Appendix A: ACTION! Language Syntax

Retyped in by Jim Patchell on February 13, 2010
It should be noted that I have made some additions to and correction to this BNF grammar. I added in the assign ops and also "fixed" the complex rel production. Maybe they were correct in the original, but I have a hard time reading these days. Anyway, the grammar seems to more or less work this way.

The following is the syntax of the ACTION! Language in Backus-Naur form (BNF). This form has a couple of special characters:

Symbol	Meaning
:=	"is defined as"
|	"or"
[]	"optional"

The appendix is set up to allow you easy access to the particular information you want, with subsections as follows:

A.7 statements
- Assignment statement
- EXIT statement
- IF statement
- DO – OD Loop
- UNTIL statement
- WILE Loop
- FOR Loop
- Code Blocks

A.8 Expressions
- Relational Expressions
- Arithmetic Expressions

A.1 ACTION! Constants

Numeric Constant

```
<num const> ::= <dec num> | <hex num> | <char>
<dec num> ::= <digit>
              | <dec num> <digit>
<hex num> ::= '$' <hex digit>
              | <hex num> <hex digit>
<hex digit> ::= <digit> | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
<dec digit> ::= '0' | '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
<char> ::= ' ' <any printable character>
```

String Constant

```
<str const> ::= "" <string> ""
<string> ::= <str char>
           |  <string> <str char>
<str char> ::= <all possible characters except "">
```

Compiler Constant

```
<comp const> ::=  <base comp const>
                | <comp const> '+' <base comp const>
<base comp const> ::= <identifier> |  <num const> | <ptr ref > | '*'
```

A.2 Operators and Fundamental Data types

Operators

```
<special op> ::= AND | OR | '&' | '%'
<rel op> ::= XOR | '!' | '=' | '#' | '>' | '<' | '<>' | '>=' | '<='
<add op> ::= '+' | '-'
<mul op> ::= '*' | '/' | MOD | LSH | RSH
<unary op> ::= '@' | '-'
```

Fundamental Data Types

<fund type> ::= CARD | CHAR | BYTE | INT

A.3 ACTION! Program Structure
<program> ::= <program> MODULE <prog module>
| < [MODULE] <prog module>
<prog module> ::= [<system decls> <routine list>

A.4 Declarations

System Declarations

<system decls> ::= <DEFINE decl> | <TYPE decl> | <var decl>

DEFINE Directive
<DEFINE decl> ::= DEFINE <def list>
<def list> ::= <def>
| <def list> ',' <def>
<def> ::= <identifier> '=' <constant>

TYPE Declaration (for records)

<TYPE decl> ::= TYPE <rec ident list>
<rec ident list> ::= <rec ident list> <rec ident>
| <rec ident>
<rec ident> ::= <rec name> = '[' <field init> ']'
<rec name> ::= <identifier>
<field init> ::= <fund var decl>

Variable Declarations

<var decl> ::= <var decl:> <base var decl>
| < base var decl>
<base var decl> ::= <fund decl>
| <POINTER decl>
| <ARRAY decl>
| <record decl>

Variable Declaration for Fundamental Data Types
<fund decl> ::= <fund decl> < base fund decl>
| <base fund decl>
<base fund cecl> ::= <fund type> <fund ident list>
<fund type> ::= CARD | CHAR | BYTE | INT
<fund ident list> := <fund ident>
| <fund ident list> ',' <fund ident>
<fund ident> ::= <identifier> ['=' <init opts>]
<init opts> ::= <addr> | '[' <value> ']'
<addr> ::= <comp const>

<value> ::= <num const>

Variable Declaration for Pointers

<POINTER decl> ::= <ptr type> POINTER <ptr ident list>
<ptr type> ::= <fund type> | <rec name>
<ptr ident list> ::= <ptr ident>
                    | <ptr ident list> ',' <ptr ident>
<ptr ident> ::= <identifier> ['=' <value>]

Variable Declarations for Arrays

<ARRAY decl> ::= <fund type> ARRAY <arr ident list>
<arr ident list>::= <arr ident>
                    | <arr ident list> ',' <arr ident>
<arr ident> ::= <identifier> ['(' <dim> ')'] ['='<array init opts>]
<dim> ::= <num const>
<arr init opts> ::= <addr> | '[' <value list> ']' | <str const>
<value list> ::= <value>
                  | <value list> <value>
<value>::= <comp const>

Variable Declaration for Records

<record decl> ::= <identifier> <rec ident list>
<rec ident list> ::= <rec ident>
                    | <rec ident list> ',' <rec ident>
<rec ident> ::= <identifier> ['=' <address>]
<address> ::= <comp const>

A.5 Variable References

Memory References

<mem reference> ::= <mem contents> | '@'<identifier>
<mem contents> ::= <fund ref> | <arr ref> | <ptr ref> | <rec ref>
<fund ref> ::= <identifier>
<arr ref> ::= <identifier> '(' <arith exp> ')'
<ptr ref> ::= <identifier> '^'
<rec ref> ::= <identifier> '.' <identifier>

A.6 ACTION! Routines

<routine list> ::= <routine>
                    | <routine list> <routine>
<routine> ::= <proc routine> | <func routine>

Procedure Structured

<proc routine> ::= <PROC decl> [<system decls>][stmt list>][RETURN]
<proc decl> ::= PROC <identifier> ['=' <addr>] '(' [<param decl>] ')'
<addr> ::= <comp const>

Function Structure

<func routine> ::= <FUNC decl> [<system decls>] [<stmt list>][RETURN '(' <arith exp> ')']
<FUNC decl> ::= <fund type> FUNC <identifier> ['=' <addr>] '(' [<param decl>] ')'
<addr> := <comp const>

Routine calls
<routine call> ::= <FUNC call> | <PROC call>
<FUNC call> ::= <identifier> '(' [<params>] ')'
<PROC call> ::= <identifier> '(' [<params>] ')'

Parameters

<param decl> ::= <var decl>
NOTE: maximum of 8 paramters allowed.

A.7 Statements

<stmt list> ::= <stmt>
| <stmt list> <stmt>
<stmt> ::= <simp stmt> | <struct stmt> | <code block>
<simp stmt> ::= assign stmt> | <EXIT stmt> | <routine call>
<struct stmt> ::= <IF stmt> | <DO loop> | <WHILE loop> | <FOR loop>

Assignment Statements

<assign stmt> ::= <mem contents> '=' <arith expr>
| <mem contents> “==+” <arith expr>
| <mem contents> “==-” <arith expr>
| <mem contents> “==/” <arith expr>
| <mem contents> “==*” <arith expr>
| <mem contents> “==&” <arith expr>
| <mem contents> “==%” <arith expr>
| <mem contents> “==|” <arith expr>
| <mem contents> “==LSH” <arith expr>
| <mem contents> “==RSH” <arith expr>
| <mem contents> “==MOD” <arith expr>

EXIT Statement

<EXIT stmt> ::= EXIT

IF Statement

<IF stmt> ::= IF <cond exp> THEN [<stmt list>] [<ELSEIF exten>][<ELSE stmt>] FI
<ELSEIF exten> ::= ELSEIF <cond exp> THEN [<stmt list>]
<ELSE exten> ::= ELSE [<stmt list>]

DO – OD Loop

<DO loop> ::= DO [<stmt list>] [<UNTIL stmt>] OD

UNTIL statement

<UNTIL stmt> ::= UNTILE <cond exp>

WHILE Loop

<WHILE loop> ::= WHILE <cond exp> <DO loop>
FOR Loop

<FOR loop> ::= FOR <identifier> '=' <start> TO <finish> [STEP <inc>] <Do loop>
<start: ::= <arith exp>
<finish> ::= <arith exp>
<inc> ::= <arith exp>

Code Blocks

<code block> ::= '[' <comp const list>']'
<comp const list> ::= <comp const>
                    | <comp const list> <comp const>

A.8 Expressions

Relational Expressions

<complex rel> ::= <complex rel> <special op> <simp rel exp>
                | <simp rel exp>

<simp rel exp> ::= arith exp> <rel op> <arith exp>
                | <arith exp>

Arithmetic Expressions

<arith exp> ::-= <arith exp><add op> <mult exp>
                | <mult exp>

<mult exp> ::= <mult exp><mult op><value>
                | <value>

<value> ::= <num const> | <mem reference> | '(' <arith exp> ')'