ditional compilations and executions are performed (for example, as is normally done from teletype) then the @MAP control statement must be included before @XQT for all executions after the first. Similarly, the <specl> field must be used in the @SIMPL statements for programs that consist of separately compiled components (see 4.4).

Appendix II - Precedence of Operators

The SIMPL-T operators are listed below in order of precedence from highest to lowest.

[]	part
.C. .NOT. - (unary)	unary
.RA. .RL. .LL. .LC.	shift
.A. .V. .X.	bit logical
* / + - (binary)	arithmetic
= <> < > <= >=	relational
.AND. .OR.	logical
.CON.	string

Appendix III - ASCII Character Codes

Character	Octal Code	Character	Octal Code
space	40	;	73
!	41	<	74
"	42	=	75
#	43	>	76
$	44	?	77
%	45	@	100
&	46	A-Z	101-132
'	47	[	133
(	50	\	134
)	51	]	135
*	52	Δ	136
+	53	≠	137
,	54	a-z	141-172
-	55	{	173
.	56	}	175
/	57	~	176
0-9	60-71	DEL	177
:	72		

Appendix IV - Formal Specification of SIMPL-T Syntax

1. Program

```
<program> ::= {<declaration list>} {<segment list>} start {<identifier>}
```

2. Declarations

```
<declaration list>::= {<declaration list>}  <declaration>
<declaration>     ::= <variable declaration>|
                      <structure declaration>|
                      <external declaration>

<variable declaration>::= <integer declaration>|
                          <string declaration>|
                          <char declaration>

<integer declaration>::= {entry} int {<int dec list>}
<int dec list>::= {<int dec list>,} <int dec item>
<int dec item>::= <identifier> {=<signed constant>}

<string declaration>::= {entry} string <string dec list>
<string dec list>::= {<string dec list>,} <string dec item>
<string dec item>::= <identifier> [<constant>] {=<string constant>}

<char  declaration>::={entry} char <char dec list>
<char dec list>::= {<char dec list>,} <char dec item>
<char dec item>::= <identifier> {=<char constant>}

<structure declaration>::= <array declaration>
<array declaration>::= <int array declaration>|
                       <string array declaration>|
                       <char array declaration>

<int array declaration>::= {entry} int array <int array dec list>
<int array dec list>::= {<int array dec list>,} <int array dec item>
<int array dec item>::= <identifier> (<constant>) {=(<int array init list>)}
<int array init list>::= {<int array init list>,} <int array init item>
<int array init item>;:= <signed constant> {(<constant>)}
```

<string array declaration>::= {entry} string array <string array dec list>
<string array dec list>::= {<string array dec list>,} <string array dec item>
<string array dec item>::= <string spec> (<constant>) {=(<string array init list>}
<string spec>::= <identifier> [<constant>]
<string array init list>::= {<string array init list>,} <string array init item>
<string array init item>::= <string constant> {(<constant>)}

<char array declaration>::= {entry} char array <char array dec list>
<char array dec list>::= {<char array dec list>,} <char array dec item>
<char array dec item>::= <identifier>(<constant>) {=<char array init list>)}
<char array init list>::= {<char array init list>,} <char array init item>
<char array init item>::= <char constant> {(<constant>)}| <string array init item>

<external declaration>::= ext {other} proc <external segment list>|
ext {other} <type> func <external segment list>|
ext int <identifier list>|
ext string <string spec list>|
ext char <identifier list>|
ext int array <array list>|
ext string array <string array list>|
ext char array <array list>
<external segment list>::= {<external segment list>,} <identifier> {(<type list>)}

<identifier list>::= {<identifier list>,} <identifier>
<string spec list>::= {<string spec list>,} <identifier> {[<constant>]}
<array list>::= {<array list>,} <identifier> {(<constant>)}
<string array list>::=
{<string array list>,} <identifier> {[<constant>]}{(<constant>)}
<type>::= int | string | char
<type list>::= {<type list>,} <type list item>
<type list item>::= {<type call>} <type> | <type> <type struct>
<type struct>::= array
<type call>::= ref

3. Program Segments

<segment list>::= {<segment list>} <segment definition>

```
<segment definition>::= <proc definition> | <func definition>

<proc definition>::= <proc heading> <segment body> {return}
<func definition>::= <func heading>{<segment body>}{return (<expr>)}
<proc heading>::= {other} {entry} {rec} proc <identifier> {(<parameter list>)}
<segment body>::= {<local declaration list>} <statement list>
<func heading>::=
        {other} {entry} {<rec>}<type> func <identifier>{(<parameter list>)}
<parameter list>::= {<parameter list>,} <parameter>
<parameter>::= {<type call>} <type> <identifier> | <type> array <identifier>

<local declaration list>::= {<local declaration list>,} <local declaration>
<local declaration>::= <local variable declaration> |
                       <local structure declaration> |
                       <external declaration>
<local variable declaration>::= <type> <identifier list>

<local structure declaration>::= int array <array bound list>|
                                 string array<string array bound list>|
                                 char array<array bound list>
<array bound list>::= {<array bound list>,} <identifier> (<constant>)
<string array bound list>::=
                {<string array bound list>,} <string spec> (<constant>)
```

4. Statements

```
<statement list>::= {<statement list>} <statement>
<statement>::= <assign stmt> | <if stmt> | {\<exit designator>\} <while stmt> |
        <case stmt> | <call stmt> | <exit stmt> | <return stmt>

<assign stmt>::= <extended int variable>::= <expr>|
                 <extended string variable>::= <string expr>|
                 <char variable>::= <char expr>

<if stmt>::= if <expr> then <then clause> {else <else clause>} end
<then clause>::= <statement list>
<else clause>::= <statement list>
```

```
<while stmt>::= while <expr> do <while clause> end
<while clause>::= <statement list>

<case stmt>::= <integer case stmt> | <char case stmt>
<integer case stmt>::= case <expr> of <case list> {else <else clause>} end
<case list>::= {<case list>,} <case form>
<case form>::= <case designator> <statement list>
<case designator>::= {<case designator>} \<integer>\
<char case stmt>::=
        case <char expr> of <char case list> {else <else clause>} end
<char case list>::= {<char case list>,} <char case form>
<char case form>::= {<char case designator>} <statement list>
<char case designator>::= {<char case designator>}  \<char constant>\

<call stmt>::= call <identifier> {(<actual parameter list>)}
<actual parameter list>::= {<actual parameter list>,} <actual parameter>
<actual parameter>::= <expr> | <string expr> | <char expr> | <array identifier>

<exit stmt>::= exit {(<exit designator>)}
<return stmt>::= return {(<expr>)}
<exit designator>::= <identifier>
```

5. Expressions

```
<expr>::= {<expr> .OR.} <logical product>
<logical product>::= {<logical product> .AND.} <relation>
<relation>::= {<relation> <relational op>} <simple expr>|
              <string expr> <relational op> <string expr>|
              <char expr> <relational op> <char expr>
<simple expr>::= {<simple expr> <add op>} <term>
<term>::= {<term> <mult op>} <bit sum>
<bit sum>::= {<bit sum> <bit or op>} <bit product>
<bit product>::= {<bit product> .A.} <shift>
<shift>::= {<shift> <shift op>} <factor>
<factor>::= <unary op> <factor> | <part primary>
<part primary>::= <primary>  {<part designator>}
<primary> ::= <constant> | <variable> | <function designator> | (<expr>)
```

```
<relational op>::= = | <> | > | < | >= | <=
<add op>::= + | -
<mult op>::= * | /
<bit or op>::= .V. | .X.
<shift op>::= .RA. | .RL. | .LL. | .LC.
<unary op>::= .NOT. | .C. | -
<part designator>::= [<first bit> {,<bit count>}]
<first bit>::= <expr>
<bit count>::= <expr>

<function designator>::= <identifier> {(<actual parameter list>)}
<variable>::= <identifier> {(<expr>)}
<constant>::= <integer> | <binary> | <octal> | <hexadecimal>
<signed constant>::= {-} <constant>
<array identifier> ::= <identifier>
<integer>::= {<integer>} <digit>
<digit>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<binary>::= B'<binary form>{<trailing zeros>}'
<binary form>::= {<binary form>}<binary character>
<binary character>::= 0 | 1
<octal>::= 0'<octal form>{<trailing zeros>}'
<octal form>::= {<octal form>}<octal character>
<octal character>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<hexadecimal>::= H'<hexadecimal form>{<trailing zeros>}'
<hexadecimal form>::= {<hexadecimal form>}<hex character>
<hex character>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
    A | B | C | D | E | F
<trailing zeros>::= Z<integer>

<identifier>::= {<identifier>} <letter> | <identifier> <digit>
<letter>::= A | B | C | D | E | F | G | H | I | J | K | L | M |
   N | O | P | Q | R | S | T | U | V | W | X | Y | Z | $
```

<string expr>::= {<string expr>.CON.}<string part primary>

<string part primary>::= <string primary>{<substring designator>}

<string primary>::= <string>|<string function designator>|(<string expr>)|<char expr>

<substring designator>::= [<first char>{,<char count>}]

<string>::= <string variable>|<string constant>

<string variable>::= <identifier>{(<expr>)}

<string constant>::= '<string form>'|''

<string form>::= {<string form>}<character>

<first char>::= <expr>

<char count>::= <expr>

<string function designator>::= <function designator>

<char expr> ::= <char constant>|<char variable>|<char function designator>

<char constant>::= "<character>"

<char variable>::= <identifier>{(<expr>)}

<character>::= any single legal character

<char function designator>::= <function designator>

<extended int variable>::= <variable>{<part designator>}

<extended string variable> ::= <variable>{<substring designator>}

Appendix V - Keywords

The following are reserved keywords and may not be used as identifiers in a SIMPL-T program.

ARRAY	DO	EXT	OF	RETURN
CALL	ELSE	FILE	OTHER	START
CASE	END	FUNC	PROC	STRING
CHAR	ENTRY	IF	REC	THEN
DEFINE	EXIT	INT	REF	WHILE

Appendix VI - Intrinsic Procedures and Functions

An intrinsic (built-in) procedure or function identifier as well as an I/O control parameter identifier (EJECT , SKIP , etc.) may be redefined by the user if desired. Note that if an intrinsic identifier is redefined by the user, then its intrinsic meaning is lost. An intrinsic that is redefined by a local declaration is lost only to the segment containing such a local redefinition, however.

1. Intrinsic Procedures

(An intrinsic procedure call need not include the keyword CALL .)

Name	Section	Arguments	Function
ABORT	4.1.3	none	terminate program abnormally
CLOSEOBJ	5.9.6	{<loc ctr>,<offset>}	relocatable output
DEFEP	5.9.3	<name>, <loc ctr>, <offset>	relocatable output
DEFLC	5.9.5	<loc ctr>, <size>	relocatable output
DEFXREF	5.9.3	<name>, <number>	relocatable output
ENDFILE	5.6.5	<file>	generate end-of-file
GENOBJ	5.9.4	<item>, <lc>, <offset> {,<rlc>{,<xref>}}	relocatable output
OPENOBJ	5.9.2	<loc ctrs>, <xrefs>	relocatable output
PACK	5.1.6	<char array>, <string>	pack char array into string
PRCLOSE	5.10		close processor
PROPEN	5.10		open processor
READ	2.7.1 3.9 5.1.7	see indicated sections	stream input
READC	5.5.2	see 5.5.2	record input
READF	5.6.3	<file>, <item list>	file input
REWIND	5.6.5	<file>	reposition file
UNPACK	5.1.6	<string expr>, <char array>	unpack chars of string into array
WRDATA	5.11	<string>	symbolic output
WREND	5.11		close symbolic output
WRITE	2.7.2 3.9 5.1.7	see indicated sections	stream output
WRITEF	5.6.4	<file>, <item list>	file output
WRITEL	5.5.3	see 5.5.3	record output

2. Integer Functions

Name	Section	Arguments	Result
DIGIT	5.1.6	<char expr>	indicates whether char is a digit
DIGITS	3.11.2	<string expr>	indicates whether all chars of string are digits
EOI	2.7.1	none	indicates whether all items have been read
EOIC	5.5.2	none	indicates whether all records have been read
EOIF	5.6.3	<file>	indicates whether all file items have been read
INTF	3.11.2 5.1.6 5.2	see indicated sections	argument converted to integer
INTVAL	5.1.6	<char expr>	ASCII binary value of char
LENGTH	3.11.2	<string expr>	(current) length of string
LETTER	5.1.6	<char expr>	indicates whether char is a letter
LETTERS	3.11.2	<string expr>	indicates whether all chars are letters
MATCH	3.11.2	<string expr>$_1$, <string expr>$_2$	position of string$_2$ in string$_1$

3. String Functions

Name	Section	Arguments	Result
STRINGF	3.11.2 5.1.6 5.2	see indicated sections	argument converted to string
TRIM	3.11.2	<string expr>	string with trailing blanks removed

4. Character Functions

Name	Section	Arguments	Result
CHARF	5.1.6	see 5.1.6	argument converted to character
CHARVAL	5.1.6	<integer expression>	inverse of INTVAL