию, приведенную в п. 2 и п. 3.

В рамках примера рассматривается:

- обработка системного события Login (подключение к контроллеру из среды CODESYS) с помощью вкладки Конфигурация задач – Системные события;
- обработка события **CreateBootprojectDone** (создание загрузочного приложения) с помощью библиотеки CmpEventMgr;
- создание собственного события и его обработка с помощью библиотеки **CmpEventMgr**.

Пример содержит визуализацию.

Ссылка на проект примера: скачать

4.1. Пример обработки системного события Login с помощью вкладки Конфигурация задач – Системные события

На вкладке **Конфигурация – Системные события** добавлено событие **Login**. При добавлении события было введено имя функции-обработчика (**LoginFromIdeHandler**) – и она была автоматически добавлена в дерево проекта (так как был выбран режим **Область – Приложение**).

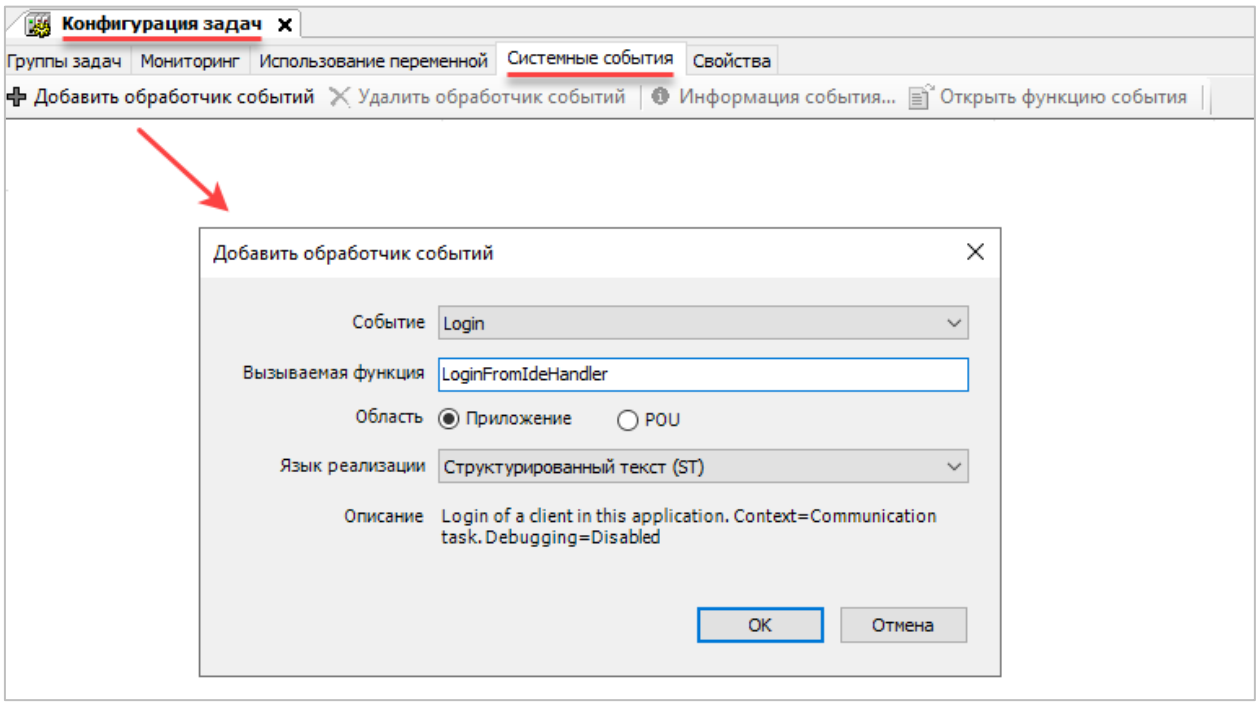


Рис. 4.1.1. Создание функции-обработчика системного события **Login**

В коде функции в рамках примера выполняется лишь инкремент переменной **udiLoginFromIdeCounter** из списка глобальных переменных **GVL**.

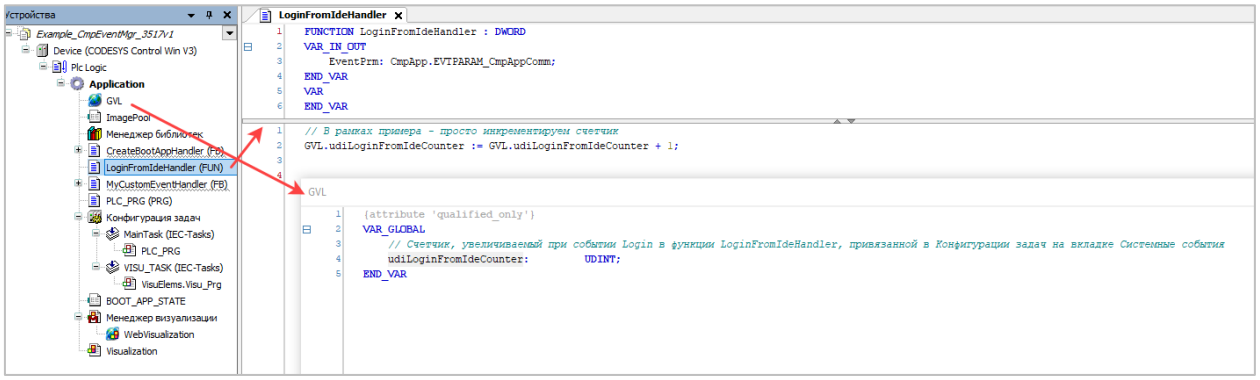


Рис. 4.1.2. Код функции-обработчика **LoginFromIdeHandler**

Параметр события (**EventPrm**) в рамках примера не используется. Для события **Login** он содержит идентификатор сессии связи (**udiSessionId**), идентификатор приложения (**udiAppSessionId**) и указатель на структуру данных приложения (см. структуру APPLICATION в библиотеке CmpApp). В реальных проектах вам может потребоваться обрабатывать структуру параметра события.

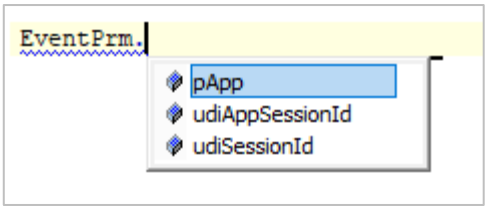


Рис. 4.1.3. Содержимое структуры параметры для события **Login**

Каждое подключение к контроллеру (с помощью команды **Онлайн – Логин**) будет приводить к формированию события **Login** и, соответственно, вызову функции **LoginFromIdeHandler**, в которой будет происходить инкремент счетчика. Текущее значение счетчика будет отображаться в визуализации. При нажатии на кнопку **Сброс** значение счетчика будет обнулено – за это отвечает первый фрагмент кода в программе **PLC_PRG**.

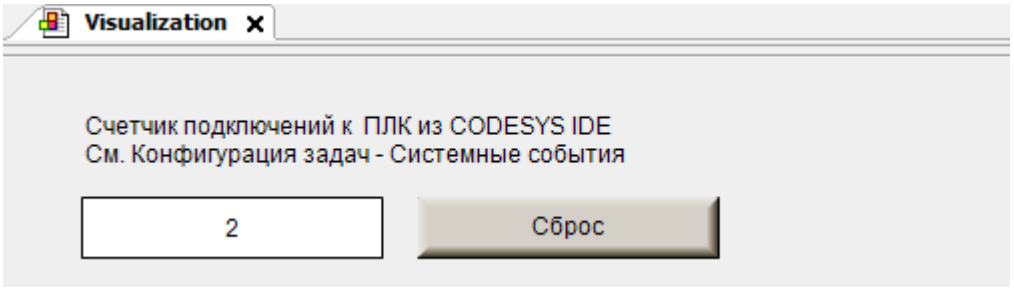


Рис. 4.1.4. Отображение счетчика подключений к контроллеру в визуализации примера

4.2. Пример обработки события CreateBootprojectDone с помощью библиотеки CmpEventMgr

Предположим, что требуется отобразить в визуализации контроллера информацию о том, было ли создано загрузочное приложение. Среди системных событий подходящего события, информирующего о факте создания загрузочного приложения, нет.

Поэтому обработаем его с помощью библиотеки CmpEventMgr в коде нашего приложения. Для начала нужно определить, к какому компоненту относится нужное нам событие. Опытным путем можно установить, что оно формируется компонентом CmpApp. В одноименной библиотеке в списке глобальных констант EventIDs можно найти его идентификатор:

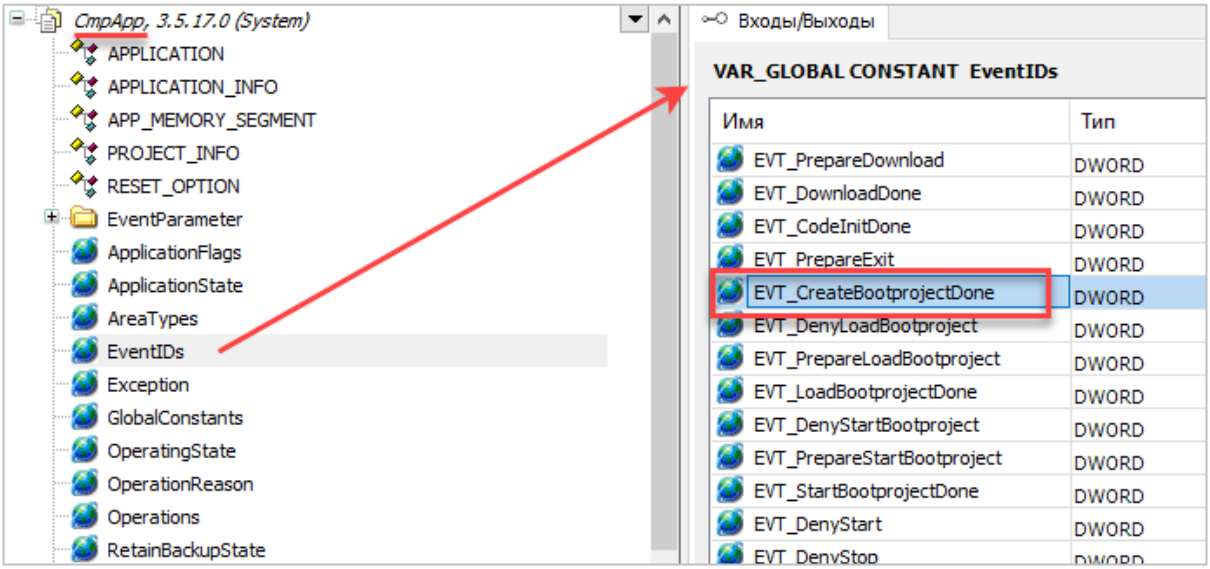


Рис. 4.2.1. Идентификатор события **CreateBootProjectDone**

В этом же списке присутствуют другие события, а также константы идентификаторов и версий их параметров. Для события **CreateBootProjectDone** нет «своих» идентификаторов и версии параметра, поэтому используются «общебиблиотечные» – **EVTPARAMID_CmpApp** и **EVTVERSION_CmpApp**. Также на рисунке ниже виден идентификатор компонента (**CMPID_CmpApp**).

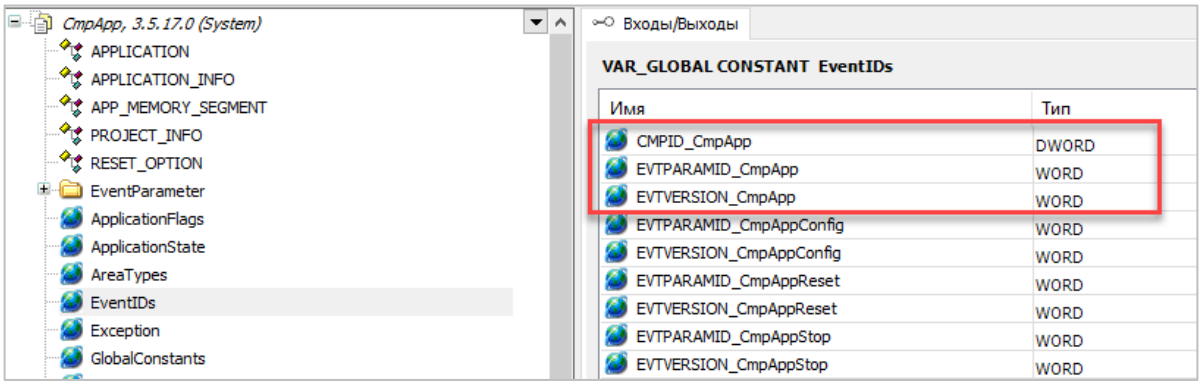


Рис. 4.2.2. Идентификатор компонента **CmpApp**, а также идентификатор и версия «общебиблиотечного» параметра

Структуры параметров размещены в папке **EventParameter**:

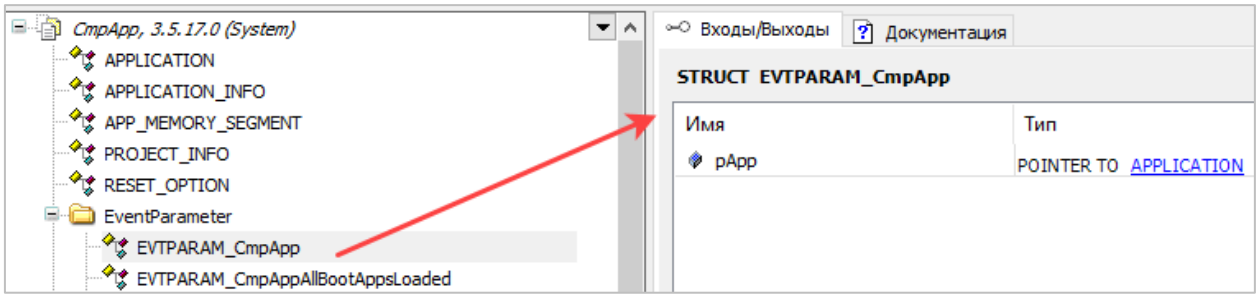


Рис. 4.2.3. Структуры параметров событий компонента **CmpApp**

Этой информации достаточно для обработки события. В качестве callback-объекта в примере используется метод функционального блока. Этот функциональный блок называется **CreateBootAppHandler**. Он реализует интерфейс ICmpEventCallback из библиотеки **CmpEventMgr Interfaces**.

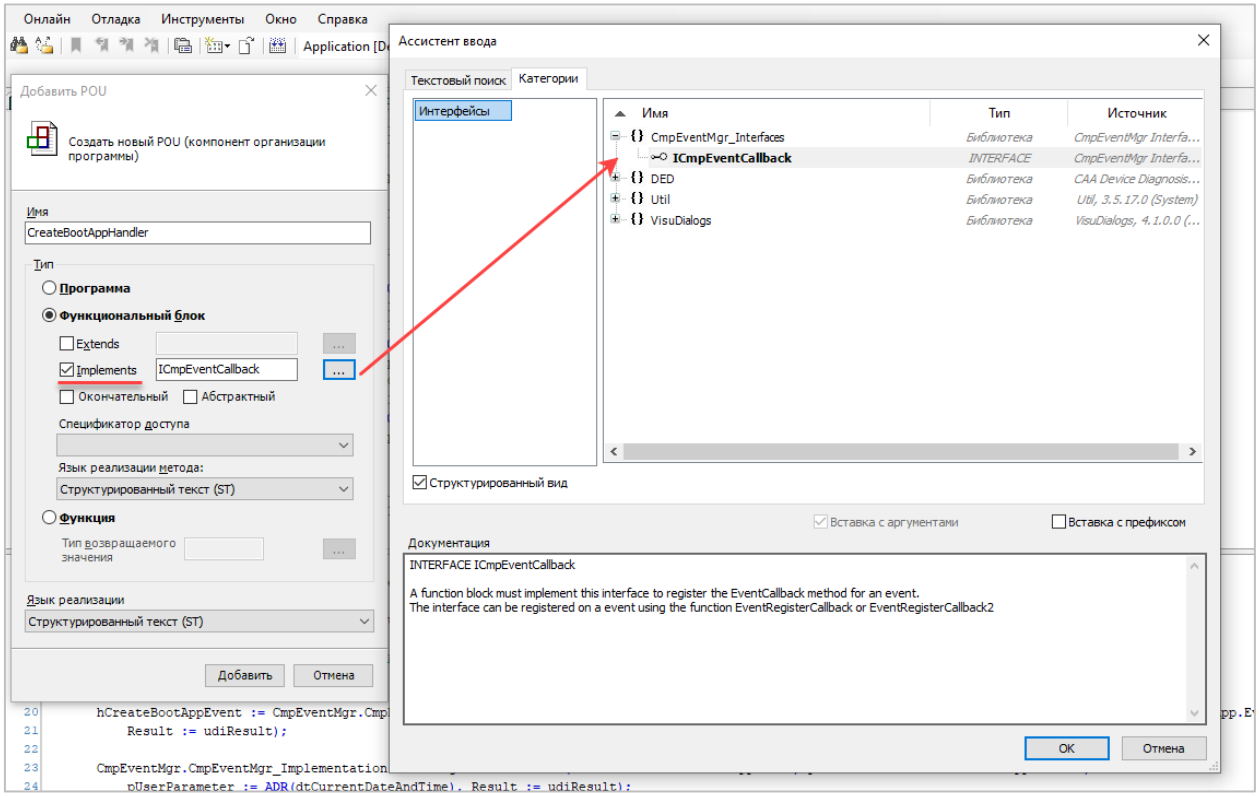


Рис. 4.2.4. Создание ФБ-обработчика события

Во входных переменных (**VAR_INPUT**) ФБ объявлены два указателя – с помощью них в программу будет передаваться флаг создания загрузочного приложения (**pxCreateBootAppDone**) и метка времени его создания (**pdtCreateBootAppDateTime**).

CreateBootAppHandler

```
1  FUNCTION_BLOCK CreateBootAppHandler IMPLEMENTS ICmpEventCallback
2  VAR_INPUT
3      // Флаг "Создано загрузочное приложение"
4      pxCreateBootAppDone:                POINTER TO BOOL;
5      // Время создания загрузочного приложения (UTC+0)
6      pdtCreateBootAppDateTime:           POINTER TO DT;
7  END_VAR
8  VAR_OUTPUT
9  END_VAR
10 VAR
11 END_VAR
```

Рис. 4.2.5. Объявление переменных ФБ-обработчика события

Метод **EventCallback**, полученный от интерфейса ICmpEventCallback, будет однократно вызван системой исполнения при наступлении интересующего нас события (для этого потребуется зарегистрировать callback – см. информацию на следующей странице). Вызывать этот метод (или сам экземпляр ФБ) в коде программы для этого необязательно (впрочем, вы можете это делать для каких-то своих целей – например, если блок реализует дополнительную логику).

Для возвращаемого методом значения сразу добавим в тип переменной пространства имен нужных библиотек – иначе вы получите ошибку при компиляции проекта.

CreateBootAppHandler.EventCallback

```
1  {attribute 'm4export_hide'}
2  {warning 'добавить реализацию метода'}
3  (* The interface ICmpEventCallback must be registered for an event. If this event is posted, this callback method is called.*)
4  METHOD EventCallback : CmpEventMgr.SysTypes.RTS_IEC_RESULT
5  VAR_INPUT
6      (*Pointer to the event parameters, see Struct EventParam*)
7      pEventParam : POINTER TO EventParam;
8  END_VAR
9  VAR
10     pEventParam2: POINTER TO EventParam2;
11 END_VAR
```

```
1  pxCreateBootAppDone^ := TRUE;
2  // приводим указатель на EventParam к указателю на EventParam2, чтобы получить доступ к pUserParameter
3  pEventParam2 := pEventParam;
4  // используем pUserParameter в коде метода для получения текущего системного времени
5  MEM.MemMove(pEventParam2^.pUserParameter, pdtCreateBootAppDateTime, SIZEOF(DT) );
```

Рис. 4.2.6. Код метода **EventCallback**

Чтобы вернуть по указателю **pdtCreateBootAppDateTime** метку времени создания загрузочного приложения – надо как-то получить эту метку в момент возникновения события. Это можно сделать различными способами, но в рамках примера мы рассмотрим ее передачу через «пользовательский» параметр (**UserParameter**). Но вход метода **pEventParam** имеет тип EventParam, а поле **pUserParameter** присутствует только в структуре **EventParam2**. Поэтому объявим в локальных переменных метода экземпляр этой структуры и присвоим в него значение указателя **pEventParam** – так мы сможем обратиться к «пользовательскому» параметру.

В коде метода мы присваиваем по указателю **pxCreateBootAppDone** значение **TRUE** (флаг создания пользовательского приложения), а по указателю **pdtCreateBootAppDateTime** копируем значение «пользовательского» параметра с помощью функции MemMove из библиотеки **CAA Memory**.

Теперь осталось написать код, в котором будет производиться регистрация экземпляра данного ФБ в качестве callback-объекта и формироваться значение пользовательского параметра.

Переменные программы **PLC_PRG**, связанные с примером:

```
// --- Пример обработки события CreateBootprojectDone
// (Создание загрузочного приложения) ---

// Текущее системное время
dtCurrentDateAndTime:                 DT;
// Команда регистрации обработчика события
xRegisterCreateBootAppHandler:        BOOL := TRUE;
// Обработчик события
fbCreateBootAppHandler:               CreateBootAppHandler;
// Флаг "Создано загрузочное приложение"
xCreateBootAppDone:                   BOOL;
// Время создания загрузочного приложения (UTC+0)
dtCreateBootAppDateTime:              DT;
// Идентификатор события CreateBootprojectDone
udiCreateBootAppEventId:              UDINT;
// Дескриптор события CreateBootprojectDone
hCreateBootAppEvent:                  CmpEventMgr.SysTypes.RTS_IEC_HANDLE;
// Код ошибки
udiResult:                            UDINT;
```

Код программы **PLC_PRG**, связанный с примером:

```
// получаем системное время контроллера (в UTC+0)
dtCurrentDateAndTime := TO_DT(SysTimeRtc.SysTimeRtcGet(udiResult) );

// --- Пример обработки события CreateBootprojectDone
IF xRegisterCreateBootAppHandler THEN

   // передаем адреса переменных, в которые ФБ-обработчик запишет результаты
   // обработки события
   fbCreateBootAppHandler.pxCreateBootAppDone      :=
      ADR(xCreateBootAppDone);
  fbCreateBootAppHandler.pdtCreateBootAppDateTime :=
     ADR(dtCreateBootAppDateTime);

   hCreateBootAppEvent :=
      CmpEventMgr.CmpEventMgr_Implementation.EventOpen(EventId :=
         CmpApp.EventIDs.EVT_CreateBootprojectDone, CmpIdProvider :=
            CmpApp.EventIDs.CMPID_CmpApp, Result := udiResult);

   CmpEventMgr.CmpEventMgr_Implementation.EventRegisterCallback2(hEvent :=
      hCreateBootAppEvent, pICallback := fbCreateBootAppHandler,
         pUserParameter := ADR(dtCurrentDateAndTime), Result := udiResult);

        xRegisterCreateBootAppHandler := FALSE;
END_IF
```

Сначала мы получаем системное время контроллера с помощью функции SysTimeRtcGet с из библиотеки SysTimeRtc и сохраняем его в переменную **dtCurrentDateAndTime**. Время возвращается без учета часового пояса (UTC+0). У реальных контроллеров различных производителей обычно есть свои способы получения системного времени – например, в контроллерах ОВЕН для этого используется компонент **OwenRTC** из дерева проекта.

Весь последующий код выполняется однократно при старте приложения (так как переменная **xRegisterCreateBootAppHandler** имеет начальное значение **TRUE** и сбрасывается в **FALSE** в конце рассматриваемого фрагмента кода).

На входы экземпляра блока передаются адреса переменных программы. При возникновении события в коде **EventCallback** будет произведена запись значений по этим адресам (см. предыдущую страницу).

Далее с помощью функции EventOpen открывается интересующее нас событие. Идентификатор события и идентификатор компонента были найдены в библиотеке CmpApp в самом начале пункта. Функция возвращает дескриптор открытого события, который сохраняется в переменную **hCreateBootAppEvent**.

После этого вызывается функция EventRegisterCallback2, с помощью которой выполняется регистрация экземпляра **fbCreateBootAppHandler** блока **CreateBootAppHandle** в качестве обработчика события. Первым аргументом функции является дескриптор события. На вход **pUserParameter** передается адрес переменной **dtCurrentDateAndTime**, в которую записывается значение системного времени (см. выше).

В рамках примера для всех функций используется одна и та же переменная для сохранения кода ошибки (**udiResult**). В реальных проектах рекомендуется использовать отдельные переменные для каждой функции и реализовать обработку ошибок.

Проверим работу примера. Загрузите проект примера в контроллер, не устанавливая галочку **Обновить загрузочное приложение** (окно с этой галочкой появляется только при попытке загрузке измененного проекта; если вы загружаете проект в первый раз – то этого окна не будет).

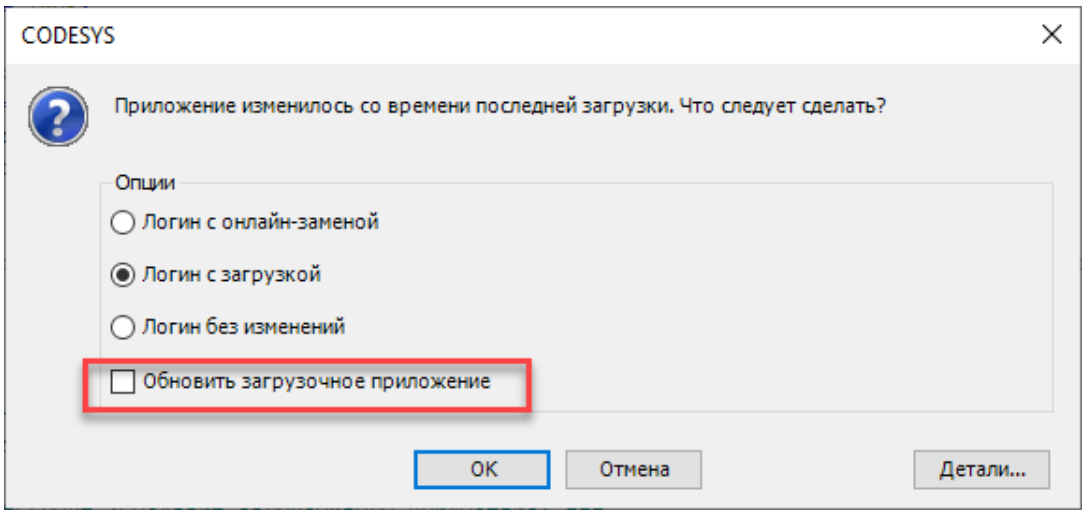


Рис. 4.2.7. Варианты загрузки измененного приложения

Запустите проект (**Отладка – Старт**) и выполните команду **Онлайн – Создать загрузочное приложение**. Наблюдайте изменение информации о загрузочном приложении в визуализации (к первому прямоугольнику привязан список текстов **BOOT_APP_STATE**, в качестве индекса которого используется переменная **xCreateBootAppDone**, преобразованная к целочисленному типу):

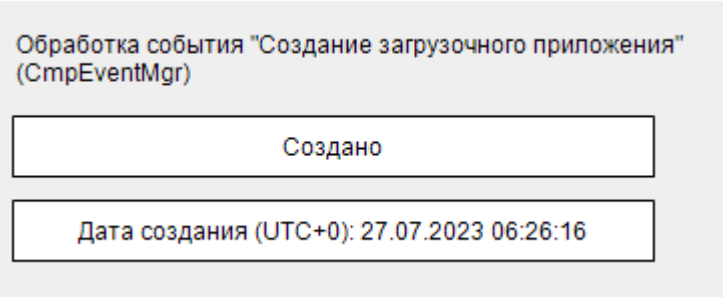


Рис. 4.2.8. Отображение информации о факте создания загрузочного приложении в визуализации примера

В рамках примера не демонстрируется дерегистрация callback-объекта (см. функцию EventUnregisterCallback и закрытие события (см. функцию EventClose2).

4.3. Пример создания и обработки пользовательского события с помощью библиотеки CmpEventMgr

В ряде случаев разработчику может потребоваться создавать и использовать собственные события. В рамках данного пункта рассматривается синтетический пример, в котором создание, отправка и обработка события выполняется в рамках одной и той же программы **PLC_PRG**. В реальных проектах отправка и обработка событий выполняется в разных местах – например, событие отправляется из библиотеки, а обрабатывается в приложении пользователя или другой библиотеке (или наоборот).

Как и в прошлом пункте – в качестве callback-объекта в примере используется метод функционального блока. Этот функциональный блок называется **MyCustomEventHandler**. Он реализует интерфейс ICmpEventCallback из библиотеки **CmpEventMgr Interfaces**.

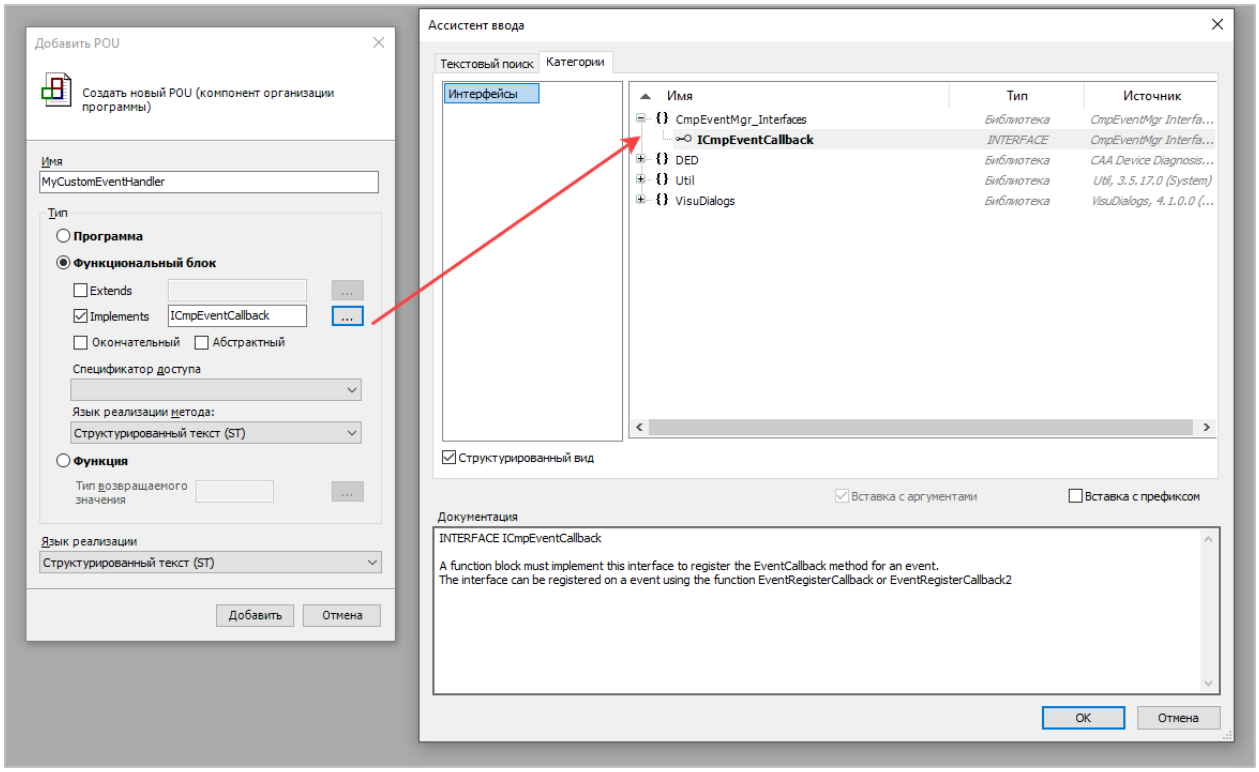


Рис. 4.3.1. Создание ФБ-обработчика события

Во входных переменных (**VAR_INPUT**) ФБ объявлен указатель на **UDINT** – с помощью него в программу будет возвращаться счетчик появления данного события (в реальном проекте, естественно, нужно реализовать свою логику обработки события).

MyCustomEventHandler

```
FUNCTION_BLOCK MyCustomEventHandler IMPLEMENTS ICmpEventCallback
VAR_INPUT
    pudiMyCustomEventCounter: POINTER TO UDINT;
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

Рис. 4.3.2. Объявление переменных ФБ-обработчика события

Метод **EventCallback**, полученный от интерфейса ICmpEventCallback, будет однократно вызван системой исполнения при наступлении интересующего нас события (для этого потребуется зарегистрировать callback – см. информацию на следующей странице). Вызывать этот метод (или сам экземпляр ФБ) в коде программы для этого необязательно (впрочем, вы можете это делать для каких-то своих целей – например, если блок реализует дополнительную логику).

Для возвращаемого методом значения сразу добавим в тип переменной пространства имен нужных библиотек – иначе вы получите ошибку при компиляции проекта. Код метода предельно прост – инкремент значения по указателю **pudiMyCustomEventCounter**.

MyCustomEventHandler.EventCallback

```
{attribute 'm4export_hide'}
{warning 'добавить реализацию метода'}
(* The interface ICmpEventCallback must be registered for an event. If this event is posted, this callback method is called.*)
METHOD EventCallback : CmpEventMgr.SysTypes.RTS_IEC_RESULT
VAR_INPUT
    (*Pointer to the event parameters, see Struct EventParam*)
    pEventParam : POINTER TO EventParam;
END_VAR
```

```
// В рамках примера - просто инкрементируем счетчик
pudiMyCustomEventCounter^ := pudiMyCustomEventCounter^ + 1;
```

Рис. 4.3.3. Код метода **EventCallback**

Теперь осталось написать код, в котором будет производиться регистрация экземпляра данного ФБ в качестве callback-объекта, а также создание и отправка события.

Переменные программы **PLC_PRG,** связанные с примером:

```
// --- Пример отправки и обработки пользовательского события ---
VAR
// Команда создания пользовательского события
xCreateMyCustomEvent:          BOOL := TRUE;
// Идентификатор пользовательского события
udiMyCustomEventId:            UDINT;
// Код ошибки
udiMyResult:                   UDINT;
// Команда регистрации обработчика события
xRegisterMyCustomEventHandler: BOOL := TRUE;
// Дескриптор события
hMyCustomEvent:                CmpEventMgr.SysTypes.RTS_IEC_HANDLE;
// Обработчик пользовательского события
fbMyCustomEventHandler:        MyCustomEventHandler;
// Счетчик пользовательских событий
udiMyCustomEventCounter:       UDINT;
// Команда отправки пользовательского события
xPostMyCustomEvent:            BOOL;
// Структура параметра пользовательского события
stMyCustomEventParam:          CmpEventMgr.CmpEventMgr_Interfaces.EventParam;
END_VAR
VAR CONSTANT
// Идентификатор нашего "компонента", являющегося отправителем
// пользовательского события
   c_udiMyCmpId:               UDINT := 55555;
END_VAR
```

Код создания события:

```
// --- Пример создания пользовательского события ---
IF xCreateMyCustomEvent THEN

   udiMyCustomEventId :=
      CmpEventMgr.CmpEventMgr_Implementation.EventCreateEventID(Event := 1,
     Class := CmpEventMgr.CmpEventMgr_Interfaces.EventClass.EVTCLASS_WARNING,
         Result := udiMyResult);

   CmpEventMgr.CmpEventMgr_Implementation.EventCreate2(EventId :=
      udiMyCustomEventId, CmpIdProvider := c_udiMyCmpId,
         nCallbacksPossible := 10, Result := udiMyResult);

       xCreateMyCustomEvent := FALSE;

END_IF
```

Сначала на основании кода события (**Event**) и класса события (**EVTCLASS_WARNING**) с помощью функции EventCreateEventID формируется идентификатор события (**udiMyCustomEventId**). Код и класс события в рамках примера выбраны произвольным образом. Далее с помощью функции EventCreate2 создается событие с данным идентификатором. В качестве идентификатора компонента в рамках примера использована константа **c_udiMyCmpId** (со значением **55555**). Идентификатор компонента выбирается разработчиком компонента, при этом он представляет собой значение типа **UDINT**, в котором старшее слово – это идентификатор

компании-разработчика (выдается CODESYS Group), а младшее – уникальный для данной компании идентификатор компонента, выбираемый произвольным образом. В рамках примера для пользовательского события установлено ограничение в 10 одновременно зарегистрированных callback-объектов (**nCallbacksPossible := 10**).

Код регистрации обработчика события (тут всё аналогично примеру из п. 4.2):

```
// --- Пример регистрации обработчика пользовательского события ---
IF xRegisterMyCustomEventHandler THEN

   fbMyCustomEventHandler.pudiMyCustomEventCounter :=
      ADR(udiMyCustomEventCounter);

   hMyCustomEvent :=
      CmpEventMgr.CmpEventMgr_Implementation.EventOpen(EventId :=
         udiMyCustomEventId, CmpIdProvider := c_udiMyCmpId,
            Result := udiMyResult);

   CmpEventMgr.CmpEventMgr_Implementation.EventRegisterCallback(hEvent :=
      hMyCustomEvent, pICallback := fbMyCustomEventHandler,
         Result := udiResult);

   xRegisterMyCustomEventHandler := FALSE;

END_IF
```

Код отправки события:

```
// --- Пример отправки пользовательского события ---
IF xPostMyCustomEvent THEN

   stMyCustomEventParam.EventId       := udiMyCustomEventId;
   stMyCustomEventParam.CmpIdProvider := c_udiMyCmpId;

   stMyCustomEventParam.usParamId := 1;
   stMyCustomEventParam.usVersion := 1;
   // В рамках примера не передаем вместе с событием никакого параметра
   // stMyCustomEventParam.pParameter := <...>

   udiMyResult := CmpEventMgr.CmpEventMgr_Implementation.EventPost(hEvent :=
      hMyCustomEvent, pEventParam := ADR(stMyCustomEventParam) );

   xPostMyCustomEvent := FALSE;

END_IF
```

Перед отправкой события необходимо заполнить поля экземпляра структуры его параметра. В рамках примера вместе с событием никакого параметра не отправляется (т. е. поле **pParameter** остается пустым); идентификатор (**usParamId**) и версия параметра (**usVersion**) выбраны произвольным образом. В реальном проекте вы можете передать вместе с вашим событием структуру данных, позволяющую предоставить компоненту-получателю дополнительную информацию о событии.

Отправка события осуществляется с помощью функции EventPost. Ее аргументами является дескриптор события и указатель на параметр события.

Проверим работу примера. Загрузите проект примера в контроллер. Запустите проект (**Отладка – Старт**). В визуализации нажимайте кнопку **Отправить** – каждое нажатие приведет к отправке пользовательского события, которое будет принято и обработано в методе **EventCallback** ФБ **MyCustomEventHandler**, что приведет к инкременту счетчика.

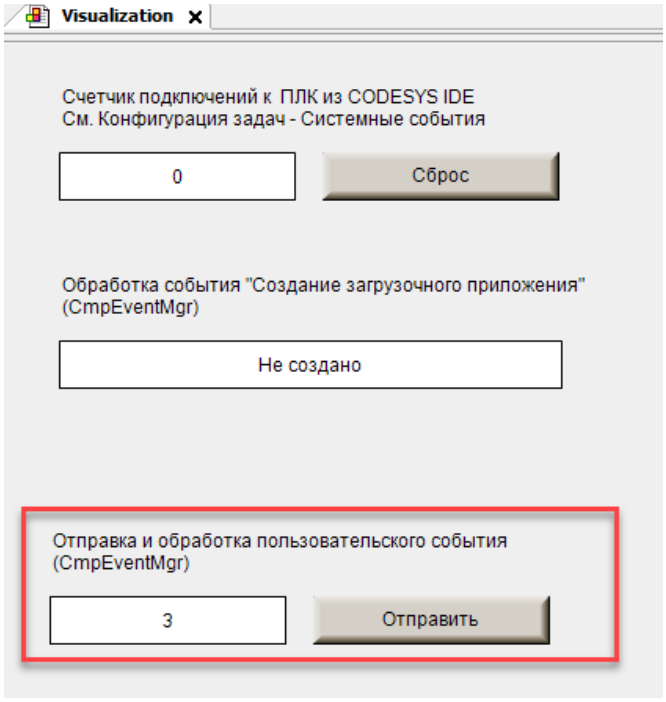


Рис. 4.3.4. Работа с пользовательским событием в примере

В рамках проекта:

- рассматривается синтетический пример, в котором создание, отправка и обработка события выполняется в рамках одной и той же программы **PLC_PRG**. В реальных проектах отправка и обработка событий выполняется в разных местах – например, событие отправляется из библиотеки, а обрабатывается в приложении пользователя или другой библиотеке (или наоборот).
- для всех функций используется одна и та же переменная для сохранения кода ошибки (**udiResult**). В реальных проектах рекомендуется использовать отдельные переменные для каждой функции и реализовать обработку ошибок.
- не демонстрируется дерегистрация callback-объекта (см. функцию EventUnregisterCallback и закрытие события (см. функцию EventClose2), а также удаление события (см. функцию EventDelete2).

5. Другие библиотеки

В рамках пункта рассматриваются другие библиотеки, которые напрямую или косвенно связаны с событиями.

5.1. Библиотека SysExcept. Обработка исключений

Исключение – это частный случай события, характеризующий ошибку в работе приложения, которая делает невозможным его дальнейшее выполнение. Примерами исключений является деление на ноль, детектирование бесконечного цикла, ошибка сегментации памяти (access violation) и т. д.

Обработка исключений осуществляется компонентом **SysExcept**. При возникновении исключения компонент формирует одно или два события класса **EVTCLASS_EXCEPTION**:

- **EVT_EXCPT_GenerateException** – отправляется при возникновении **любого** исключения;
- **EVT_EXCPT_GenerateException2** – отправляется при возникновении исключения, которого не было обработано с помощью механизма структурированной обработки исключений (**Structured Exception Handling** – **SEH**).

То есть при возникновении исключения будет отправлено либо только событие **EVT_EXCPT_GenerateException** (если исключение было обработано с помощью **SEH**), либо будут отправлены события **EVT_EXCPT_GenerateException** и **EVT_EXCPT_GenerateException2** (если исключение не было обработано с помощью **SEH**).

Структурированная обработка исключений в приложении CODESYS осуществляется с помощью операторов __TRY, __CATCH, __FINALLY, __ENDTRY.

Обработка событий об исключениях не отличается от обработки других событий.

В приложении CODESYS компонент **SysExcept** представлен одноименной библиотекой. В ее состав входят:

- RegContext – структура, описывающая контекст исключения (указатели на инструкцию, стековый кадр и т. д.). В основном предназначена для использования разработчиками компонентов;
- RtsExceptions – глобальный список констант, описывающих возможные исключения. Эти константы используются при структурированной обработке исключений и в качестве аргументов функции **SysExceptGenerateException**;
- SysExceptGenerateException – функция, позволяющая сгенерировать исключение. Аргументом функции является идентификатор конкретного исключения из списка глобальных констант **RtsExceptions**. Если функция успешно выполняется – то она ничего не возвращает (так как выполнение приложения прекращается), а во всех остальных случаях возвращает код ошибки **ERR_FAILED** из библиотеки CmpErrors.

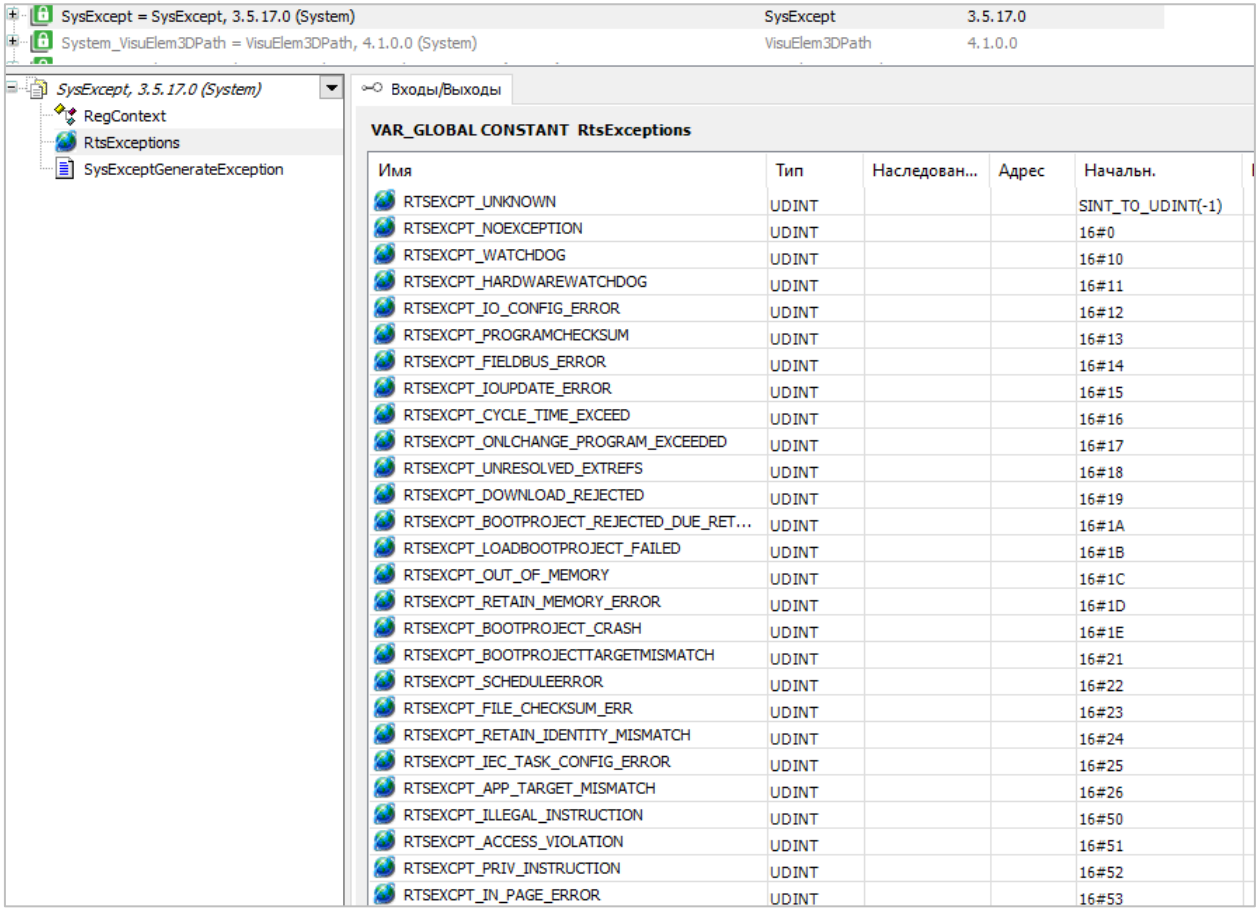


Рис. 5.1.1. Содержимое библиотеки **SysExcept**

5.2. Библиотека CAA Callback

Библиотека CAA Callback является альтернативой библиотеки **CmpEventMgr** и тоже используется для обработки событий. В целом, функционал библиотек достаточно близок; основными отличиями **CAA Callback** является существенно урезанный набор доступных системных событий (он приведен в перечисление Event) и отсутствие возможности регистрации функционального блока в качестве callback-объекта. Также набор функций библиотеки сокращен – например, для отправки события не требуется его открывать (впрочем, такой же функционал в библиотеке **CmpEventMgr** доступен с помощью функций EventPostByEvent и EventPostByEvent2) и отсутствует функция для закрытия события.

Соответствие между функциями библиотек **CmpEventMgr** и **CAA Callback** приведено в таблице ниже:

Табл. 5.2.1. Соответствие между функциями библиотек **CmpEventMgr** и **CAA Callback**

Функция библиотеки CmpEventMgr	Соответствующая функция библиотеки CAA Callback
EventPostByEvent	PostEvent, SendEvent
EventRegisterCallbackFunction	RegisterCallback
EventUnregisterCallbackFunction	UnregisterCallback
EventGetClass	DecodeClass
EventGetEvent	DecodeEvent
EventCreateEventID	EndodeSpec
EventRegisteredCallbacks	GetNumberActiveCallbacks

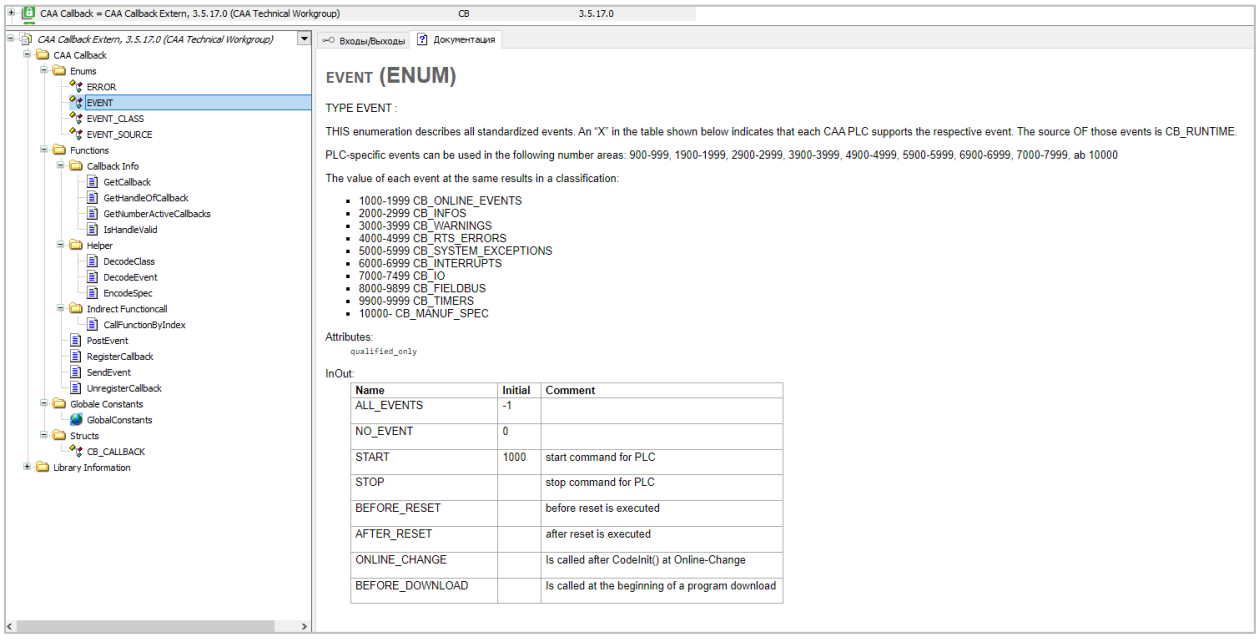


Рис. 5.2.1. Содержимое библиотеки **CAA Callback**

5.3. Библиотека SysEvent

Название библиотеки SysEvent может навести на мысль, что она как-то связана с темой документа. Но на самом деле это не так – данная библиотека используется для синхронизации выполнения двух задач приложения. Сначала в контексте каждой из задач с помощью функции **SysEventCreate** создается событие с названием **szEvent** (в каждой из задач должно использоваться одно и то же название события). Далее в нужный момент времени одна из задач «засыпает» и ждет события от другой задачи (за это отвечает функция **SysEventWait**). Другая задача в нужный момент времени отправляет событие (с помощью функции **SysEventSet**), что приводит к «пробуждению» первой задачи. В случае необходимости созданное событие можно удалить с помощью функции **SysEventDelete**.

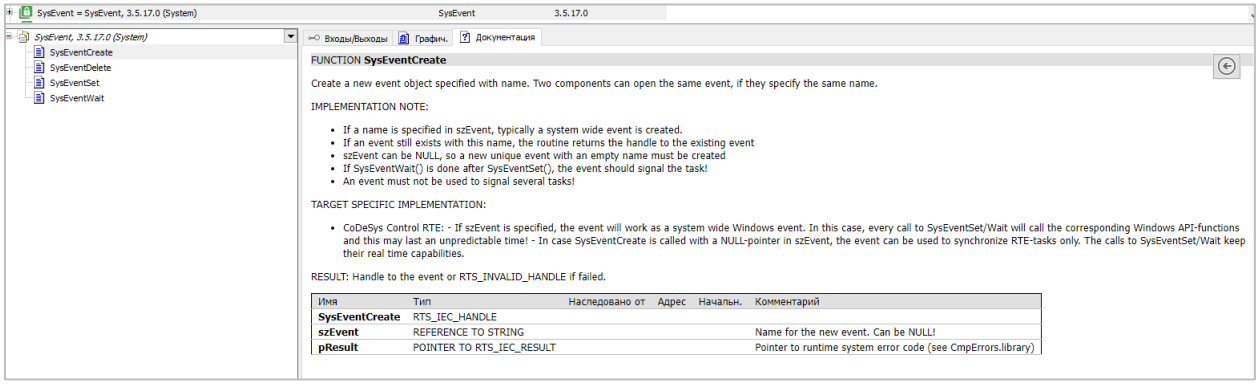


Рис. 5.3.1. Содержимое библиотеки **SysEvent**

Приложение А. Список событий системы исполнения CODESYS

Компонент	Событие	Класс события	Код события
CmpAlarmManager	EVT_AlarmManagerGetCount	EVTCLASS_INFO	1
	EVT_AlarmManagerGetAlarms	EVTCLASS_INFO	2
	EVT_AlarmManagerHandleService	EVTCLASS_INFO	3
CmpAppBP	EVT_CmpAppBP_CodePatch	EVTCLASS_INFO	1
	EVT_CmpAppBP_CodeSave	EVTCLASS_INFO	2
CmpAppForce	EVT_AppReleaseAllForceValues	EVTCLASS_INFO	1
CmpApp	EVT_PrepareStart	EVTCLASS_INFO	1
	EVT_StartDone	EVTCLASS_INFO	2
	EVT_PrepareStop	EVTCLASS_INFO	3
	EVT_StopDone	EVTCLASS_INFO	4
	EVT_PrepareReset	EVTCLASS_INFO	5
	EVT_ResetDone	EVTCLASS_INFO	6
	EVT_PrepareOnlineChange	EVTCLASS_INFO	7
	EVT_OnlineChangeDone	EVTCLASS_INFO	8
	EVT_PrepareDownload	EVTCLASS_INFO	9
	EVT_DownloadDone	EVTCLASS_INFO	10
	EVT_CodeInitDone	EVTCLASS_INFO	11
	EVT_PrepareDelete	EVTCLASS_INFO	12
	EVT_DeleteDone	EVTCLASS_INFO	13
	EVT_PrepareExit	EVTCLASS_INFO	14
	EVT_ExitDone	EVTCLASS_INFO	15
	EVT_CreateBootprojectDone	EVTCLASS_INFO	16
	EVT_DenyLoadBootproject	EVTCLASS_INFO	18
	EVT_PrepareLoadBootproject	EVTCLASS_INFO	19
	EVT_LoadBootprojectDone	EVTCLASS_INFO	17
	EVT_DenyStartBootproject	EVTCLASS_INFO	20
	EVT_PrepareStartBootproject	EVTCLASS_INFO	21
	EVT_StartBootprojectDone	EVTCLASS_INFO	22
	EVT_DenyStart	EVTCLASS_INFO	23
	EVT_DenyStop	EVTCLASS_INFO	24
	EVT_AllBootprojectsLoaded	EVTCLASS_INFO	25
	EVT_GlobalExitOnResetDone	EVTCLASS_INFO	26
	EVT_ExitDoneWithConfigAppInfo	EVTCLASS_INFO	27
	EVT_CmpApp_Exception	EVTCLASS_EXCEPTION	28
	EVT_RegisterBootproject	EVTCLASS_INFO	29
	EVT_CreateBootprojectFileFailed	EVTCLASS_ERROR	30
	EVT_CreateBootprojectFailed	EVTCLASS_ERROR	31
	EVT_OperatingStateChanged	EVTCLASS_INFO	32
	EVT_SourceDownload	EVTCLASS_INFO	33
	EVT_Login	EVTCLASS_INFO	34
	EVT_Logout	EVTCLASS_INFO	35
	EVT_DenyDelete	EVTCLASS_INFO	36
	EVT_DenyDeleteBootproject	EVTCLASS_INFO	37
	EVT_OEMDownloadServiceTag	EVTCLASS_INFO	39
	EVT_OEMRegisteredIecFunction	EVTCLASS_INFO	40
	EVT_POUTable_Changed	EVTCLASS_INFO	41
	EVT_CommCycle	EVTCLASS_INFO	42
	EVT_StateChanged	EVTCLASS_INFO	43
	EVT_OnlineChangeExecution	EVTCLASS_INFO	44
	EVT_PrepareCodeInit	EVTCLASS_INFO	45

	EVT_CDBlob	EVTCLASS_INFO	46
	EVT_RetainBackupState	EVTCLASS_INFO	47
	EVT_ResetAllApplications_Prepare	EVTCLASS_INFO	48
	EVT_ResetAllApplications_Done	EVTCLASS_INFO	49
	EVT_AllocArea	EVTCLASS_INFO	40
	EVT_DenyDownload	EVTCLASS_INFO	51
	EVT_DenyOnlineChange	EVTCLASS_INFO	52
	EVT_LoadBootprojectFailed	EVTCLASS_INFO	53
CmpChannelServer	EVT_ChSChannelClosed	EVTCLASS_INFO	1
	EVT_ChSChannelOpened	EVTCLASS_INFO	2
	EVT_ChSChannelClosedDone	EVTCLASS_INFO	3
CmpCodeMeter	EVT_CMPCODEMETER_LICENSEACTIVATED	EVTCLASS_INFO	1
CmpCoreDump	EVT_BeforeCreate	EVTCLASS_INFO	1
	EVT_AfterCreate	EVTCLASS_INFO	2
CmpDevice	EVT_CmpDevice_InteractiveLogin	EVTCLASS_INFO	1
	EVT_CmpDevice_ResetOrigin	EVTCLASS_INFO	2
	EVT_CmpDevice_SetOperationMode	EVTCLASS_INFO	3
	EVT_CmpDevice_ResetOriginGetConfig	EVTCLASS_INFO	4
	EVT_DevSBeforeCreateSession	EVTCLASS_INFO	5
	EVT_DevSSessionDeleted	EVTCLASS_INFO	6
CmpDeviceManagement	EVT_DEVICEMANAGEMENT	EVTCLASS_INFO	1
CmpEventMgr	EVT_EventCreate	EVTCLASS_INFO	1
	EVT_EventDelete	EVTCLASS_INFO	2
	EVT_EventOpen	EVTCLASS_INFO	3
	EVT_EventClose	EVTCLASS_INFO	4
	EVT_EventCommCycle	EVTCLASS_INFO	5
CmpFileTransfer	EVT_FileTransfer	EVTCLASS_INFO	1
	EVT_SyncFileTransfer	EVTCLASS_INFO	2
	EVT_GetReplaceFileName	EVTCLASS_INFO	3
CmpHilscherCIFX	EVT_DOWNLOAD_PROGRESS	EVTCLASS_INFO	1
	EVT_UPLOAD_PROGRESS	EVTCLASS_INFO	2
	EVT_PACKET_UNHANDLED	EVTCLASS_INFO	3
	EVT_PACKET_INDICATION	EVTCLASS_INFO	4
	EVT_CARDS_INIT_DONE	EVTCLASS_INFO	5
	EVT_PACKET_CONFIRMATION	EVTCLASS_INFO	6
	EVT_CIFX_INTERRUPTS_ENABLE	EVTCLASS_INFO	7
	EVT_CIFX_INTERRUPTS_DISABLE	EVTCLASS_INFO	8
	EVT_CIFX_CONFIGURE_DPM	EVTCLASS_INFO	9
	EVT_CIFX_GETFIRMWARE	EVTCLASS_INFO	10
	EVT_CIFX_LOADFIRMWARE	EVTCLASS_INFO	11
	EVT_CIFX_XCHANNEL_OPEN	EVTCLASS_INFO	12
	EVT_CIFX_XCHANNEL_CLOSE	EVTCLASS_INFO	13
	EVT_PACKET_INDICATION2	EVTCLASS_INFO	14
	EVT_PACKET_CONFIRMATION2	EVTCLASS_INFO	15
	EVT_PACKET_RECV	EVTCLASS_INFO	16
CmpIecTask	EVT_AfterReadingInputs	EVTCLASS_INFO	1
	EVT_BeforeWritingOutputs	EVTCLASS_INFO	2
	EVT_BeforeReadingInputs	EVTCLASS_INFO	3
	EVT_AfterWritingOutputs	EVTCLASS_INFO	4
	EVT_IecTask_ReloadInit	EVTCLASS_INFO	5
	EVT_IecTaskCreateDone	EVTCLASS_INFO	6
	EVT_IecTaskPrepareDelete	EVTCLASS_INFO	7
	EVT_IecTaskDebugLoop	EVTCLASS_INFO	8
CmpIecVarAccess	EVT_CmpIecVarAccess_PrepareWriteVariable	EVTCLASS_INFO	1
	EVT_CmpIecVarAccess_WriteVariableDone	EVTCLASS_INFO	2

	EVT_CmpIecVarAccess_SymbolsChanged	EVTCLASS_INFO	3
CmpIoDrvDPV1C1Slave	EVT_IoDrvDPV1_Recv	EVTCLASS_INFO	1
CmpIoMgr	EVT_PrepareUpdateConfiguration	EVTCLASS_INFO	1
	EVT_DoneUpdateConfiguration	EVTCLASS_INFO	2
	EVT_ConfigAppStartedDone	EVTCLASS_INFO	3
	EVT_PrepareConfigAppStopped	EVTCLASS_INFO	4
	EVT_PrepareUpdateMapping	EVTCLASS_INFO	5
	EVT_DoneUpdateMapping	EVTCLASS_INFO	6
	EVT_UpdateDiagDone	EVTCLASS_INFO	7
	EVT_ConfigAppStoppedDone	EVTCLASS_INFO	8
CmpLog	EVT_LogAdd	EVTCLASS_INFO	1
CmpMonitor2	EVT_CmpMonitor2_PrepareWrite	EVTCLASS_INFO	1
	EVT_CmpMonitor2_WriteDone	EVTCLASS_INFO	2
	EVT_CmpMonitor2_PrepareForce	EVTCLASS_INFO	3
	EVT_CmpMonitor2_ForceDone	EVTCLASS_INFO	4
CmpMonitor	EVT_PrepareWriteVariable	EVTCLASS_INFO	1
	EVT_WriteVariableDone	EVTCLASS_INFO	2
	EVT_PrepareForceVariable	EVTCLASS_INFO	3
	EVT_ForceVariableDone	EVTCLASS_INFO	4
CmpNameServiceServer	EVT_CmpNSSOemCallback	EVTCLASS_NONE	1
CmpNetXCanDlDrv	EVT_START_NETX_CONFIG	EVTCLASS_INFO	1
CmpOPCUAProvider	EVT_CmpOPCUAProviderHideIecVariable	EVTCLASS_INFO	1
CmpOPCUAServer	EVT_CmpOPCUAServerSessionsChanged	EVTCLASS_INFO	1
CmpPlcShell	EVT_PlcShellCommand	EVTCLASS_INFO	1
	EVT_PlcShellCommandRegister	EVTCLASS_INFO	2
CmpRedundancy	EVT_RedundancyDataSyncStart	EVTCLASS_INFO	2
	EVT_RedundancyDataSyncEnd	EVTCLASS_INFO	3
	EVT_RedundancySeamlessDownloadStarting	EVTCLASS_INFO	4
	EVT_RedundancySeamlessDownloadFinishing	EVTCLASS_INFO	5
	EVT_RedundancySeamlessDownloadSwitched	EVTCLASS_INFO	6
	EVT_RedundancySwitchState	EVTCLASS_INFO	7
CmpRetain	EVT_RetainGetSRAM	EVTCLASS_INFO	1
CmpSchedule	EVT_TaskCreateDone	EVTCLASS_INFO	1
	EVT_PrepareTaskDelete	EVTCLASS_INFO	2
	EVT_ExternalEventTaskCreateDone	EVTCLASS_INFO	3
	EVT_PrepareExternalEventTaskDelete	EVTCLASS_INFO	4
	EVT_ScheduleTick	EVTCLASS_INFO	5
	EVT_ScheduleTaskGap	EVTCLASS_INFO	6
	EVT_ScheduleTaskGapApp	EVTCLASS_INFO	7
CmpSecurityManager	EVT_CmpSecMan_ConfigurationChanged	EVTCLASS_INFO	1
CmpSettings	EVT_Settg_ReadSetting	EVTCLASS_INFO	1
	EVT_Settg_WriteSetting	EVTCLASS_INFO	2
	EVT_Settg_ReadSettings	EVTCLASS_INFO	3
CmpSupervisor	EVT_Supervisor_StateChanged	EVTCLASS_INFO	1
CmpTargetVisu	EVT_CmpTargetVisu_LoadImage	EVTCLASS_INFO	1
	EVT_CallbackContentPainted	EVTCLASS_INFO	2
	EVT_WindowCreated	EVTCLASS_INFO	3
	EVT_WindowDestroyed	EVTCLASS_INFO	4
	EVT_WindowFirstPaint	EVTCLASS_INFO	5
CmpTraceMgr	EVT_TRACEMGR_PACKET_CREATE	EVTCLASS_INFO	1
	EVT_TRACEMGR_PACKET_DELETE	EVTCLASS_INFO	2
	EVT_TRACEMGR_ADD_RECORD	EVTCLASS_INFO	3
	EVT_TRACEMGR_REMOVE_RECORD	EVTCLASS_INFO	4
	EVT_TRACEMGR_PACKET_COMPLETE	EVTCLASS_INFO	5
	EVT_TRACEMGR_PACKET_STATE_CHANGED	EVTCLASS_INFO	6

	EVT_TRACEMGR_UPDATE_RECORD	EVTCLASS_INFO	7
	EVT_TRACEMGR_PACKET_TRIGGER	EVTCLASS_INFO	8
	EVT_TRACEMGR_PACKET_SAMPLE	EVTCLASS_INFO	9
CmpUserMgr	EVT_UserMgrDatabaseChanged	EVTCLASS_INFO	1
	EVT_UserMgrCreateDefaultDatabase	EVTCLASS_INFO	2
	EVT_UserMgrConvertOldDatabaseAnnounce	EVTCLASS_INFO	3
	EVT_UserMgrConvertOldDatabaseFinish	EVTCLASS_INFO	4
CmpVisuHandler	EVT_CmpVisuHandler_ClientCreated	EVTCLASS_INFO	1
	EVT_CmpVisuHandler_ClientRemoved	EVTCLASS_INFO	2
CmpVisuHandler Remote	EVT_CmpVisuHandlerRemote_CheckAutoStartup	EVTCLASS_INFO	1
	EVT_CmpVisuHandlerRemote_VisuRunning	EVTCLASS_INFO	2
CmpVisuServer	EVT_VisuUserMgmtGetUserCount	EVTCLASS_INFO	1
	EVT_VisuUserMgmtGetUsers	EVTCLASS_INFO	2
	EVT_VisuUserMgmtCheckChangeUser	EVTCLASS_INFO	3
	EVT_VisuUserMgmtGetSetDB	EVTCLASS_INFO	4
CmpX509Cert	EVT_X509CertStoreChanged	EVTCLASS_INFO	1
CmpXMLParser	EVT_XMLStart	EVTCLASS_INFO	1
	EVT_XMLData	EVTCLASS_INFO	2
	EVT_XMLEnd	EVTCLASS_INFO	3
Component Manager	EVT_CmpMgr_PrepareShutdown	EVTCLASS_INFO	1
	EVT_CmpMgr_PrepareExitComm	EVTCLASS_INFO	2
	EVT_CmpMgr_PrepareExitTasks	EVTCLASS_INFO	3
	EVT_CmpMgr_Exit3	EVTCLASS_INFO	4
	EVT_CmpMgr_PrepareExit	EVTCLASS_INFO	5
	EVT_CmpMgr_Exit2	EVTCLASS_INFO	6
	EVT_CmpMgr_DisableOperation	EVTCLASS_INFO	7
	EVT_CmpMgr_PrepareExitCommProcessing	EVTCLASS_INFO	8
	EVT_CmpMgr_LicenseRequest	EVTCLASS_INFO	9
SysCom	EVT_SysComOpenBefore	EVTCLASS_INFO	1
	EVT_SysComOpenAfter	EVTCLASS_INFO	2
SysCpuMultiCore	EVT_GroupBinding	EVTCLASS_INFO	1
	EVT_IecCoreSetOverwrite	EVTCLASS_INFO	2
SysEthernet	EVT_EthPacketArrived	EVTCLASS_INFO	1
	EVT_EthPacketSent	EVTCLASS_INFO	2
	EVT_EthGetParameterValue	EVTCLASS_INFO	3
	EVT_EthSetParameterValue	EVTCLASS_INFO	4
SysExcept	EVT_EXCPT_GenerateException	EVTCLASS_EXCEPTION	1
	EVT_EXCPT_GenerateException2	EVTCLASS_EXCEPTION	2
SysFile	EVT_SysFileOpen	EVTCLASS_INFO	1
	EVT_SysFileClose	EVTCLASS_INFO	2
	EVT_SysFileFlashGetFileMapIndex	EVTCLASS_INFO	3
SysGraphic	EVT_SysGraphic_OpenKeyboard	EVTCLASS_INFO	1
	EVT_SysGraphic_CloseKeyboard	EVTCLASS_INFO	2
SysInt	EVT_INTERRUPT_0	EVTCLASS_INTERRUPT	0
	EVT_INTERRUPT_1	EVTCLASS_INTERRUPT	1
	EVT_INTERRUPT_2	EVTCLASS_INTERRUPT	2
	EVT_INTERRUPT_3	EVTCLASS_INTERRUPT	3
	EVT_INTERRUPT_4	EVTCLASS_INTERRUPT	4
	EVT_INTERRUPT_5	EVTCLASS_INTERRUPT	5
	EVT_INTERRUPT_6	EVTCLASS_INTERRUPT	6
	EVT_INTERRUPT_7	EVTCLASS_INTERRUPT	7
	EVT_INTERRUPT_8	EVTCLASS_INTERRUPT	8
	EVT_INTERRUPT_9	EVTCLASS_INTERRUPT	9
	EVT_INTERRUPT_10	EVTCLASS_INTERRUPT	10
	EVT_INTERRUPT_11	EVTCLASS_INTERRUPT	11

	EVT_INTERRUPT_12	EVTCLASS_INTERRUPT	12
	EVT_INTERRUPT_13	EVTCLASS_INTERRUPT	13
	EVT_INTERRUPT_14	EVTCLASS_INTERRUPT	14
	EVT_INTERRUPT_15	EVTCLASS_INTERRUPT	15
	EVT_INTERRUPT_255	EVTCLASS_INTERRUPT	255
SysNativeControl	EVT_SysNativeControlCreate	EVTCLASS_INFO	1
	EVT_SysNativeControlCreate2	EVTCLASS_INFO	2
	EVT_SysNativeControlDestroy	EVTCLASS_INFO	3
	EVT_SysNativeControlCallMethod	EVTCLASS_INFO	4
	EVT_SysNativeControlMove	EVTCLASS_INFO	5
	EVT_SysNativeControlShow	EVTCLASS_INFO	6
SysSocket	EVT_SysSocket_BeforeSetIpAndMask	EVTCLASS_INFO	1
	EVT_SysSocket_AfterSetIpAndMask	EVTCLASS_INFO	2
	EVT_SysSocket_BeforeSetGateway	EVTCLASS_INFO	3
	EVT_SysSocket_AfterSetGateway	EVTCLASS_INFO	4
	EVT_SysSocket_GetAdditionalAdapterInfo	EVTCLASS_INFO	5
SysTarget	EVT_SysTarget_SetNodeName	EVTCLASS_INFO	1
	EVT_SysTarget_SetNodeNameDone	EVTCLASS_INFO	2
	EVT_SysTarget_OverloadTargetIdent	EVTCLASS_INFO	3
SysTask	EVT_TaskCreate	EVTCLASS_INFO	1
	EVT_TaskDelete	EVTCLASS_INFO	2
	EVT_TaskSetInterval	EVTCLASS_INFO	3
	EVT_TaskGetPriority	EVTCLASS_INFO	4
	EVT_TaskLeave	EVTCLASS_INFO	5
SysWindow	EVT_SysWindow_OnInvokeInWindowTask	EVTCLASS_INFO	1
	EVT_SysWindow_BeforeCreate	EVTCLASS_INFO	2
	EVT_SysWindow_AfterCreate	EVTCLASS_INFO	3