Was ist dran an Action!?

Im 4. Teil holen wir eine kleine Einführung in die Sprache nach.

Heute kommen wir zur vierten Folge unseres Action!-Centers. Wir haben bereits eine Reihe von Zuschriften und Anrufen von Lesern erhalten, die an den Artikeln im **ATARI***magazin* lebhaftes Interesse zeigten, sich aber unter dem Namen Action! nichts Konkretes vorstellen konnten. Das ist Grund genug für uns, sich dieser Leser anzunehmen und näher auf Grundlagen und Philosophie dieser Sprache einzugehen.

Am besten beginnt man in einem solchen Fall mit der Geschichte. Alles begann 1978, als sich Atari auf der Suche nach einem Basic für die damals in Entwicklung befindlichen Computer der Serien 400 und 800 befand. Das zu diesem Zeitpunkt bereits populäre Microsoft-Basic wollte nicht so recht in das zur Verfügung stehende 8-KByte-ROM der neuen Rechner passen, jedenfalls nicht mit den Grafik- und Sound-Befehlen, die man sich bei Atari vorstellte. Daher wurde die kleine Firma Shepardson Microsystems Inc., die auch schon ein DOS für Apple entworfen hatte, mit einer Neuentwicklung beauftragt.

8 Bit

So entstand das Atari-Basic, welches sich in wenig veränderter Form auch heute noch im 130 XE befindet. Bei dieser Firma arbeitete Bill Wilkinson, der auch schnell erkannte, daß hier das letzte Wort noch nicht gesprochen war. Er gründete zusammen mit Mike Peters die Firma Optimized System Software (OSS), die wohl jedem Atari-Besitzer ein Begriff ist. OSS entwikkelte das diskettenorientierte Basic/A+, das CP/A-DOS, schließlich mit Basic XL die erste Programmiersprache in einer Supercartridge, den MAC/65, das DOS XL und in direkter Folge das Basic XE.

Spezialisiert auf die Produktion von Programmiersprachen, griff man bei OSS auch zu, als Clinton Parker der Firma ein gänzlich neues und vielversprechendes Produkt anbot, das Action! heißen sollte. OSS verpackte diese komplette Programmierumgebung in die damals gerade fertiggestellte Supercartridge, und im August 1983 war es dann soweit. Action! begeisterte die amerikanische Fachpresse, und so bestellte auch ich eine der damals noch poppig orangefarbenen Cartridges für knapp 300 DM. Es war kein Fehlgriff, wie sich in den folgenden Jahren herausstellen sollte.

Clint Parker hatte sich beim Entwurf von Action! nicht nur an bestehenden Standardsprachen wie Pascal, C oder Basic orientiert, sondern nahm lediglich das, was er für positiv hielt. Besonders wichtig war auch, daß sich all dies effektiv auf einem 6502 programmieren ließ. So entwikkelte er einen der schnellsten (wenn nicht überhaupt den schnellsten) Compiler für die 6502-Rechner. Dies bezieht sich sowohl auf die Kompilierzeit als auch auf die erzeugten Programme.

Man kann Action! also getrost als eine Mischung aus vielen Programmiersprachen betrachten. Die Einfachheit der Bedienung ist einem Basic-Interpreter vergleichbar, die Struktur der Sprache erinnert an Pascal, und die Möglichkeit, fast auf Maschinenebene zu programmieren, ist eigentlich eine Spezialität von C. Nicht zuletzt kann man die Geschwindigkeit der erzeugten Programme wohl nur mit Assembler erreichen. Zu all dem gesellte sich der überaus elegante Texteditor, der mit seiner Fähigkeit, zwei Texte gleichzeitig zu bearbeiten, für 8-Bit-Computer Maßstäbe setzte.

Wer bisher nur Basic oder auch Assembler programmierte, muß sich beim Übergang zu Action schon etwas umstellen. Schließlich handelt es sich um eine strukturierte Sprache. Nun sollte aber niemand vor diesem oft strapazierten Ausdruck erschrecken. Eine solche Programmiermethode bringt eigentlich nur Vorteile.

Strukturiert bedeutet, daß ein Programm strikt in Module zu unterteilen ist, die jeweils nur einen Eingang und Ausgang besitzen. Auf diese Weise wird der berüchtigte "Spaghetti-Code" vermieden, der bei allzu sorgloser Programmierung in Basic häufig resultiert. Strukturierte Sprachen erfordern außerdem, daß alle Variablen vor der Verwendung definiert werden.

Wichtig ist auch, daß Action! eigentlich nur aus einer sehr kleinen Anzahl von Befehlen besteht (s. Kasten). Hierzu zählen die Kommandos zur Definition von Variablen (BYTE, CARD und TYPE) sowie zur Ablaufsteuerung (IF, ELSE, WHILE und UNTIL). Nicht zum Grundvokabular gehören jedoch Anweisungen wie PRINT, INPUT, DRAWTO usw. Sie sind nur als Prozeduren (Unterprogramme) oder Funktionen implementiert.

Dieses Konzept wird jedem C-Programmierer sehr bekannt erscheinen; auch dort müssen die I/O-Funktionen als Include-Datei zum Sprachkern hinzugeladen werden. Um den Action!-Usern diese Mühe zu ersparen, wurde eine Library mit den wesentlichsten Befehlen erstellt. Diese Un-

terprogrammbibliothek ist gleich im ROM der Action!-Cartridge enthalten.

Diese Lösung hat nun aber Vor- und Nachteile. Größter Vorteil ist sicherlich, daß beim Kompilieren kein Include-File notwendig ist. Ein Diskettenlaufwerk ist also nicht unbedingt Voraussetzung. Der Nachteil dieser Methode besteht allerdings darin, daß Action!-Programme, die diese Library benutzen, nur mit eingesteckter Cartridge lauffähig sind. Ein geschickter Programmierer kann diesen Mangel jedoch einfach umgehen, indem er die Bibliotheksfunktionen durch eigene ersetzt. Auf dieser Basis gibt es auch ein sogenanntes Run-Time-Package, das Programme vom Steckmodul unabhängig macht. Leider ist es in Deutschland nur schwer erhältlich. Leser der CK-Computer Kontakt konnten jedoch bereits ein einfaches Run-Time-Modul abtippen.

Schauen wir uns zum Abschluß dieses Action!-Centers nun noch einige Beispiele an, die den Unterschied zu Basic verdeutlichen. Eine Endlosschleife würde man in dieser Sprache z.B. so programmieren:

```
10 PRINT "HALLO"
20 GOTO 10
```

In Action! sieht dies so aus:

```
DO
PRINT ("HALLO")
OD
```

Alle Befehle, die zwischen DO und OD (die Umkehrung von DO) eingeschlossen sind, werden endlos wiederholt. Will man eine Begrenzung der Durchläufe erreichen, so gibt es viele Möglichkeiten. Die einzige, die ein Pendant in Basic besitzt, ist die FOR-Schleife:

```
10 FOR i = 1 TO 10
20 PRINT i
30 NEXT i
```

In Action! gibt man folgendes ein:

```
BYTE i
FOR  i = 1 TO 10
     DO
     PRINTBE (i)
     OD
```

Wie bereits erwähnt, muß eine Variable vor der Verwendung definiert werden – daher die BYTE-Deklaration. Interessant ist auch der PRINT-Befehl. PRINTBE ist eine Funktion, die einen BYTE-Wert und ein End-of-Line ausgibt. Bei der Ausgabe von Zahlen ist also genau festzulegen, von welchem Typ die auszugebende Variable ist. Eine andere Verwendung der DO-OD-Blocks wäre:

```
BYTE  s
      DO
      s = Stick (0)
      UNTIL s = 14
      OD
```

Wenn Sie dieses Programm in Basic umsetzen wollen, dann müssen Sie mit einigen GOTO-Befehlen arbeiten, und die klare Struktur geht auch schon verloren:

```
10 S = STICK (0)
20 IF S = 14 THEN 40
30 GOTO 10
40 ...
```

Man könnte dies natürlich auch anders schreiben, aber es geht ja schließlich ums Prinzip. Action!-Programme sind dank der festen Struktur einfach besser lesbar. Ein anderes Stilmittel stellen die Abfragen dar. Während IF-Abfragen in Basic auf eine Zeile begrenzt sind, können sich IF-FI-Blöcke über beliebig viele erstrecken. Da auch Befehle wie ELSE und ELSEIF vorhanden sind, bleibt dem Programmierer so mancher Umweg über GOTO erspart:

```
IF STICK (0) = 14 THEN
        Oben ()
ELSEIF STICK (0) = 11 THEN
        Unten ()
ELSE
        KeineBewegung ()
FI
```

Dieses Beispiel einer einfachen Joystick-Abfrage würde in Basic folgendermaßen lauten:

```
10 IF STICK (0) = 14 THEN
   GOSUB 1000: GOTO 40:
   REM Oben
20 IF STICK (0) = 11 THEN
   GOSUB 2000: GOTO 40:
   REM Unten
30 GOSUB 3000: REM Keine
   Bewegung
40 ...
```

Sie sehen auch, daß durch die Verwendung von Namen die Unterprogramme sich selbst dokumentieren (solange man sinnvolle Bezeichnungen benutzt). Somit kann mancher Kommentar entfallen.

Daneben gestattet Action! auch den direkten Zugriff auf den Speicher, da man Variablen und auch Feldvariablen direkt in bestimmte Zellen legen kann:

```
BYTE  chbas = 756
BYTE  ARRAY COLOR (4) =
      $2C4
```

Mit Hilfe dieser Definitionen können Sie direkt den Zeichensatz oder die Farben verändern. Mit chbas = 204 wird z.B. der internationale Zeichensatz aktiviert (in Basic: POKE 756,204). COLOR (0) = 255 schaltet Farbregister eins auf Weiß um. Solche Konstruktionen machen Action! erstens elegant und zweitens auch sehr schnell. Wenn Sie etwas von Assembler verstehen und sich den erzeugten Code von chbas = 204 ansehen, dann werden Sie folgendes finden:

```
LDA  #204
STA  $2F4
```

Schneller geht es nun wirklich nicht mehr, und damit ist unsere kleine Reise durch das Action!-Steckmodul auch schon wieder beendet.

Peter Finzel

Der ACTION!-Sprachkern verfügt über nur wenige Befehle

Action!-Sprachkern

AND	FI	OR	UNTIL
ARRAY	FOR	POINTER	WHILE
BYTE	FUNC	PROC	XOR
CARD	IF	RETURN	
CHAR	INCLUDE	RSH	
DEFINE	INT	SET	
DO	LSH	STEP	
ELSE	MOD	THEN	
ELSEIF	MODULE	TO	
EXIT	OD	TYPE	