# Technische Informatik IV: Praktikum

Protokoll zu Aufgabe 7

von Alexander Steen, Max Wisniewski

## Vorbereitung

Für diese Aufgabe haben wir uns mit der Sektion Incoming AT Commands des ADL User Guide auseinandergeetzt.

## Aufgaben

1. AT-Kommando AT+COUNTDOWN erstellen
2. Dieses Kommando erweitern, sodass heruntergezählt wird
3. Erweitern, sodass durch einen Read-Aufruf der ursprüngliche Startwert angezeigt wird
4. Erweitern, sodass durch einen Test-Aufruf die möglichen Startwerte angezeigt werden

## Dokumentation

**adl_atCmdSubscribe** Dieser Befehl erlaubt es einem, selbst geschriebene Funktionen bei Eingabe eines benutzerdefinierten AT-Kommandos auszulösen Die Syntax der Aufrufes ist:

```
s16 adl_atCmdSubscribe (ascii * Kommando, adl_atCmdHandler_t Handler, u16 Optionen)
```

Dabei ist Kommando die Zeichenkette, bei dem die Funktion Handler aufgerufen werden soll. Der Parameter Optionen ermöglicht es einem zu bestimmen, wie das Kommando eingesetzt werden können soll. Der zweite 4-er Block der 16-Byte-Zahl Optionen gibt an, ob das Kommando lesend, testend oder als Aktionen eingesetzt werden kann. Außerdem wird hier noch festgelegt, ob das Kommando Parameter akzeptiert. Der dritte 4-er Block von Optionen beschreibt die maximale, der vierte 4-er-Block die minimale Anzahl an Parametern, falls erlaubt. Für die Optionen existieren zur besseren Lesbarkeit auch Konstanten. Also ergibt sich:

| Bitmuster | Bedeutung | Konstante |
|---|---|---|
| 0000 0001 0000 0000 | Das Kommando akzeptiert Parameter | ADL_CMD_TYPE_PARA |
| 0000 0010 0000 0000 | AT+KOMMANDO=? wird akzeptiert | ADL_CMD_TYPE_TEST |
| 0000 0100 0000 0000 | AT+KOMMANDO? wird akzeptiert | ADL_CMD_TYPE_READ |
| 0000 1000 0000 0000 | AT+KOMMANDO wird akzeptiert | ADL_CMD_TYPE_ACT |

Um Optionen zu kombinieren, nutzt man die bitweise Disjunktion (|).
Der Handler muss dann folgende Form haben:

```
void (*) adl_atCmdHandler_t (adl_atCmdPreParser_t *Params)
```

## Durchführung und Auswertung

```
void countdown(adl_atCmdPreParser_t *parameter) {
    adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
}
```

Dieser Handler kann von einem AT+Kommando angesprochen werden und gibt dann "OK" aus.

```
void main_task(void) {
    adl_atCmdSubscribe("AT+COUNTDOWN", countdown, ADL_CMD_TYPE_ACT);
}
```

Der weiter oben gezeigte Handler wird auf das Kommando AT+COUNTDOWN registriert, sodass bei Nutzung des Kommandos als Aktion, die Funktion countdown angesprungen wird.

```
adl_tmr_t *timer_n;
s16 timerZahl = -1;
s16 paramZahl;
```

Wir speichern den Timer global, und speichern uns unter paramZahl, mit welchem Startwert der Timer das letzte Mal aufgerufen wurde. Mit timerZahl wird die aktuelle Countdown-Sekunde gespeichert, ein Wert von -1 bedeutet, dass gerade kein Timer läuft.

```
void countdownhandler(u8 event, void *context) {
    ascii puffer[20]; // Genug Platz für die Ausgabe
    if (--timerZahl <= 0) {
        adl_atSendResponse(ADL_AT_RSP, "\r\n␣+COUNTDOWN:␣Fertig\r\n");
        // Timer deregistrieren
        adl_tmrUnSubscribe(timer_n, countdownhandler, ADL_TMR_TYPE_100MS);
        timerZahl = -1;
    } else {
        // aktuellen Countdown ausgeben
        wm_sprintf(puffer, "\r\n␣+COUNTDOWN:␣%i\r\n", timerZahl);
        adl_atSendResponse(ADL_AT_RSP, puffer);
    }
}
```

Diesen Countdown-Handler benutzen wir, wenn ein Timer registriert wird. Dieser gibt uns periodisch den aktuellen Countdown bzw. eine Nachricht, falls dieser zuende ist.

```
void countdown(adl_atCmdPreParser_t *parameter) {
    ascii param[3];
    // Bei Action-Abfrage nur OK senden
    if (parameter->Type == ADL_CMD_TYPE_ACT) {
        adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
    }
    // Parameter wurde mit übergeben
    if (parameter->Type == ADL_CMD_TYPE_PARA) {
        wm_strGetParameterString(param, parameter->StrData, 0);
        // Prüfe, ob es eine Zahl ist
        if (wm_isnumstring(param)) {
            paramZahl = wm_atoi(param);
            if (paramZahl >= 0 && paramZahl <= 300) {
                // Bereich stimmt
                if (timerZahl >= 0) {
                    // Ein Countdown läuft noch: Fehler!
                    adl_atSendResponse(ADL_AT_RSP, "\r\n+COUNTDOWN: ERROR\r\n");
                } else {
                    // Countdown anlegen
                    adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
                    timerZahl = paramZahl;
                    // Timer anmelden, zyklisch jede sekunde
                    timer_n = adl_tmrSubscribe(
                        TRUE,
                        10,ADL_TMR_TYPE_100MS, // 10 * 100ms = 1s
                        countdownhandler);
                }
            } else {
                // Parameter außerhalb des Bereiches
                adl_atSendResponse(ADL_AT_RSP, "\r\n +COUNTDOWN: ERROR\r\n");
            }
        } else {
            // Parameter ist keine Zahl: Fehler!
            adl_atSendResponse(ADL_AT_RSP, "\r\n +COUNTDOWN: ERROR\r\n");
        }
    }
}
```

Diesen Kommando-Handler benutzen wir, um auf die Anfragen zu antworten. Diesen registrieren wir jetzt mit:

```
void main_task(void) {
    adl_atCmdSubscribe("AT+COUNTDOWN", countdown, ADL_CMD_TYPE_PARA | ADL_CMD_TYPE_ACT
        | 0x0011);
}
```

Dabei ist unser Kommando sowohl als Aktion als auch mit Parametern (minimal 1, maximal 1) aufrufbar.

**Testlauf**

```
+GSM: Anmeldung im Netz abgeschlossen
AT+COUNTDOWN
OK

AT+COUNTDOWN=20
OK
+COUNTDOWN: 19
+COUNTDOWN: 18
...
+COUNTDOWN: Fertig
```

Um den möglichen Parameterbereich ausgeben zu lassen, ergänzen wir folgende Zeile in unseren Kommando-Handler:

```
    if (parameter->Type == ADL_CMD_TYPE_TEST) {
        adl_atSendResponse(ADL_AT_RSP, "\r\n+COUNTDOWN: (1...300)\r\n");
    }
```

Um dann auch einzubauen, dass der letzte Startwert ausgegeben wird, ergänzen wir den Kommando-Handler so, dass er die globale Variable paramZahl ausgibt, bzw. anzeigt, dass gerade keine Countdown läuft:

```
    if (parameter->Type == ADL_CMD_TYPE_READ) {
        // Kein Countdown läuft
        if (timerZahl == -1)            {
            adl_atSendResponse(ADL_AT_RSP, "\r\n+COUNTDOWN: NONE\r\n");
        } else {
            // String zusammenbauen und ausgeben
            ascii puffer[20];
            wm_sprintf(puffer, "\r\n+COUNTDOWN: %i\r\n", paramZahl);
            adl_atSendResponse(ADL_AT_RSP, puffer);
        }
    }
```

Um jetzt alle neuen Möglichkeiten auch bereitzustellen, registrieren wir jetzt unser Kommando mit allen Optionen:

```
void main_task(void) {
    adl_atCmdSubscribe("AT+COUNTDOWN", countdown, ADL_CMD_TYPE_PARA | ADL_CMD_TYPE_ACT
        | ADL_CMD_TYPE_TEST | ADL_CMD_TYPE_READ | 0x0011);
}
```

**Testlauf**

```
+GSM: Anmeldung im Netz abgeschlossen
AT+COUNTDOWN
OK

AT+COUNTDOWN?
+COUNTDOWN: NONE

AT+COUNTDOWN=?
+COUNTDOWN: (1...300)

AT+COUNTDOWN=20
OK
+COUNTDOWN: 19
+COUNTDOWN: 18
AT+COUNTDOWN?   // Letzten Startwert ausgeben
+COUNTDOWN: 20
+COUNTDOWN: 17
...
AT+COUNTDOWN=20  // Neuen Countdown starten, obwohl noch einer läuft
+COUNTDOWN: ERROR  // produziert einen Fehler
...
+COUNTDOWN: Fertig
```