Implementierungsvorschriften AUTOSAR

6. Juni 2015

Kurzer Überblick

1 Coding-Rules

1.1 Runnables

Jeder Teil des Systems wird in Runnables aufgeteilt. Eine Runnable ist dabei nur eine Funktion, die vom Task, dem die Runnable zugeordnet ist, angesprungen wird(Im besten Fall per `#define RUNABLE (CODE)`). In die Runnable gehört NUR der reine Applikationscode. D.h. sobald eine Kommunikation mit einer anderen Komponente erfolgt, sind entsprechende **RTE**-Funktionen aufzurufen(Dokumentation: siehe AUTOSAR.pdf). Beispiel:

```
//Falsch:
TASK(Output) // definition siehe Komponentendiagramm
{
        myPrintln("three"); // mit wie in den Ubungen definierter
            Funktion myPrintln
        TerminateTask();
}
// Problem: Applikationscode nicht in Runnables aufgeteilt und direkt
    eine OS-Funktion aufgerufen

//RICHTIG:

void myRunnable(char *s)
{
        RTE_Write_OutputPort_out(s); // bei der Codegenerierung
            entscheiden wir dann, welcher Codeschnipsel fuer diese
            Funktion eingefuegt wird
}

TASK(Output)
{
        myRunnable("three");
        TerminateTask();
}
```

Funktionen um Runnables übersichtlicher zu gestalten folgen im Punkt 1.2. Runnables zeichnen sich außerdem aus durch:

- keinen Rückgabewert
- keine übergebenen Parameter
- keine `while(true)` Schleifen (??)

1.2 Funktionen

Es gibt zwei Arten von Funktionen:

- RTE-Funktionen
- nicht RTE-Funktionen, diese sind nur lokal für eine Runnable nötig

1.2.1 RTE-Funktionen

Alle RTE-Funktionen sind im Dokument AUTOSAR.pdfdefiniert. Es sind nur diese Funktionsköpfe zu nutzen. Es darf keine individuellen Anpassungen geben. Sollte jemand eine Änderung an einer Funktion benötigen(für Applikationscode) ist erst ein *Issue* auf github zu schreiben. Eventuell wird dann eine andere RTE-Funktion zusätzlich verfasst oder eine Änderung vorgenommen. Wenn jemand RTE-Funktionen einfach so ändert, ist er für die Seiteneffekte verantwortlich!

1.2.2 nicht RTE-Funktionen

diese Funktionen zeichnen sich dadurch aus, das sie nichts mit der übrigen Applikation zu tun haben und nur der Übersichtlichkeit innerhalb einer Runnable dienen. Es gibt zwei Möglichkeiten damit umzugehen:

1.) lokal innerhalb der Runnable definieren. Beispiel:

```
void myRunnable()
{
        uint32_t returnbigger(uint32_t a, uint32_t b){
                if(a < b)
                        return b;
                else
                        return a;
        }

        ...

        bigger = returnbigger(a,b);
}
```

2.) global im speziellen file (Input filename), welches dann inkludiert werden muss(eindeutiger Funktionsname mit Beschreibung). Im besten Fall die Funktion durch `#define XX` definieren(Achtung: siehe 1.3). Beispiel:

```
//im File funktionen.h
#define returnbigger(a,b) (((a>b) ? a : b))

//im Runnable
void myRunnable()
{
        bigger = returnbigger(a,b);
}
```

1.3 Defines

Für die allermeisten Fälle in unserem Projekt reicht es Funktionen durch `#define` zu deklarieren(Außerdem wird dann die Codegenerierung tendentiel leichter). Punkte, die zu beachten sind:

- Bei der Deklaration auf die Konkrete Notation achten! siehe Notation
- keine terminierendes ; am Ende des `#define`! Wenn sich die Berechnung über mehrere Zeilen zieht oder einen komplett abgeschlossenen Codeblock darstellt, müssen natürlich die entsprechenden ; vorhanden sein. Unter Umständen ist dann auch ein abschließendes ; nötig. Dies ist deutlich kenntlich zu machen und ordentlich zu dokumentieren!
- Ein `#define` ist GLOBAL. Bitte bei der Namensgebung dann darauf achten, das dieses `#define` eindeutig ist. Wenn lokale Funktionen `#define`t werden, dann den Namen so lange und eindeutig wählen, dass zu 100% keine Seiteneffekte auftreten können!

2 Ordner- und Filestruktur