CoDeSys Automation Alliance (CAA) Technik-Workshop

CAA_Callback.lib

Version 0.16

Bibliothekspräfix:	CB
Spezifikationsversion:	0
Bibliotheksart:	extern

Releasedatum:

Abhängigkeiten: CAA_Types.lib

Autoren:		
	Wolfgang Doll	3S Smart Software Solutions GmbH
	Hilmar Panzer	3S Smart Software Solutions GmbH

Änderungshistorie:

0.01	08.09.04	Erstellt für Präsentation zum 4. Workshop
0.02	10.09.04	Änderungen nach 4. Workshop
0.03	02.02.05	Änderungen nach 5. Workshop
0.04	16.02.05	Datentyp von Callback-Parameter: DWORD
0.05	11.05.05	identische Callbacks; Abarbeitungsreihenfolge; Callbackreihenfolge
0.06	13.05.05	CAA_Handle Callback Funktionen müssen mit dem Präfix Callback beginnen.
0.07	18.05.05	Erklärung Abarbeitungsreihenfolge
0.08	20.05.05	Erweitert um CB_CallFunctionByIndex
0.09	20.05.05	CB_ERR_POWERFAIL; Rückgabewert ; Erklärung zum Aufruf
0.10	30.06.05	Abhängigkeit zu CAA_Types.lib
0.11	01.07.05	eSource bei CB_PostEvent hinzugefügt
0.12	04.07.05	CB_FIELDBUS Ergänzt in CB_EventClass
0.13	05.07.05	CB_gc_cbNULL, CB_DecodeClass, CB_DecodeSource, CB_EncodeSpec ergänzt. Parameter, Beschreibung und Beispiel von CB_PostEvent erweitert.
0.14	08.07.05	Äderungen nach 9. TWS Treffen: CB_gc_cbNULL, Returnvalues benannt, CB_Event, CB_EventClasses, CB_EventSource
0.15	19.09.05	Anmerkungen zu CoDeSys SP 3.0 Alle Funktionen haben min. einen Input Änderung Signatur CB_GetCallback Änderung Signatur der CallbackXXX Funktionen Löschen CB_DecodeSource Einführen CB_DecodeEvent ENUM Typen haben nur noch INT Werte
0.16	26.01.06	Timer-Events hinzugefügt
0.17	10.02.06	CB_EventsClass CB_Timer durch CB_Timers ersetzt.

1. Inhalt

1 Übersicht

Im Laufzeitsystem treten verschiedene Ereignisse auf. Jedes Ereignis kann einer Ereignis-Klasse zugeordnet werden. Jedes Ereignis stammt außerdem von einer gewissen Quelle. Das Laufzeitsystem verfügt intern über eine Liste registrierter Callback-Definitionen. Bei jedem auftretenden Ereignis prüft das Laufzeitsystem seine Liste von Callback-Definitionen und ruft ggfs. eine Callback-Funktion auf.

Eine Callback-Definition besteht aus:
- Ereignis
- Ereignis-Klasse
- Ereignis-Quelle
- Index der Funktion, die bei Eintreten des Ereignisses aufgerufen werden soll.

Das Ereignis e der Klasse c aus der Quelle s löst den Callback (e_c, c_c, s_c) aus, wenn

```
(e = e_c OR e_c = CB_ALL_EVENTS) AND
((c AND c_c) > 0 OR c_c = CB_ALL_CLASSES) AND
(s = s_c OR s_c = CB_ALL_SOURCES)
```

Eine Funktion, die als Callback-Funktion verwendet werden soll, muss folgende Schnittstelle haben:

```
FUNCTION Callback<Name> : BOOL
VAR_INPUT
  dwSpec  : DWORD; (* CB_Event and CB_EventClass *)
  dwSource: DWORD; (* CB_EventSource *)
  dwParam : DWORD; (* Application specific Parameter *)
END_VAR
```

Die Datentypen sind aus Kompatibilitätsgründen zur V2.3 DWORDs. Hinter ihnen verbergen sich allerdings die in Klammern angegebenen Enumerationen. Der Parameter dwSpec kann über die Funktion CB_DecodeEvent und CB_DecodeClass in zwei Variablen vom Type CB_Event und CB_EventClass zerlegt werden. Der Übergabearameter dwSource kann mithilfe der Konvertierung DWORD_TO_INT in einen Datentyp CB_EventSource verwandelt werden.

Bemerkungen zu CoDeSys SP 3.0:
In der aktuellen Version von CoDeSys SP 2.4 sind Ereignisse als eine statische Menge von Event Nummern (im Enum CB_Event) definiert.
Ab CoDeSys SP 3.0 werden Events von den jeweiligen Komponenten dynamisch angelegt. Aus diesem Grund wird es zwei weitere Funktionen (CB_EventCreate und CB_EventDelete) geben mit denen diese Aufgabe verwirklicht werden kann. Dadurch muss der Aufbau und die Interpretation des CB_Callback Datentyp entsprechend angepasst werden. Jedem Event ist über den Komponentennamen eindeutig eine Komponente zugeordnet. In der jeweiligen Callback-Funktion enthält der Parameter dwSource dann ein Handle mit dem Informationen über die Komponente erfragt werden können, die das Ereignis ausgelöst hat. In dwParam wird dann ein Pointer auf eine Struktur hinterlegt. Der standardisierte Aufbau dieser Struktur ermöglicht eine Versionsprüfung, die sicherstellt das für die Auslöser-Komponente und die jeweilige Behandlungsroutine eine identische Interpretation der Parameter gegeben ist. Aus diesen Gründen wird es ab CoDeSys SP 3.0 eine neue Bibliothek mit dem Namen CAA_CallbackEx.lib geben, die diese neuen Anforderungen erfüllt.

Bemerkungen zu Aufbau und Verwendung der Callback-Funktionen:

Der Funktionsname **muss** mit dem Präfix Callback versehen sein.

Die Funktion darf **keine** lokalen Variablen aufweisen.

Die Funktion darf **nicht** mit Breakpoints, ... getestet werden.

Auf einen Event können mehrere Callbacks eingehängt werden. Ein Callback (Ereignis, -Klasse, -Quelle, Index der Funktion) kann jedoch nur einmal registriert werden; wird versucht, einen identischen Callback einzuhängen, liefert CB_RegisterCallback einen Fehler zurück.

Alle registrierten Callbacks werden unmittelbar nach einem Steuerungsreset automatisch deregistriert. Eventuell auf das Ereignis CB_AFTER_RESET registrierte Callbacks werden vor der automatischen Deregistrierung genau noch einmal aufgerufen.

Man beachte, dass Callback-Funktionen direkt nach dem Eintreten des Ereignisses aufgerufen werden. Dabei kann eine zu diesem Zeitpunkt laufende IEC-Task unterbrochen werden. Dem Applikateur muss bewusst sein, dass sich dabei ähnliche Problematiken wie bei Multitasking-Systemen ergeben (Datenkonsistenz etc.); außerdem sollte die Laufzeit einer Callback-Funktion so kurz wie möglich gehalten werden, da dadurch das gesamte System blockiert wird.

Der Rückgabewert der Callback-Funktion hat keine Bedeutung[1].

1.1 Abarbeitungsreihenfolge

Die Abarbeitung von Callbacks, die vom selben Event ausgelöst werden, erfolgt in der umgekehrten Definitions-Reihenfolge.[2]

1.2 Auslösereihenfolge

Manche Ereignisse lösen eine Kette von Events aus. Diese werden in folgender, fester Reihenfolge aufgerufen:

Steuerungsreset	CB_STOP (nur, wenn Steuerung läuft) CB_BEFORE_RESET CB_AFTER_RESET
Shutdown	CB_SHUTDOWN CB_STOP (nur, wenn Steuerung läuft)
OnlineChange	CB_ONLINE_CHANGE
Programm-Download	CB_STOP (nur, wenn Steuerung läuft) CB_BEFORE_DOWNLOAD

[1] Zur Erklärung: Hätte der Rückgabewert eine Bedeutung, so wäre bei mehreren, auf demselben Event registrierten Callback-Funktionen unklar, welcher Rückgabewert dominant ist.

[2] Grund hierfür ist, dass Systemkomponenten denselben Mechanismus benutzen und i.d.R. vor der Applikation Callbacks registrieren. Häufig tritt der Fall auf, dass eine Systemkomponente auf das Ereignis RESET hin Speicher freigibt. Hat die Applikation ebenso einen Callback auf dasselbe Ereignis definiert, könnte der Fall auftreten, dass die Applikation versucht, auf freigegebenem Speicher zu operieren.

2 Datentypen/Enumerationen

2.1 CB_Event (ENUM)

Diese Enumeration beschreibt alle standardisierten Events. In der rechten Spalte ist markiert, ob jede CAA-Steuerung dieses Event unterstützen muss.

Steuerungsspezifische Events können in folgenden Nummer-Bereichen verwendet werden:
900-999, 1900-1999, 2900-2999, 3900-3999, 4900-4999, 5900-5999, 6900-6999, 7000-7999, ab 10000

Der Wert jedes Events führt gleichzeitig zu einer Klasseneinteilung:

1000-1999	CB_ONLINE_EVENTS
2000-2999	CB_INFOS
3000-3999	CB_WARNINGS
4000-4999	CB_RTS_ERRORS
5000-5999	CB_SYSTEM_EXCEPTIONS
6000-6999	CB_INTERRUPTS
7000-7499	CB_IO
8000-9899	CB_FIELDBUS
9900-9999	CB_TIMERS
ab 10000	CB_MANUF_SPEC

CB_NO_EVENT	0		X
CB_ALL_EVENTS	-1		X
CB_START	1000	start	X
CB_STOP	1001	stop	X
CB_BEFORE_RESET	1002	vor reset	X
CB_AFTER_RESET	1003	nach reset	X
CB_SHUTDOWN	1004	shutdown	X
CB_ONLINE_CHANGE	1005	nach CodeInit bei Online Change	
CB_BEFORE_DOWNLOAD	1006	vor Programm-Download	X
CB_TASKCODE_NOT_CALLED	1007	Taskcode wird nicht aufgerufen	
CB_TIMER	1008	Scheduler-Tick (nicht bei stop)	
CB_DEBUG_LOOP	1009	Kommunikations-Takt-Tick (wenn Steuerung auf Breakpoint steht)	X
CB_SCHEDULE	1010	Scheduler-Tick (auch bei stop)	
CB_ERR_WATCHDOG	4000	Software watchdog	
CB_ERR_HARDWARE_WATCHDOG	4001	Hardware watchdog	
CB_ERR_FIELDBUS	4002	Feldbus-Fehler	X
CB_ERR_IOUPDATE	4003	E/A-Fehler	X
CB_ERR_POWERFAIL	4004	PowerFailure	
CB_EXCPT_ILLEGAL_INSTRUCTION	5000	ungültige Anweisung	
CB_EXCPT_ACCESS_VIOLATION	5001	Zugriffsverletzung	
CB_EXCPT_PRIV_INSTRUCTION	5002	Priv. Anweisung	
CB_EXCPT_IN_PAGE_ERROR	5003	Zugriffsverletzung	
CB_EXCPT_STACK_OVERFLOW	5004	Stack-Überlauf	
CB_EXCPT_MISALIGNMENT	5005	Misalignment	
CB_EXCPT_ARRAYBOUNDS	5006	Array-Grenzen	
CB_EXCPT_DIVIDEBYZERO	5007	Division durch Null	
CB_EXCPT_OVERFLOW	5008	Überlauf	

CB_EXCPT_NONCONTINUABLE	5009	nicht fortsetzbare Ausnahme	
CB_EXCPT_NO_FPU_AVAILABLE	5500	keine FPU vorhanden	
CB_EXCPT_FPU_ERROR	5501	FPU-Fehler	
CB_EXCPT_FPU_DENORMAL_OPERAND	5502	Unzulässiger Operand	
CB_EXCPT_FPU_DIVIDEBYZERO	5503	Division durch Null	
CB_EXCPT_FPU_INVALID_OPERATION	5504	unzulässige Operation	
CB_EXCPT_FPU_OVERFLOW	5505	Überlauf	
CB_EXCPT_FPU_STACK_CHECK	5506	FPU spezifische Ausnahme	
CB_INTERRUPT_0	6000		
CB_INTERRUPT_1	6001		
CB_INTERRUPT_2	6002		
CB_INTERRUPT_3	6003		
CB_INTERRUPT_4	6004		
CB_INTERRUPT_5	6005		
CB_INTERRUPT_6	6006		
CB_INTERRUPT_7	6007		
CB_INTERRUPT_8	6008		
CB_INTERRUPT_9	6009		
CB_INTERRUPT_10	6010		
CB_INTERRUPT_11	6011		
CB_INTERRUPT_12	6012		
CB_INTERRUPT_13	6013		
CB_INTERRUPT_14	6014		
CB_INTERRUPT_15	6015		
CB_INTERRUPT_255	6255		
CB_AFTER_READING_INPUTS	7000	nach dem Lesen der Eingänge	
CB_BEFORE_WRITING_OUTPUTS	7001	vor dem Schreiben der Ausgänge	
	8000-8099	reserviert für CAN-Bibliothek	
	8100-8199	reserviert für Profibus-Bibliothek	
CB_TIMER_1	9900-9999	Reserviert für CAA_Timer.lib	

2.2 CB_EventClass (ENUM)

Diese Enumeration beschreibt die Klassen, in welche die Events eingeteilt sind.

CB_ALL_CLASSES	-1	
CB_NO_CLASS	0	
CB_ONLINE_EVENTS	16#0001	Allgemeine Online-Events
CB_INFOS	16#0002	Info-Events
CB_WARNINGS	16#0004	Warnungen
CB_RTS_ERRORS	16#0008	Fehler des LZSs
CB_SYSTEM_EXCEPTIONS	16#0010	Ausnahmefehler des Systems
CB_INTERRUPTS	16#0020	Interrupts
CB_IO	16#0040	E/A-Ereignisse
CB_FIELDBUS	16#0080	Feldbus-Ereignisse
CB_TIMERS	16#0100	Timer-Ereignis
CB_MANUF_SPEC	16#0200	steuerungsspezifische Ereignisse

2.3 CB_EventSource (ENUM)

Diese Enumeration beschreibt die Auslöser eines Ereignisses.

CB_ALL_SOURCES	-1	
CB_NO_SOURCE	0	
CB_RUNTIME	16#0001	Laufzeitsystem
CB_SYSTEM	16#0002	(Betriebs)system
CB_IECTASK	16#0004	Taskverwaltung
CB_IECPROGRAM	16#0008	IEC-Programm
CB_DRIVER	16#0010	Treiber

2.4 CB_Errors (ENUM)

Diese Datenstruktur beschreibt Fehler, die im Umgang mit den Funktionen dieser Bibliothek auftreten können

CB_NO_ERROR	0	kein Fehler
CB_HANDLE_INVALID	1	handle ungültig
CB_UNKNOWN_EVENT	2	Event unbekannt
CB_CALLBACK_NOT_REMOVABLE	3	Callback kann nicht deaktiviert werden
CB_WRONG_ARGUMENT	4	Es wurde z.B. ein Null-Pointer überg.
CB_MF_SPEC	16#7FFF	Hersteller-spezifischer Fehler

2.5 CB_CALLBACK (STRUCT; Präfix cb)

Diese Datenstruktur beschreibt die Callback-Bedingungen und –Funktion.

iPOUIndex	INT	INDEX der aufzurufenden Funktion
eEvent	CB_Event	Auslöse-Ereignis oder CB_ALL_EVENTS
eClass	CB_EventClass	Ereignisklasse (Maske)
eSource	CB_EventSource	Ereignisquelle

2.6 CB_gc_cbNull (VAR GLOBAL CONSTANT)

Diese Konstante kann zur Initialisierung von nicht verwendeten Eingängen des Typs CB_CALLBACK verwendet werden.

CB_gc_cbNull	CB_CALLBACK	iPOUIndex := -1;
		eEvent := CB_NO_EVENT;
		eClass := CB_NO_CLASS;
		eSource := CB_NO_SOURCE;

3 Bausteine

3.1 CB_RegisterCallback (FUN)

Mit dieser Funktion können neue Callbacks aktiviert werden.

Input:

cbNew	CB_Callback	Beschreibung des Callbacks

Output:

hCallback	CAA_HANDLE	Handle des neuen Callbacks; 0 wenn Fehler auftrat.

3.2 CB_UnregisterCallback (FUN)

Mit dieser Funktion können Callbacks gelöscht werden.

Input:

hHandle	CAA_HANDLE	Handle des Callbacks

Output:

eError	CB_Errors	Fehlerbeschreibung

3.3 CB_GetCallback (FUN)

Mit dieser Funktion kann ein Callback über sein Handle geholt werden.

Input:

hHandle	CAA_HANDLE	Handle des Callbacks
pCallback	POINTER TO CB_CALLBACK	Rückgabe: Beschreibung der Callback-Bedingungen und der –Funktion

Output:

eError	CB_Errors	Fehlerbeschreibung

3.4 CB_IsHandleValid (FUN)

Mit dieser Funktion kann geprüft werden, ob ein Handle eines Callbacks gültig ist.

Input:

hHandle	CAA_HANDLE	Handle auf Callback

Output:

xValid	BOOL	TRUE: Handle gültig; FALSE: Handle ungültig

3.5 CB_GetNumberActiveCallbacks (FUN)

Diese Funktion liefert die Anzahl aktiver Callbacks.

Input:

xDummy	BOOL	

Output:

uiNumCallbacks	UINT	Anzahl aktiver Callbacks

3.6 CB_GetHandleOfCallback (FUN)

Mit dieser Funktion kann ein Callback über sein Handle geholt werden.

Input:

uiNumber	UINT	Nummer des Callbacks (zw. 1 und CB_GetNumberActiveCallbacks)

Output:

hCallback	CAA_HANDLE	Handle des Callbacks

3.7 CB_PostEvent (FUN)

Mit dieser Funktion kann ein Ereignis simuliert werden, um damit ein/mehrere Callbacks auszulösen. Es wird empfohlen, dass IEC-Programme, die diese Funktion verwenden, als Event-Quelle CB_IECPROGRAM und als Event-Klasse CB_ALL_CLASSES setzen. Wird die Funktion von Treibern verwendet empfiehlt sich als Event-Quelle der Wert CB_DRIVER.

Input:

eEvent	CB_EVENT	Event
eClass	CB_EventClass	Ereignisklasse
eSource	CB_EventSource	Ereignisquelle
dwParam	DWORD	Parameter für den Empfänger des Event

Output:

eError	CB_Errors	Fehlerbeschreibung

3.8 CB_CallFunctionByIndex (FUN)

Mit dieser Funktion kann eine IEC-Funktion indirekt über ihren POU-Index aufgerufen werden. Folgende Randbedingungen sind einzuhalten:

- Der Name der Funktion muss mit dem Präfix „Callback“ versehen sein.
- Die Funktion muss einen Rückgabewert mit einem Type der Länge 4 Byte aufweisen.
- Die Funktion muss genau drei Parameter mit einem Type der Länge 4 Byte aufweisen.

Input:

iPOUIndex	INT	POU-Index der aufzurufenden Funktion
dwParam1	DWORD	Erster Parameter der aufzurufenden Funktion
dwParam2	DWORD	Zweiter Parameter der aufzurufenden Funktion
dwParam3	DWORD	Dritter Parameter der aufzurufenden Funktion

Output:

dwValue	DWORD	Rückgabe der aufgerufenen Funktion

3.9 CB_DecodeEvent (FUN)

Mit dieser Funktion kann aus dem Parameter dwSpec der Callback-Funktion der Wert eEvent vom Type CB_Event extrahiert werden.

Input:

dwSpec	DWORD	Parameter der Callback-Funktion

Output:

eEvent	CB_Event	Eventbezeichnung für den Callback

3.10 CB_DecodeClass (FUN)

Mit dieser Funktion kann aus dem Parameter dwSpec der Callback-Funktion der Wert eClass vom Type CB_EventClass extrahiert werden.

Input:

dwSpec	DWORD	Parameter der Callback-Funktion

Output:

eClass	CB_EventClass	Ereignis Klasse des Callbacks

3.11 CB_EncodeSpec (FUN)

Mit dieser Funktion kann aus zwei Werten vom Typ CB_EventClass und CB_EventSource, der für Callback-Funktionen notwendige Parameter dwSpec vom Typ DWORD erzeugt werden.

Input:

eEvent	CB_EventEvent	Parameter der Callback-Funktion
eClass	CB_EventClass	Parameter der Callback-Funktion

Output:

dwSpec	DWORD	Parameter der Callback-Funktion

4 Beispiele

4.1 Callbacks einhängen

Diese Programmzeilen hängen zwei Callbacks ein. Den ersten auf die Funktion BeforeResetFunction, die aufgerufen wird, bevor die Steuerung geresetted wird; der zweite ruft die Funktion ErrorFunction auf, sobald ein Ereignis der Klasse CB_RTS_ERRORS oder CB_SYSTEM_EXCEPTIONS auftritt.

```
cbNew.eEvent := CB_BEFORE_RESET;
cbNew.eClass := CB_ALL_CLASSES;
cbNew.eSource := CB_ALL_SOURCES;
cbNew.iPOUIndex := INDEXOF(CallbackBeforeReset);
CB_RegisterCallback(cbNew);

cbNew.eEvent := CB_ALL_EVENTS;
cbNew.eClass := CB_RTS_ERRORS OR CB_SYSTEM_EXCEPTIONS;
cbNew.eSource := CB_ALL_SOURCES;
cbNew.iPOUIndex := INDEXOF(CAllbackError);
CB_RegisterCallback(cbNew);
```

4.2 Callbacks löschen

4.2.1 Handle bekannt

Dieses Beispiel setzt voraus, dass beim Registrieren des Callbacks das zurückgelieferte Handle gespeichert wurde:

```
(* Callback definieren *)
cbNew.eEvent := CB_BEFORE_RESET;
cbNew.eClass := CB_ALL_CLASSES;
cbNew.eSource := CB_ALL_SOURCES;
cbNew.iPOUIndex := INDEXOF(CallbackBeforeReset);

(* Callback einhängen und Handle merken*)
h := CB_RegisterCallback(cbNew);


(* Callback löschen *)
IF CB_IsHandleValid(h) THEN
  CB_UnregisterCallback(h);
END_IF
```

4.2.2 Handle unbekannt

Sollte das Handle nicht mehr bekannt sein, dann zeigt folgendes Beispiel wie ein Callback gelöscht werden kann:

```
(* Callback definieren *)
cbDelete.eEvent := CB_BEFORE_RESET;
cbDelete.eClass := CB_ALL_CLASSES;
cbDelete.eSource := CB_ALL_SOURCES;
cbDelete.iPOUIndex := INDEXOF(CAllbackBeforeReset);

(* alle Callbacks durchsuchen *)
```



```
FOR i:=1 TO CB_GetNumberActiveCallbacks() DO
  (* Handle holen *)
  h := CB_GetHandleOfCallback(i);

  (* wenn gesuchter Callback *)
  IF CB_GetCallback(h)=cbDelete THEN
       (* Callback löschen *)
       CB_UnregisterCallback(h);
       EXIT;
  END_IF
END_FOR
```

4.3 Callbacks auslösen

Diese Zeile simuliert ein Ereignis, worauf das System darauf registrierte Callbacks auslöst:

```
CB_PostEvent
(
    eEvent  := CB_EXCPT_OVERFLOW,
    eClass  := CB_ALL_CLASSES,
    eSource := CB_IECPROGRAM,
    dwParam := 0
);
```

4.4 Indirekter Funktionsaufruf

1. Implementierung einer Funktion:

```
FUNCTION CallbackTestFunction : DWORD
      VAR_INPUT
            tTime    : TIME;
            udiCount : UDINT;
            dwState  : DWORD;
      END_VAR
```

2. Indirekter Aufruf dieser Funktion:

```
IF xCallTest THEN
      xCallTest := FALSE;
      dwValue := CB_CallFunctionByIndex
            (
                  iPOUIndex := INDEXOF(CallbackTestFunction),
                  dwParam1  := TIME_TO_DWORD(TIME()),
                  dwParam2  := UDINT_TO_DWORD(udiCount),
                  dwParam3  := dwState
            );
END_IF
```

