Beschreibung AddOn Packages

ab SoftwareVersion 2.3.8

Inhaltsverzeichnis

1. Allgemeines

Mit den AddOns (Packages) können Sie selbst die Funktionalität von Cockpit erweitern.
Möglich ist fast alles.
Grundkenntnisse in der Programmierung müssen/sollten vorhanden sein.

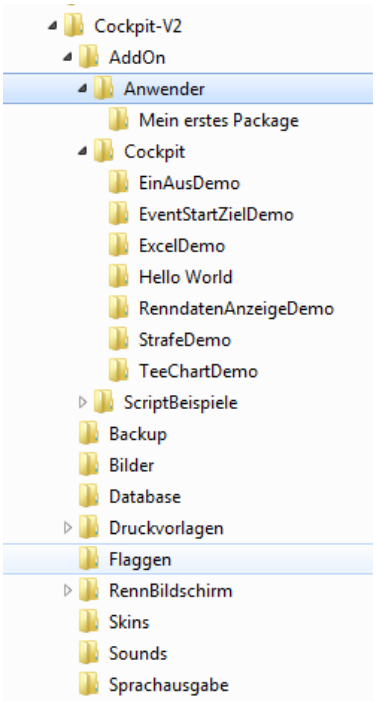
Prinzip: Jedes Package kann eine oder mehrere neue Funktionalitäten übernehmen. Diese Packages können dann über die Optionsseite „AddOn Aktivierung“ beliebig aktiviert bzw. auch deaktiviert werden.

In jedem Package kann man auf die verschiedenste Ereignisse wie z.B. Überfahrt Start/Ziel, Neuer Rundenrekord usw. ein Pascal-Script programmieren, dass dann die gewünschte Funktionalität ausführt.

Was kann man machen:

- Ein- und Ausblenden von Bildschirmen mit Variablen
- Zugriff auf die internen Renndaten wie Rundenzeit,TopSpeed usw.
- Setzen von Ausgängen und Abfragen von Eingänge
- In Dateien schreiben / lesen
- Zugriff auf Excel / Word (alle Ole Container möglich)
- TeeChart für grafische Ausgaben (ohne Excel)
- Vergeben von Strafen
- Sprachausgabe und WAV-Datei abspielen lassen
- Geschwindigkeit- und Bremswert abfragen/setzen (CU30352)

2. Aufbau eines Package



Jedes Packages ist wie die Rennbildschirme in verschiedene Ordner untergebracht.

Hier gibt es allerdings noch den Unterschied, dass alle Beispiele von mir in dem Ordner „Cockpit“ abgelegt werden.

Eigene Packages werden in dem Ordner „Anwender“ abgelegt.

Der Haupt-Ordner „AddOn“ befindet sich auf der gleichen Ebene wie der Ordner „RennBildschirm“

Wichtig: Bei jedem Update wird der Ordner „Cockpit“ gelöscht und wieder mit dem neusten Stand aktualisiert.
Hier also keine Änderungen durchführen !

Im ConfigCenter sieht das dann so aus:

Ein einfaches Package kann nur aus der Datei: „AddOn.cps.xml“ bestehen. Diese Datei ist zwingend notwendig.
In dieser Datei wird die komplette Programmierung der einzelnen Ereignisse abgespeichert.
Hinweis: Diese Datei nicht über einen externen Editor bearbeiten sondern nur mit dem Ereignis-Editor in Cockpit !

ConfigPackage.xml
In dieser Datei befindet sich
Package Variablen mit Datentyp und Vorbesetzungswert die über die Oberfläche eingegeben werden können.

```
<Variable xmlindex="1" beschreibung="Dies ist die Testvariable 1 Integer" name="Test1" datatyp="Integer" value="123"></Variable>
<Variable xmlindex="2" beschreibung="Dies ist die Testvariable 2 String" name="Test2" datatyp="String" value="Dies ist String"></Variable>
<Variable xmlindex="3" beschreibung="Dies ist die Testvariable 3 Flaot" name="Test3" datatyp="Float" value="123,245"></Variable>
```

xmlindex: Muss eindeutig sein und bestimmt die Reihenfolge der Eingabevariablen. Startet mit 1
beschreibung: Beschreibungstext der in der Oberfläche angezeigt wird
datatyp: Datentyp der Variable: Integer oder String oder Float
value: Vorbesetzungswert

a) Name für einen digitalen Ausgang bzw. Eingang definieren
Über diesen Namen kann ein Eingang abgefragt bzw. ein Ausgang gesetzt werden.

```
<DigInput name="Eingang1" devicenumber="1" number="1"></DigInput>
<DigOutput name="Ausgang1" devicenumber="1" number="1"></DigOutput>
```

name: Name des Ein- bzw. Ausganges

devicenumber: Gerätenummer der Box (sieht man im Gerätemanager von der Rennbahn unter Gerätenummer)
number: Nummer des Ein- bzw. Ausganges

Diese Datei kann direkt im ConfigCenter mit einem Doppelklick geöffnet und bearbeitet werden.

Alle weiteren XML-Dateien in einem Package sind dann Bildschirme die analog zu einem Rennbildschirm aufgebaut sind.
Diese Bildschirme werden auch mit dem normalen Designer bearbeitet.

Der Name von diesen Bildschirmen kann beliebig gewählt werden.

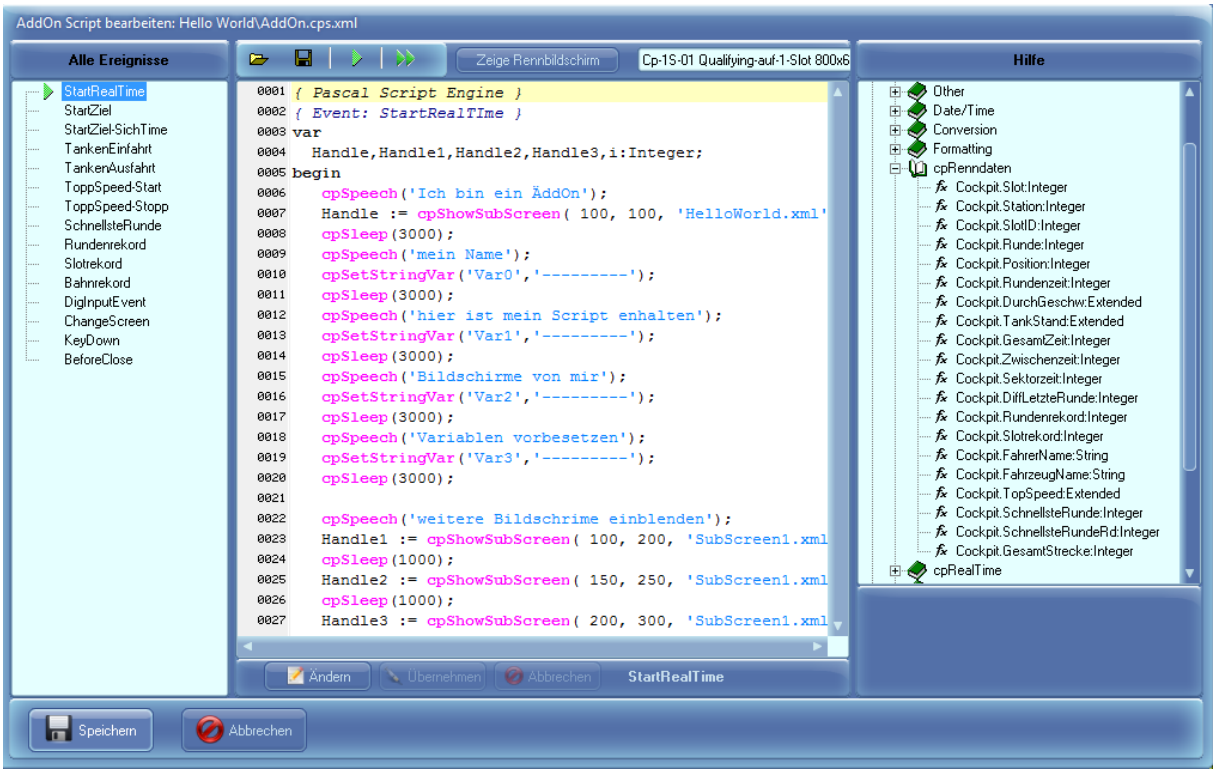
Über ein Kontextmenü (rechte Maustaste auf den Namen des AddOn) kann das AddOn aktiviert bzw. auch deaktiviert werden.
Ein Deaktiviertes AddOn ist in der AddOn Aktivierungsseite nicht sichtbar.

3. Programmiersprache

Die Ereignisse werden in Pascal Script programmiert.
Denke im WWW findet man genügend Beispiele zu dieser Sprache
Es wird nicht der komplette Umfang von Pascal Befehlen unterstützt.
Die meisten Befehle stehen in der Hilfe. (Aber auch nicht alle)

Weitere Beispiel von mir im Ordner: AddOn_Demos

4. Ereignis-Editor



Der Ereignis-Editor besteht im wesentlichen aus 3 Teilen
Links: Liste mit allen Ereignissen die programmiert werden können
Mitte: Der Editor für die Programmierung
Rechts: Hilfe der verfügbaren Funktionen/Proceduren. Durch ein Doppelklick wird die Auswahl in den Editor übernommen

Weiterhin hat man hier die Möglichkeit das Ereignis zu testen. Für den Test einen Rennbildschirm zu starten und ein externes Script zu laden bzw. zu speichern.

Ablauf:
Auswahl des Ereignisses, dass programmiert bzw. geändert werden soll. Nach der Auswahl sieht man das programmierte Ereignis im Editor. Dieser ist aber für die Eingabe noch nicht freigegeben.
Nun den „Ändern“ Button drücken. Der Editor ist nun freigegeben und man kann los legen.
Über den „einfachen grünen Pfeil“ kann man seinen Code überprüfen lassen.
Über den „doppelten grünen Pfeil“ wird der Code ausgeführt. Je nach Code ist es notwendig, dass vorher ein Rennbildschirm ausgewählt und gestartet wird.
Durch drücken von „Übernehmen“ wird der Code für dieses Ereignis abgelegt.
Beachte: Eine endgültige Speicherung aller Ereignisse findet erst statt, wenn man den „Speichern“ Button drückt. Bei „Abbrechen“ werden alle Änderungen verworfen.

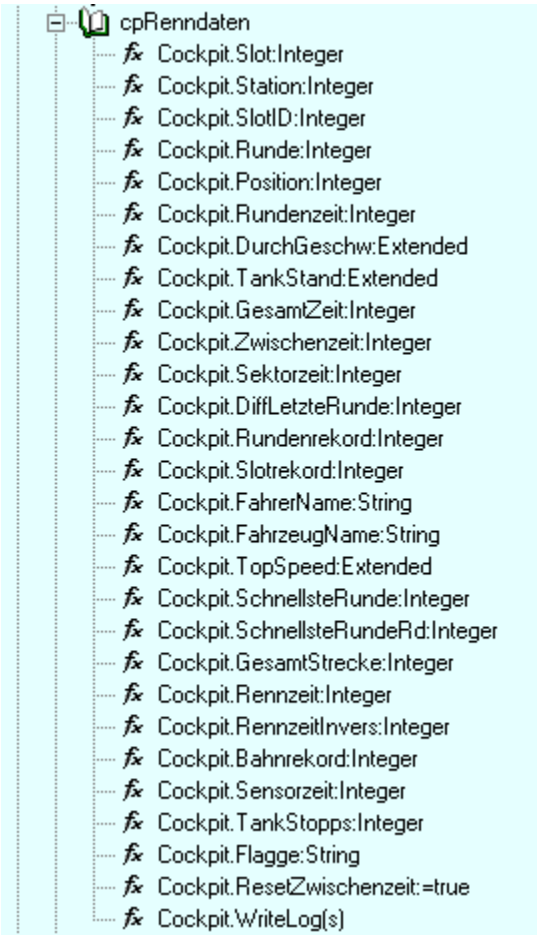
In der Hilfe stehen alle Funktionen/Erweiterungen die Cockpit mit bringt

Generell gilt: Mit Slot ist nicht die Slotnummer gemeint sondern die Position des Fahrers im StartCenter bzw. in der Liste der am Rennen beteiligten Fahrer.
(Bei Analog und Einfaches Rennen ist Slot = Slotnummer. Bei einem SerienRennen muss dies nicht so ein)

cpRenndaten
Über die Cockpit Instanz findet der Zugriff auf die Renndaten statt.
Immer zuerst den Slot setzen und dann die Daten vom Slot abrufen
Bei TopSpeed muss zusätzlich zum Slot noch die Station gesetzt werden.

Beispiel: Cockpit.Slot := 3;
cpShowMessage(IntToStr(Cockpit.Position));
Liefert die Position vom 3. Fahrer

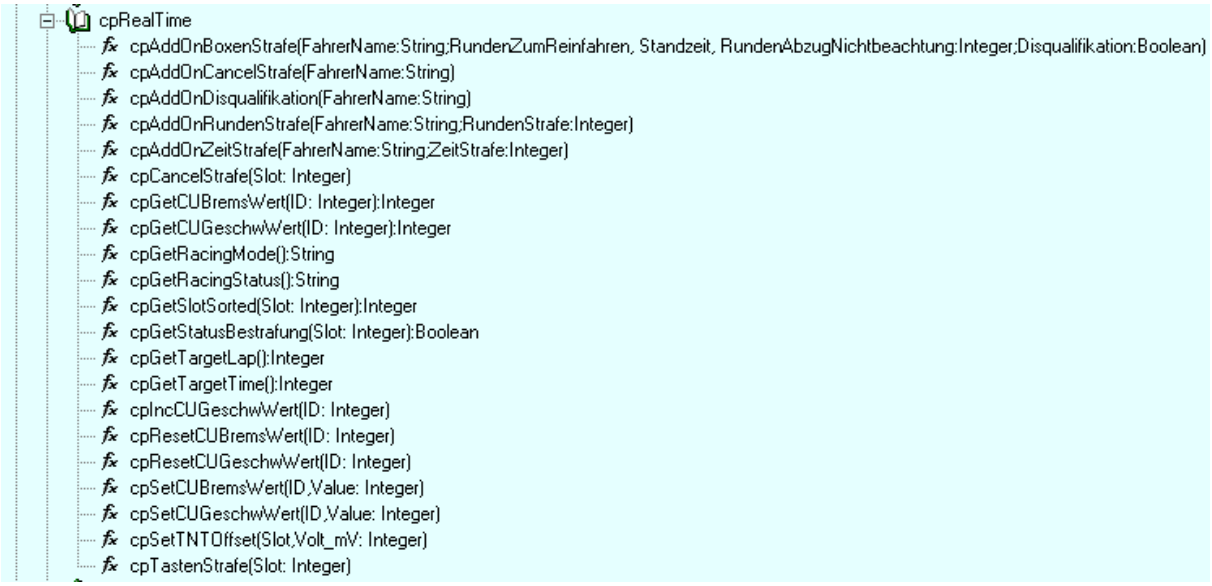
Cockpit.Station := 1;
cpShowMessage(IntToStr(Cockpit.TopSpeed));
Liefert die TopSpeed Geschwindigkeit der Station 1 vom 3. Fahrer



Mehr Info siehe Schnittstellenaufrufe

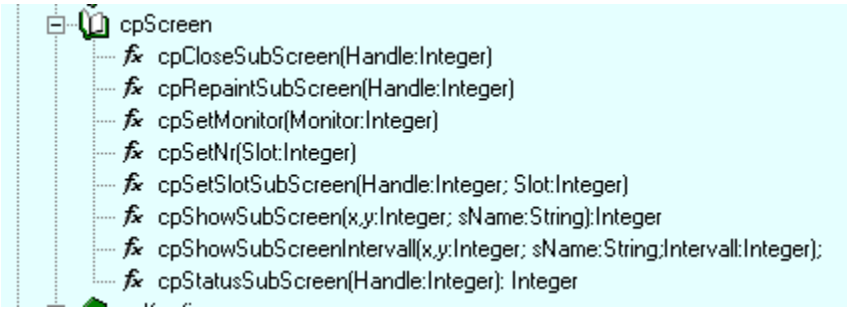
cpRealTime
Funktionen die Informationen über das Renngeschehen liefern, solange die Session läuft

Beispiel: cpTastenStrafe(1);
Der 1. Fahrer in der Starterliste erhält die Strafe die im Tastenbestrafungsdialog eingestellt ist.



cpScreen
Ausgeben von SubScreens im Rennbildschirm
Beispiel: cpSetMonitor(2);
cpShowSubScreenIntervall(100,200,‘Info.xml‘,5000);

5 Sek. einen SubScreen im Monitor 2 anzeigen lassen

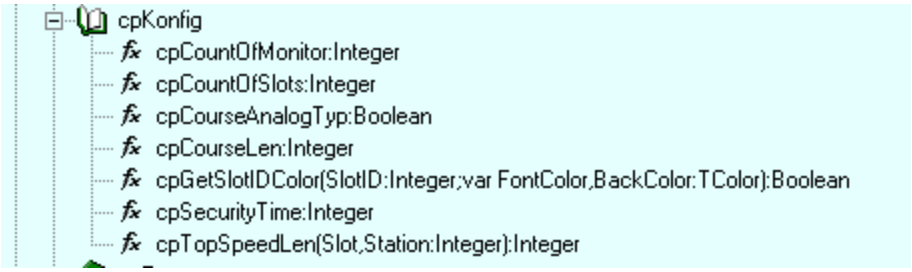


Es gibt 2 Möglichkeiten eine Variable in einem SubScreen anzuzeigen

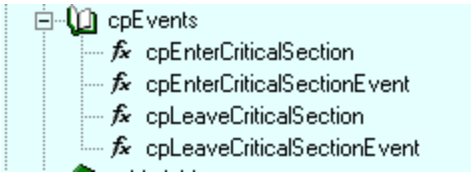
1. Über setzen von Nr=-2
   Im Designer bei der Variablen im Eigenschaftseditor und dort im Feld: Spur/Kodierung oder Platz eine -2 eintragen
   Im Script über cpSetNr(Slot) zeigen dann alle Variablen die -2 eingetragen haben die Daten vom Wert: Slot an.

2. Siehe dazu: RBS mit AddOn Variablen

cpKonfig
Konfigurationsdaten der Bahn
Beispiel: cpCountOfSlots: Liefert Anzahl Slots/Fahrer die in der Rennbahnkonfig angegeben ist.



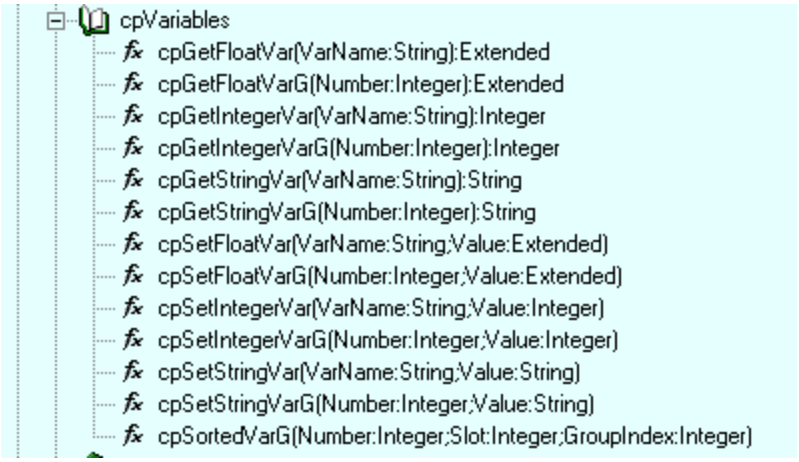
cpEvents
CriticalSection für einen Event oder für ein komplettes Package aufrufen



Damit kann man verhindern, dass ein bestimmter ProgrammierCode mehrfach aktiv läuft. Weitere Tasks die diesen ProgrammierCode benutzen wollen müssen warten bis cpLeaveCriticalSection aufgerufen wurde.

cpVariables
Globale und Package Variablen setzen / abfragen

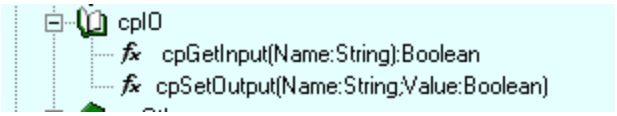
Globale Variablen haben einen Integer Wert als Parameter und sind Package übergreifend gültig
Package Variablen haben einen Namen als Parameter und sind nur innerhalb eines Packages gültig.



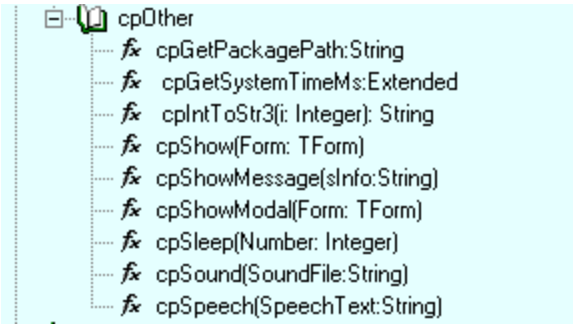
cpSortedVarG:

Wird benötigt wenn man mehrere Variablen mit Werten hat die Sortiert auf dem Rennbildschirm ausgegeben werden sollen.
Siehe dazu Package: Cockpit.AbstandZumSchnellstenFahrer

cpIO
Eingänge abfragen und Ausgänge setzen (AUS (false) oder EIN (true))



cpOther
Alles andere



Mehr Info siehe Schnittstellenaufrufe

5. Ereignisse

Für jedes programmierte Ereignis das Auftritt wird eine eigenständige Task erzeugt.
Dies ist sehr wichtig und muss bei der Programmierung immer berücksichtigt werden.

Beispiel Start/Ziel:
Dieses Ereignis wird aufgerufen, sobald ein Fahrzeug über Start/Ziel fährt.
Welches Fahrzeug das war kann über die „Cockpit“ Instanz abgefragt werden.
„Cockpit.Slot“ enthält den Index der Fahrerliste im StartCenter. Startet mit 1.

Beispiel:
„Das Team“ ist Slot=1
„Die 2 Mädels“ ist Slot=2

Fahren die „Die 2 Mädels“ über Start/Ziel wird Cockpit.Slot auf 2 gesetzt.
Über Cockpit.SlotID kann man aber die tatsächliche Spur bzw. ReglerNummer abfragen. (also Cockpit.SlotID würde 3 liefern)

5.1. Ereignis: StartRealTime

Dieses Ereignis wird beim Start des Rennbildschirmes aufgerufen.
Kommt also nur einmal.
Hier ist es Möglich eine Endlosschleife zu programmieren und auf irgendwelche andere Ereignisse zu warten. ***Aber immer auch ein cpSleep einbauen***

Beispiel:

```
begin
     while ( True ) do
     begin
       // auch bei Pause drin bleiben
       if (cpGetRacingMode() = 'R') AND ((cpGetRacingStatus()='R') OR (cpGetRacingStatus()='P')) then
       begin
         if StartRacing = False then
         begin
           // Das Rennen wurde gestartet -> Initialisierungen durchführen
           StartRacing:=True;
         End
         Else
         Begin
           //  Das Rennen läuft nun

         End;
        End;
      End;
     // sehr wichtig
     cpSleep( 250 );
End;
```

In diesem Beispiel wird solange gewartet bis ein Rennen gestartet wird.
Im ersten Abschnitt (StartRacing=False) kann man notwendige Variablen initialisieren und im zweiten Abschnitt ist man dann solange bis das Rennen wieder beendet ist.

5.2. Ereignis: StartZiel

Dieses Ereignis wird beim überfahren des Start/Ziel Sensors/Lichtschranke ausgelöst. D.h. dieses Ereignis wird mehrmals aufgerufen werden.
Eine Endlosschleife würde hier dazu führen, dass jedes Ereignis eine Task erzeugt die nicht mehr beendet werden würde. -> SystemCrash

Beispiel:

Welcher Slot das Ereignis ausgelöst hat steht in Cockpit.Slot

Beispiel: „Ausgang8“ soll für 2 Sek. gesetzt und dann wieder für 1 Sek. zurück gesetzt werden wenn Start/Ziel für Slot=2 das Ereignis ausgelöst hat.

Würde man so programmieren

```
if Cockpit.Slot = 2 then
begin
  // Hier Beginn kritische Sektion
  cpSetOutput('Ausgang8',True);
  cpSleep( 2000 );
  cpSetOutput('Ausgang8',False);
  cpSleep( 1000 );
end;
```

Hier macht der Ausgang8 unter Umständen nicht das was man will.
Kommt z.B. nach 2,1 Sek. nochmals dieses Ereignis wird der Ausgang8 wieder gesetzt, obwohl er in der ersten Task nach 2 Sek. ausgeschaltet wurde.

Kann man ausschließen, dass die Ereignisse so schnell hinter einander wieder eintreffen, kann man es so programmieren.

```
if Cockpit.Slot = 2 then
begin
  if cpGetIntegerVar(,Sperre‘)=0 then
  begin
    cpSetIntegerVar(,Sperre‘,1);
    cpSetOutput('Ausgang8',True);
    cpSleep( 2000 );
    cpSetOutput('Ausgang8',False);
    cpSleep( 1000 );
    cpSetIntegerVar(,Sperre‘,0);
 end;
end;
```

In diesem Beispiel werden alle weiteren Tasks die innerhalb der 3 Sek. kommen abgewiesen und werden nicht bearbeitet.

```
if Cockpit.Slot = 2 then
begin
  // Hier Beginn kritische Sektion
  cpEnterCriticalSectionEvent;
  cpSetOutput('Ausgang8',True);
  cpSleep( 2000 );
  cpSetOutput('Ausgang8',False);
  cpSleep( 1000 );
  // Hier Ende kritische Sektion
  cpLeaveCriticalSectionEvent;
end;
```

Mit cpEnterCriticalSectionEvent; und cpLeaveCriticalSectionEvent; kann man einen Bereich so absichern, dass dieser nur von einer Task belegt wird.
Alle weiteren Tasks warten in cpEnterCriticalSectionEvent bis der geschützte Bereich wieder frei ist. Sobald dieser Bereich wieder frei ist wird er wieder von einer Task belegt.

Programmiert man so, werden alle Ereignisse sequentiell hintereinander bearbeitet. Es geht kein Ereignis verloren.

Welche Variante man nun wählt hängt davon ab was man will.

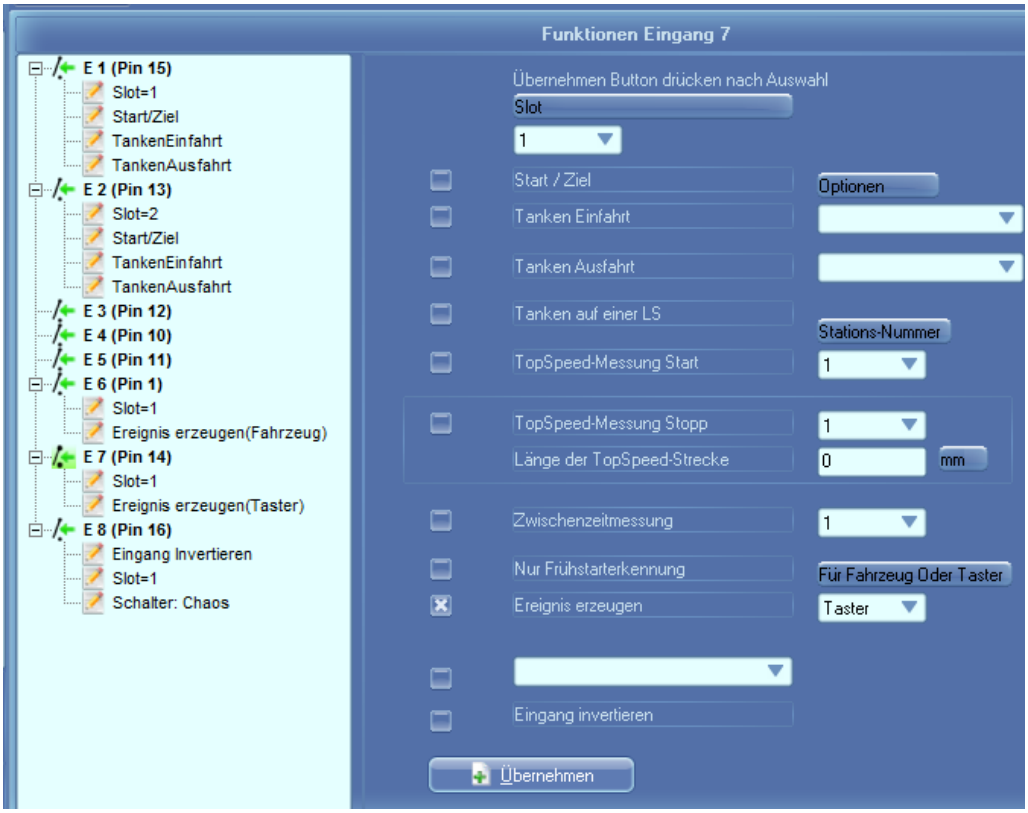
5.3. Ereignis: TopSpeedStart bzw. Stopp

Wird ausgelöst wenn der TopSpeed Start bzw. Stopp Sensor überfahren wird

Übergabeparameter: Cockpit.Slot: Welcher Slot
Cockpit.Stationt: Welche Station hat Ereignis ausgelöst

5.4. Ereignis: DigInputEvent

Um Ereignisse von zusätzlichen Sensoren/Taster erfassen zu können muss man in der Rennbahn die Funktion „Ereignis erzeugen“ auf einen Eingang legen.
Dies kann für ein Fahrzeug oder für ein Taster konfiguriert werden
Siehe Eingang E6 und E7



Dieses Ereignis wird dann ausgelöst wenn der Sensor / Taster aktiviert wird.

Event für Fahrzeug

Übergabeparameter: Cockpit.Slot: Welcher Slot
Cockpit.Station: Eingang 1-8
Cockpit.Parameter: USB-Steckplatz USB-Box ab 1

Event für Taster

Übergabeparameter: Cockpit.Slot: 1: Signal wechselte von 0 auf 1
0: Signal wechselte von 1 auf 0
Cockpit.Station: Eingang 1-8
Cockpit.Parameter: USB-Steckplatz USB-Box ab 1

5.5. Ereignis: KeyDown

Wird ausgelöst wenn eine Taste gedrückt wurde.
Der Haupt-Rennbildschirm muss dazu aktiviert sein (Geht nur im Haupt-RBS)
Parameter:
Cockpit.Parameter: Key-Code der Taste

Welche Taste welchen Code liefert kann man mit diesem Befehl heraus finden
cpShowMessage(IntToStr(Cockpit.Parameter));

5.6. Ereignis: ChangeRealTimeStatus

Wird ausgelöst wenn sich der RennStatus verändert.
Parameter:
Der Status steht dann in Cockpit.Parameter

1 = Rennen ist gestartet:Sequenz 0 und 1. Rennen geht Weiter: Nur 1
2 = Start mit Startampel. Frühstartüberwachung aktiviert
3 = Sieger ist im Ziel
4 = Rennen beendet
6 / 7 =6 Pause; 7 Pause mit Nachlaufzeit
9 = Rennen Abgebrochen

5.7. Ereignis: StartZiel-SichTime

Wird ausgelöst wenn bei einem Fahrzeug/Fahrer die Sicherheitsrundenzeit Überwachung ausgelöst wird.
Parameter:
Cockpit.Slot enthält den Auslöser

5.8. Ereignis: Tanken Ein- und Ausfahrt

Wird ausgelöst wenn ein Fahrzeug über die Tanken Ein- bzw. Ausfahrt Sensoren fährt.
Parameter:
Cockpit.Slot enthält den Auslöser

5.9. Ereignis: SchnellsteRunde

Wird ausgelöst wenn ein Fahrer eine neue schnellste Runde fährt.
Parameter:
Cockpit.Slot enthält den Auslöser

5.10. Ereignis: Rundenrekord/Slotrekord/Bahnrekord

Wird ausgelöst wenn ein Fahrer einen neuen Rundenrekord/Slotrekord oder Bahnrekord fährt.
Parameter:
Cockpit.Slot enthält den Auslöser

5.11. Ereignis: Bestrafung

Wird ausgelöst wenn eine Bestrafung erfolgt ist
Parameter:
Cockpit.Slot enthält den Auslöser
Cockpit.Station=<1=ausgesprochen;2=disqualifiziert>
Cockpit.Parameter=<1=Rundenstrafe;2=Zeitstrafe;3=Boxengassestrafe>

5.12. Ereignis: ChangeScreen

Wird ausgelöst wenn der Rennbildschirm gewechselt wurde.
Also z.B. wenn man unterschiedliche Rennbildschirme für Training/Quali und Rennen definiert hat und nun von Training auf Rennen umschaltet.
Während dem Rennen tritt dieses Ereignis nicht auf.
Keine Parameter

5.13. Ereignis: CUSensorGruppe

Nur mit CU30352:
Wird ausgelöst wenn ein Fahrzeug über den CUAdapter fährt und dieser eine SensorGruppe 2 oder 3 liefert.
Parameter:
Cockpit.Slot=<FahrzeugID> bzw. Reglernummer
Cockpit.Station=<SensorGruppe 2/3>

5.14. Ereignis: BeforeClose

Wird ausgelöst nachdem der Rennbildschirm geschlossen wurde.
Keine Parameter

6. RBS mit AddOn Variablen

Möchte man Ergebniss von einem AddOn in einem RBS darstellen gibt es folgende Möglichkeiten:

Über SubScreens die man im RBS einblenden kann.
Dazu einfach mal die Beispiele (z.B.: HelloWorld) ansehen.
SubScreens haben den Vorteil, dass man einen Standard RBS verwenden kann und blendet nur seine AddOn Ergebnisse ein.

Eine zweite Möglichkeit besteht einen vorhandenen RBS anzupassen und entsprechende Variablen einzufügen.

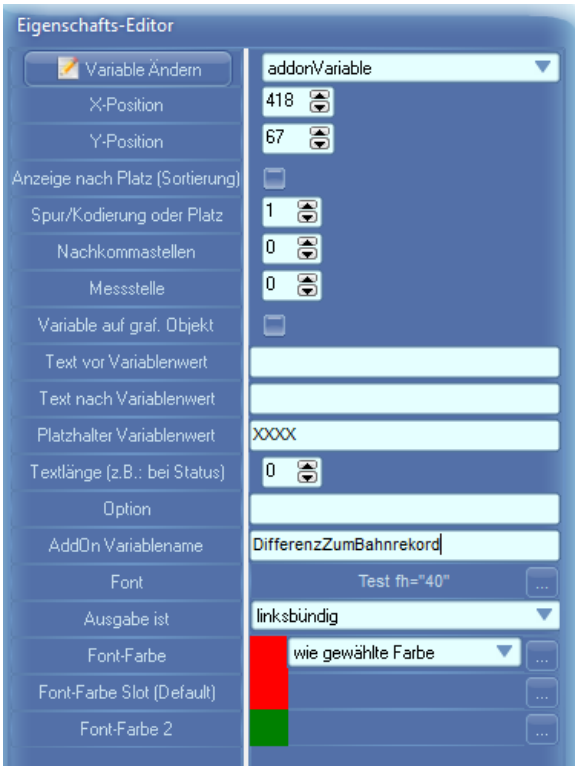
VariablenTypen die verwendet werden können:

- AddOn Variable
- AddOn Balkengrafik
- Einblenden von Symbolen / Bilder bzw. oder mit Audio-Ausgabe

Diese Typen können Fahrergebunden angezeigt werden.
D.h. Ändert sich die Position des Fahrers im RBS wandert der Variablenwert vom AddOn mit.

Direkt im Designer kann man nun AddOn Variable oder Balkengrafik auswählen.

AddOn Variable



Hier ist wichtig: (außer allen anderen)

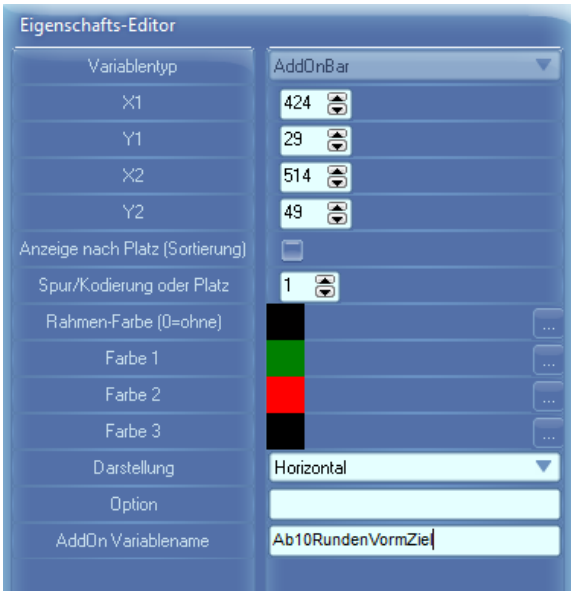
Spur/Kodierung oder Platz: Bindung an den Fahrer.

Nachkommastellen : Wird bei Integer oder Extended Variablen benötigt.

Platzhalter: Für den Platz die der Wert benötigt

AddOn Variablenname: Name der im AddOn verwendet wird.

Balkengrafik



Hier ist wichtig: (außer allen anderen)

Spur/Kodierung oder Platz: Bindung an den Fahrer.

Farbe 1 bestimmt die Farbe des Balkens (Farbe 2/3 wird nicht benötigt)

AddOn Variablenname:
Name der im AddOn verwendet wird.

Die Balkengrafik benötigt Werte von 0 (Balken nicht sichtbar) bis 100 (Balken komplett sichtbar)

cpSetRBSBarColor(VarName:String;Slot:Integer;Value:Integer);
Damit kann man im AddOn die Farbe des Balkens setzen

Symbole

Hier ist wichtig: (außer allen anderen)

Spur/Kodierung oder Platz: Bindung an den Fahrer.

Symbol-Anzeigen / Symbol-Ausschalten:
Mit welchen Events wird das Symbol Ein- bzw. Ausgeblendet.

Dies wären dann alle Möglichkeiten um in einem RBS direkt AddOn Werte darzustellen (ohne SubScreen)

Wie sieht nun die Programmierung im AddOn aus:

Um diese Variablen zu beschreiben gibt es 3 neue AddOn Befehle:

procedure cpSetRBSIntegerVar(VarName:String;Slot:Integer;Value:Integer)',
procedure cpSetRBSFloatVar(VarName:String;Slot:Integer;Value:Extended)'
procedure cpSetRBSStringVar(VarName:String;Slot:Integer;Value:String)'

Beispiel 1:

Wir wollen zeitlichen Abstand von jeder gefahrenen Rundenzeit zum Bahnrekord darstellen.
Jeder Fahrer im RBS bekommt 3 neue AddOn Variablen für Zahlen/Texte (addOnVariable). Damit auch nach einer Sortierung im RBS die Zuordnung noch stimmt muss im RBS diesen Variablen die richtige Spur/Kodierung/Platz zugewiesen werden. Dies muss dann auch beim setzen der Variable berücksichtig werden.

Im Start/Ziel Ereignis muss folgendes Programmiert werden:

```
begin
  cpSetRBSIntegerVar( 'DifferenzZumBahnrekord', Cockpit.Slot, Cockpit.Rundenzeit - Cockpit.Bahnrekord );
end.
```

Das wars schon.
Zeiten sind im AddOn alle in mSek. deshalb muss man bei den Variablen im AddOn die Anzahl Nachkommastellen auf 3 setzen wenn man Sekunden anzeigen möchte.

Der Slot kommt direkt vom Start/Ziel Event. Deshalb kann man gleich Cockpit.Slot verwenden.

Beispiel 2:

Wir wollen für jeden Fahrer eine Balkengrafik darstellen die ab 10 Runden vor dem Ziel bei jeder Runde den Balken kleiner darstellt.

Dies wird wieder im Start/Ziel Ereignis programmiert

```
var
  BisZiel:Integer;
begin
   if cpGetRacingMode() = 'R' then
   begin
     BisZiel := cpGetTargetLap - Cockpit.Runde;
     cpSetRBSIntegerVar( 'Ab10RundenVormZiel', Cockpit.Slot, BisZiel * 10 );
   end;
end.
```

Sind wir mehr oder gleich 10 Runden vor dem Ziel wird ein Wert größer oder gleich 100 an die Balkengrafik übergeben.
D.h. diese wird zu 100% dargestellt.
Bei 9 Runden vor dem Ziel wird ein Wert von 90 übergeben. D.h. 90% des Balkens werden dargestellt. Usw.

Das ganze funktioniert nur beim Rennen und nur wenn ein Rundenrennen stattfindet.

Beispiel 3:

Hier wird es schon schwieriger ein Beispiel zu finden weil es doch jede Menge Audio Ereignisse gibt die schon das meiste Abfangen ohne Programmierung.

Wir wollen ein Symbol darstellen wenn die Rundenzeit zwischen 6 Sek und 8 Sek liegt

Dies wird wieder im Start/Ziel Ereignis programmiert

```
begin

  if ( Cockpit.Rundenzeit >= 6000) AND ( Cockpit.Rundenzeit <= 8000 ) then
    cpSymbolEvent('Customer1 Ein', Cockpit.Slot)
  else
    cpSymbolEvent('Customer1 Aus', Cockpit.Slot);

end.
```

Für Symbole die über ein AddOn gesteuert werden müssen die Customer1 bis Customer3 Events verwendet werden. (Die anderen kann man natürlich auch verwenden)
Als Name wird der angezeigte Name im AddOn-Manager verwendet.

Mit ***cpAudioEvent(NameAudioEvent)*** wird das Symbol + der eingetragene Sound/bzw. Sprachtext im AudioManager ausgegeben !

AddOnVariable, BalkenGrafik und Symbole Fahrerunabhängig darstellen

Will man eine Variable ohne Bindung an den Fahrer anzeigen lassen
(z.B.: Restrennzeit) muss man beim Aufruf der Prozeduren den Wert
Slot=0 angeben.

Im Eigenschaftseditor zu dieser Variable bei Spur/Kodierung oder Platz muss ebenfalls eine 0 eingetragen werden.

Aufurf sieht dann z.B. so aus:

cpSetRBSIntegerVar('Restrennzeit', 0, 150);

Hinweis:

Also IntegerVar und FloatVar können über die Angabe der Anzahl Nachkommastellen formattiert werden.
StringVar wird so ausgegeben wie angegeben.

Diese Variablen sind nicht identisch mit den Variablen von cpSetIntegerVar.
Auch wenn man den gleichen Namen verwendet.

7. Schnittstellenaufrufe

Hinweis: Slot ist nicht die Slotnummer / Regler-ID sondern der Index des Fahrers in der Starterliste !!
SlotID ist bei Analog die Spurnummer und bei digital die Reglernummer

Funktion	Variablentyp	Beschreibung
cpRenndaten Entspricht weitgehend den Variablen für die Rennbildschirmentwicklung		
Cockpit.Slot	Integer	Slot ist der Index des Fahrers/Fahrzeugs in der Starterliste Also erster Eintrag hat den Index=1 Will man Daten vom 2ten Fahrer in der Liste anfordern muss Cockpit.Slot=2 gesetzt werden.
Cockpit.Station	Integer	Liefert die Messstelle für Sektorzeit, Zwischenzeit und TopSpeed-Messung.
Cockpit.SlotID	Integer	Liefert die tatsächliche Regler-ID oder die Spur-Nr
Cockpit.Runde	Integer	Enthält die Anzahl der gefahrenen Runden
Cockpit.Position	Integer	Enthält die Position
Cockpit.Rundenzeit	Integer	Enthält die aktuelle Rundenzeit
Cockpit.DurchGeschw	Extended	Enthält die durchschnittliche Geschwindigkeit pro Runde. Nicht zu verwechseln mit TopSpeed!
Cockpit.TankStand	Extended	Liefert den Tankfüllstand
Cockpit.GesamtZeit	Integer	Enthält die Gesamtzeit, die für die Renndistanz benötigt wurde.
Cockpit.Zwischenzeit Cockpit.Sektorzeit Cockpit.Station	Integer	Liefert die Zwischenzeit /Sektorzeit. Vorher muss Cockpit.Station gesetzt werden um die gewünschte Station zu erhalten.
Cockpit.DiffLetzteRunde	Integer	Enthält die Zeitdifferenz zwischen aktueller und der vorherigen Runde
Cockpit.Rundenrekord	Integer	Enthält den Rundenrekord. (Macht Sinn bei digitalen Rennbahnen) (nicht mehr verwenden)
Cockpit.Slotrekord	Integer	Liefert den Slotrekord. (Macht Sinn bei analogen Rennbahnen)
Cockpit.FahrerName	String	Enthält den Namen des Fahrers. Abfrage des Feldes "Fahrername" aus dem DataCenter.
Cockpit.FahrzeugName	String	Enthält den Namen des Fahrzeugs ("Name des Fahrzeugs") aus dem DataCenter.
Cockpit.TopSpeed Cockpit.Station	Extended	Liefert die Höchstgeschwindigkeit auf einer Strecke mit TopSpeed-Messung. Vorher muss Cockpit.Station gesetzt werden um die gewünschte Station zu erhalten.
Cockpit.SchnellsteRunde	Integer	Enthält die schnellste im aktuellen Rennen gefahrene Rundenzeit
Cockpit.SchnellsteRundeRd	Integer	Enthält die Runde, in der die schnellste Rundenzeit erzielt wurde
Cockpit.GesamtStrecke	Integer	Enthält die zurückgelegte Gesamtstrecke für ein Rennen
Cockpit.Rennzeit	Integer	Enhält die aktuelle Rennzeit (noch zu fahren)
Cockpit.RennzeitInvers	Integer	Enthält die umgekehrte Rennzeit (bis jetzt gefahren)
Cockpit.Bahnrekord	Integer	Schnellste jemals gefahrene Runde
Cockpit.Sensorzeit	Integer	Liefert die Zeit gerechnet vom Rennstart beim Überfahren des entsprechenden Sensors.
Cockpit.TankStopps	Integer	Anzahl der absolvierten Boxenstopps
Cockpit.Flagge	String	Liefert das Flaggensymbol
Cockpit.ResetZwischenzeit	Boolean	Setzt alle Zwischenzeiten auf 0
Cockpit.WriteLog(s)	String	Schreibt in das Logfile. Logging muss eingeschaltet sein Logfile wird im Datenverzeichnis von Cockpit abgelegt
Cockpit.TankenAktiviert	Boolean	Liefert „True“ wenn getankt werden kann bzw. wenn aufgetankt wird.
Cockpit.EsWirdGetankt	Boolean	Liefert „True“ wenn ein Tankvorgang läuft. True bleibt dann bis „Tanken Ausfahrt“ stehen.

Cockpit.PersRundenrekord	Integer	Gefahrener Rundenrekord von diesem Fahrer
Cockpit.PersRundenrekordFz	Integer	Gefahrener Rundenrekord von diesem Fahrer mit diesem Fahrzeug
Cockpit.PersSlotrekord	Integer	Gefahrener Slotrekord von diesem Fahrer auf diesem Slot
Cockpit.PersSlotrekordFz	Integer	Gefahrener Slotrekord von diesem Fahrer mit diesem Fahrzeug auf diesem Slot
Cockpit.RennTeilnehmer	Boolean	Prüft ob dieser Fahrer ein Rennteilnehmer ist (Ghost/Pace ist keiner)
Cockpit.CockpitTankenSperren	Boolean	True: Sperrt das Cockpit-Tanken für diesen Fahrer
Cockpit.AnzahlSiege	Integer	Statistik: Anzahl Siege von diesem Fahrer
Cockpit.AnzahlRennen	Integer	Statistik: Anzahl Rennen von diesem Fahrer

cpRealTime: Funktionen die Informationen über das Renngeschehen liefern, solange die Session läuft

cpGetRacingMode()	String	Liefert den aktuellen Rennmodus. Mögliche Werte sind: T, Q, R
cpGetRacingStatus()	String	Liefert den aktuellen Rennstatus. Siehe Ereignis: ChangeRealTimeStatus
cpGetStatusBestrafung(Slot:Integer)	Boolean	Fragt den Status für eine ausgesprochene Strafe für diesen SLot ab. Ist das Ergebnis 'true' wurde eine Strafe verhängt. Ist das Resultat 'false' ist keine Strafe aktiv.
cpGetTargetLap()	Integer	Ermittelt die eingestellten Sollrennrunden
cpGetTargetTime()	Integer	Liefert die eingestellte Sollrennzeit
cpSetTNTOffset(Slot:Integer, Volt_mV:Integer)		Setzt bei Einsatz eines Tanknetzteils auf einer analogen Rennbahn per AddOn die für eine Spur vorhandene Spannung.
cpTastenStrafe(Slot:Integer)		Verhängt eine Tastenstrafe
cpCancelStrafe(Slot:Integer)		Hebt die verhängte Strafe für Slot auf
cpAddOnRundenStrafe (FahrerName:String;RundenStrafe:Integer)		Rundenstrafe für Fahrer vergeben
cpAddOnZeitStrafe (FahrerName:String;ZeitStrafe:Integer)		Zeitstrafe für Fahrer vergeben
cpAddOnBoxenStrafe (FahrerName:String;RundenZumReinfahrn, Standzeit,RundenAbzugNichtbeachtung: Integer;Disqualifikation:Boolean; RundenAbzugRennende:Integer)		BoxengasseStrafe für Fahrer vergeben
cpAddOnDisqualifikation (FahrerName:String)		Fahrer disqualifizieren
cpAddOnCancelStrafe(FahrerName:String)		Strafe wieder aufheben
cpSimulateButton(ID:Integer)		Gleiche wie ein Anklicken der unteren Buttonleiste im Rennbildschirm vom Hauptmonitor ID = 1: Startampel; 2: Abbruch; 3: Pause; 4: Weiter; 5: Training 6: Quali; 7: Schnellstart

Nur mit CU30352:
ID ist die ReglerNummer

cpGetCUBremsWert(ID:Integer)	Integer	Holt den aktuellen Bremswert
cpGetCUGeschwWert(ID:Integer)	Integer	Holt den aktuellen Geschw.Wert
cpIncCUGeschwWert(ID:Integer)		Erhöht den Geschwindigkeitswert für die Fahrzeug ID um den Wert + 1
cpResetCUBremsWert(ID:Integer)		Reset gesetzten Bremswert
cpResetCUGeschwWert(ID:Integer)		Reset gesetzten Geschwindigkeitswert
cpSetCUBremsWert(ID:Integer)		Setzt einen bestimmten Bremswert
cpSetCUGeschwWert(ID:Integer)		Setzt einen bestimmten Geschwindigkeitswert
cpSetCUBremsWertFast(ID:Integer)		Setzt einen bestimmten Bremswert sofort
cpSetCUGeschwWertFast(ID:Integer)		Setzt einen bestimmten Geschwindigkeitswert sofort
cpResetCUBremsWertFast(ID:Integer)		Reset den gesetzten Bremswert sofort
cpResetCUGeschwWertFast(ID:Integer)		Reset den gesetzten Geschwindigkeitswert sofort

cpScreen: Funktionen die mit Rennbildschirmen und individuellen Unterbildschirmen zu tun haben		
cpCloseSubScreen(Handle:Integer)		Schließt einen angezeigten Unterbildschirm mit der entsprechenden Zuordnungsnummer
cpRepaintSubScreen(Handle:Integer)		Aktualisiert den Inhalt des Unterbildschirms mit der Zuordnungsnummer
cpSetMonitor(Monitor:Integer)		Setzt die Nummer für den angegebenen Zusatzmonitor. So können z.B. Anzeigelemente gezielt auf einem bestimmten Monitor dargestellt werden.
cpSetNr(Slot:Integer)		Setzt die Nummer im SubScreen, wenn dort eingestellt ist, dass die Daten aus dem AddOn kommen.
cpSetSlotSubScreen(Handle:Integer, Slot:Integer)		Nummer im SubScreen wird damit geändert
cpShowSubScreen(x,y:Integer, SName:String)		Zeigt einen Unterrennbildschirm an. X,y repräsentieren die obere linke Ecke. sName ist der Name der Unterbildschirmdatei. Diese muss im AddOn Verzeichnis liegen.
cpShowSubScreenIntervall (x,y:Integer, SName:String, Intervall:Integer)		Wie oben. Blendet einen Unterbildschirm ein und aus. Die Häufigkeit ist durch den Wert 'Intervall' vorgegeben. Dieser ist in Millisekunden als Zahl einzutragen.
cpStatusSubScreen(Handle:Integer)	Integer	Liefert Status=1 wenn SubScreen noch existiert

Funktionen für die Abfragen von Konfigurationsdaten der Rennbahn		
cpCountOfMonitor	Integer	Anzahl definierter Zusatzmonitore
cpCountOfSlots	Integer	Anzahl Fahrer die in der Rennbahn hinterlegt ist
cpCourseAnalogTyp	Boolean	Liefert die Information, ob es sich um eine analoge Rennbahn handelt. Ist das Ergebnis 'false' handelt es sich um eine digitale Strecke.
cpCourseLen	Integer	Liest die in der Rennbahn hinterlegte Streckenlänge.
cpGetSlotIDColor (SlotID:Integer, var FontColor, BackColor: TColor)	Boolean	Liefert die eingestellten Spurfarben
cpSecurityTime	Integer	Liest die in der Rennbahn eingetragene Sicherheitsrundenzeit aus.
cpTopSpeedLen(Slot, Station)	Integer	Holt die in der Rennbahn hinterlegte Messtreckenlänge für die TopSpeed-Messung. Parameter ist der Slot und die Messstelle.

cpEvents: Funktionen für Ereignisbehandlung		
cpEnterCriticalSection		Absichern von Bereiche gegen mehrfache Aufrufe Ist global gültig
cpEnterCriticalSectionEvent		Absichern von Bereiche gegen mehrfache Aufrufe Gilt nur für das Ereignis in dem es aufgerufen wird
cpLeaveCriticalSection		Bereich wieder freigeben damit nächste Task den Bereich betreten kann
cpLeaveCriticalSectionEvent		Bereich wieder freigeben damit nächste Task den Bereich betreten kann

cpVariables: Funktionen zur Arbeit mit AddOn bezogenen Variablen diverser Datentypen		
cpGetFloatVar(VarName:String)	Extended	Liest den Wert einer Package Variable vom Typ Extended aus
cpGetFloatVarG(Number:Integer)	Extended	Liest den Wert einer globalen Variable vom Typ Extended aus
cpGetIntegerVar(VarName:String)	Integer	Liest den Wert einer Package Variable vom Typ Integer aus
cpGetIntegerVarG(Number:Integer)	Integer	Liest den Wert einer globalen Variable vom Typ Integer aus
cpGetStringVar(VarName:String)	String	Liest den Wert einer Package Variable vom Typ String aus
cpGetStringVarG(Number:Integer)	String	Liest den Wert einer globalen Variable vom Typ String aus
cpSetFloatVar(VarName:String, Value:Extended)		Setzt den Wert einer Package Variable vom Typ Extended
cpSetFloatVarG(Number:Integer, Value:Extended)		Setzt den Wert einer globalen Variable vom Typ Extended
cpSetIntegerVar(VarName:String, Value:Integer)		Setzt den Wert einer Package Variable vom Typ Integer
cpSetIntegerVarG(Number:Integer, Value:Integer)		Setzt den Wert einer globalen Variable vom Typ Integer
cpSetStringVar(VarName:String, Value:String)		Setzt den Wert einer Package Variable vom Typ String
cpSetStringVarG(Number:Integer, Value:String)		Setzt den Wert einer globalen Variable vom Typ String

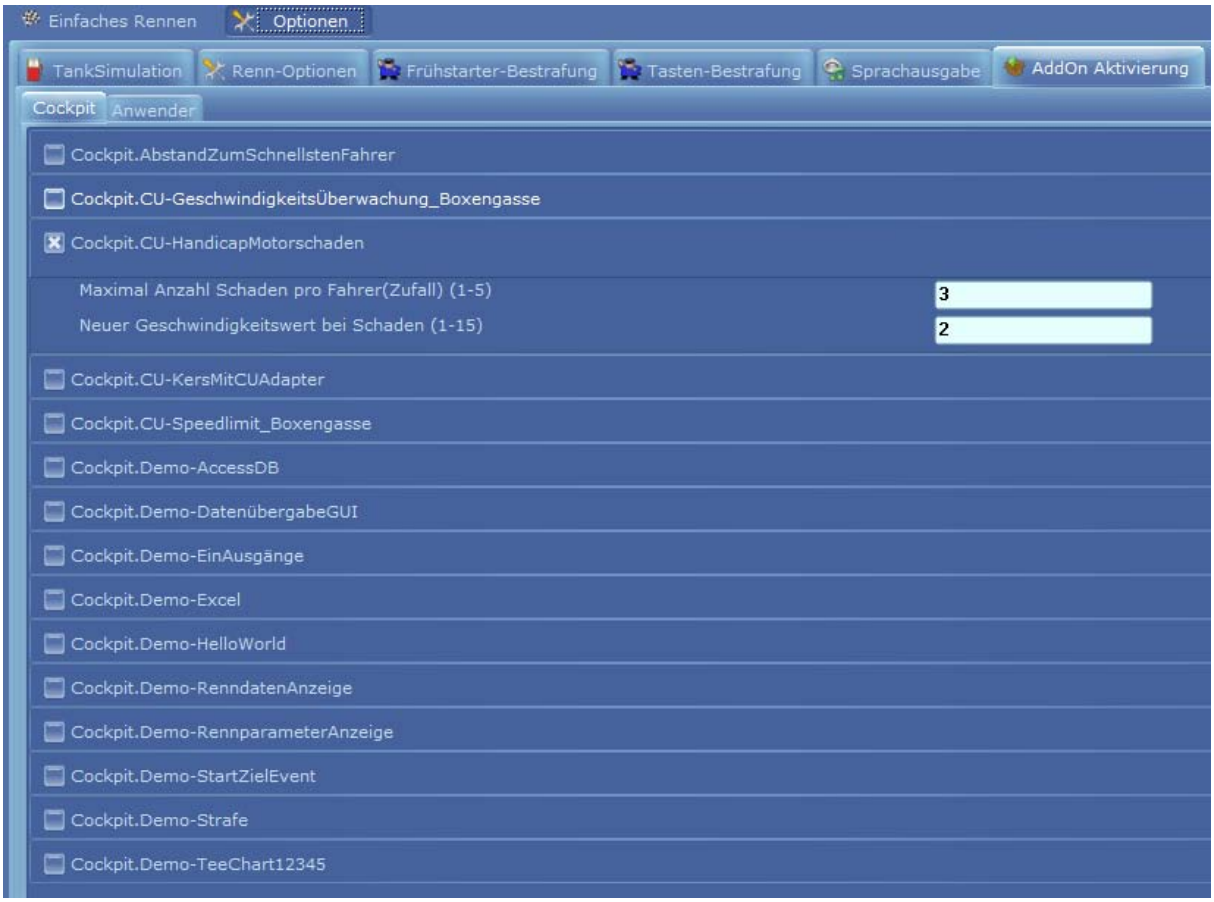
cpRBSVariables: Funktionen um Werte vom AddOn in einem RBS darzustellen (Fahrergebunden Slot>0)		
cpSetRBSIntegerVar(VarName:String;Slot:Integer; Value:Integer)		Setzt einen Integerwert bezogen auf den Slot für eine RBS Variable
cpSetRBSFloatVar(VarName:String;Slot:Integer; Value:Extended)		Setzt einen Floatwert bezogen auf den Slot für eine RBS Variable
cpSetRBSStringVar(VarName:String;Slot:Integer; Value:String)		Setzt einen String bezogen auf den Slot für eine RBS Variable
cpSetRBSBarColor(VarName:String;Slot:Integer;Value:Integer);		Setzt die Farbe für eine Balkengrafik

cpIO: Funktionen für das Arbeiten mit digitalen Ein/Ausgänge Der Name wird in der Datei: ConfigPackage.xml definiert		
cpGetInput(Name:String)	Boolean	cpGetFloatVar(VarName:String)
cpSetOutput(Name:String, Value:Boolean)		Schaltet einen Ausgang, z.B. ein Relais. Ist der Wert "true", ist die Funktion aktiv. Ist der Wert "false" entsprechend inaktiv.

cpOther: Funktionen die übergreifende Aufgaben erfüllen		
cpGetPackagePath	String	Liefert den Pfadnamen zum AddOn-Verzeichnis ausgehend vom Cockpit-V2 AddOn-Ordner
cpGetSystemTime(TimeMs:Extended)	Extended	Liest die aktuelle Systemzeit aus. Da dieser Wert größer sein kann als ein Integer-Wert, hat die Funktion den Datentyp Extended
cpIntToLongTimeFormat(Value,NKs: Integer)	Integer	Formatiert einen Integerwert in ein Datum
cpIntToStr3(i:Integer)	Integer	Formatiert einen Integerwert in eine Zahl mit 3 Nachkommastellen
cpShow(Form: TForm)	TForm	Zeigt ein Dialogformular auf dem Hauptbildschirm an
cpShowMessage(sInfo:String)		Gibt eine Meldung aus
cpShowModal(Form:TForm)	TForm	Zeigt ein Dialogformular auf dem Hauptbildschirm an, das bis zum Schließen vor allen anderen Anwendungsfenstern liegt.
cpSleep(Number:Integer)		Hält die Ausführung des Programms für die in Klammern angegebene Zeitdauer in ms an.
cpSound(SoundFile:String)		Spielt die in Klammern definierte Sounddatei ab. Kommt ein neuer cpSound Befehl wird der aktuelle Sound abgebrochen und der neue Sound gespielt
cpStopSound()		Stoppt den gerade laufende Sound der über cpSound() gestartet wurde
cpSound2(SoundFile:String)	Integer	Spielt die in Klammern definierte Sounddatei ab. Sound läuft unabhängig von einem weiteren cpSound2() Aufruf weiter. Liefert als Returnwert ein Handle auf das Soundobjekt zurück. Kann der Sound nicht abgespielt werden kommt 0 zurück
cpStopSound2(Handle:Integer)		Stoppt einen Sound der über cpSound2() gestartet wurde. Als Übergabeparameter muss das Handle von cpSound2 übergeben werden.
cpSpeech(SpeechText:String)		Gibt den in Klammern stehenden Text über die Sprachausgabe wieder.
cpSymbolEvent(Name des Events:String)		Schaltet ein zugewiesenens Symbol zu diesem Event Ein.- oder Aus
cpAudioEvent(Name des Events:String)		Löst dieses AudioEvents aus. Eingetragener Sound/bzw. Sprachtext wird ausgegeben und ein Symbol (wenn definiert)wird Ein- oder Ausgeblendet

8. Aktivierung der AddOns

Die „AddOn Aktivierungsseite“ ist unter Optionen zu finden.
In den allgemeinen Einstellungen legt man allerdings fest, ob diese Seite im StartCenter / Rennbahn oder in den „allgemeinen Einstellungen“ zu finden ist.



Einfach den Haken für das gewünschte AddOn setzen und damit ist das AddOn aktiviert.
Bei AddOns mit Dateneingabe werden dann auch die Eingabefelder sichtbar.

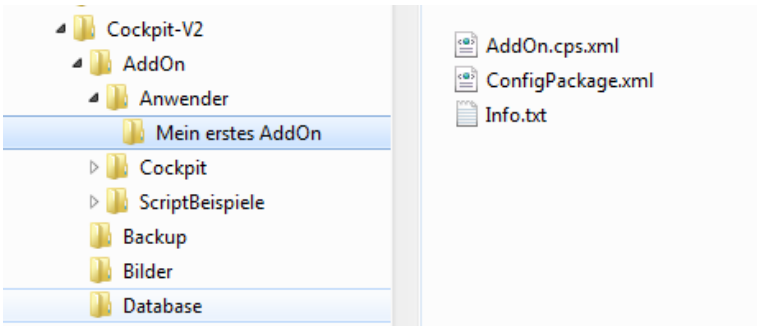
Bleibt man mit der Maus über dem AddOn Text wird die Datei: Info.txt mit der Beschreibung des AddOns angezeigt.

Über Favoriten kann man verschiedene Aktivierungen und Einstellungen für die AddOns ablegen und schnell wieder aufrufen.

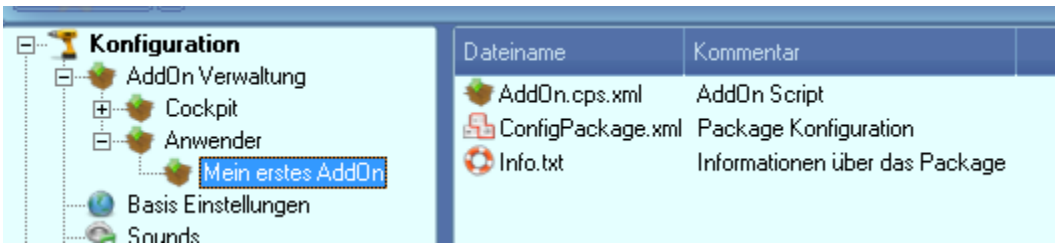
9. Mein erstes AddOn

Wenn Sie ein neues AddOn entwickeln wollen muss man so vorgehen

1) Cockpit beenden
2) Im Verzeichnis AddOn\Anwender habe ich ein leeres Packages für den ersten Start abgelegt. Dieses nun kopieren in diesem Verzeichnis.
3) Dann diesem Verzeichnis einen beliebigen bzw. sprechenden Namen geben. z.B.: Mein erstes AddOn. Keine Umlaute/Sonderzeichen verwenden



4) Nun kann man Cockpit wieder starten und im ConfigCenter können die Dateien AddOn.cps.xml / ConfigPackage.xml und Info.txt bearbeitet werden.



5) Damit man das AddOn auch Life testen kann, muss es aktiviert werden