

VoRDE V-Module

Neda Document Number: 103-102-05

Last Updated: 2000/03/23 18:31:31

Doc. Revision: 1.1.1.1

Neda Communications, Inc.
17005 SE 31st Place
Bellevue, WA 98008

June 15, 1999

Chapter 1

V-Module

1.1 V-Compiler

The following sections describe the V language.

1.1.1 Lexical Elements

This section describes the lexical structure of the V language; that is, the characters that may appear in a V source file and how they are collected into lexical units, or tokens.

The blank, newline and formatting characters are known collectively as white space characters. These characters are ignored except insofar as they are used to separate adjacent tokens or when they appear in string constants. White space characters may be used to lay out a V program in a way that is pleasing to a human reader.

Comments

A "comment" in V is identical to a comment in C. A comment in a V program begins with the characters `'/*'` and ends with the first subsequent occurrence of the characters `'*/'`. Comments may contain any number of characters and are always treated as white space. For example the program presented in Figure (2) contains four comments.

The preprocessor also treats comments as white space. That is, the preprocessor does not look inside comments for commands or for macro invocations, nor do line breaks inside comments terminate preprocessor commands.

Nested comments are not supported. The same effect as nested comments can be achieved with the preprocessor's conditional commands. In particular,

```
#if 0
...
#endif
```

will effectively "comment out" any section of a program. Refer to "The V Preprocessor" section for more details.

Reserved Words

The following words are reserved in the V language.

appdef	basedir	answer	play0
play1	key	nokey	otherkey
endplay	break	hangup	assign

A reserved word may be defined as a preprocessor macro name. This is not recommended.

1.1.2 The V Preprocessor

The V preprocessor is the same as the C preprocessor. The V preprocessor is a simple macro processor that processes the source text of a V program before the compiler proper parses the source program. The preprocessor is actually a separate program that reads the original source file and writes out a new "preprocessed" source file that can then be used as input to the V compiler.

"vpp" is the V preprocessor program. vpp is a port of Decus CPP to MSDOS. Decus CPP is public domain software. Appendix MAN-APDX contains a manual page for this program.

Preprocessor Commands

The preprocessor is controlled by special preprocessor command lines, which are lines of the source file beginning with the character '#' in column one. Note that the character '#' has no other use in the V language. Lines that do not contain preprocessor commands are called lines of source program text. The standard preprocessor commands are:

1. "#define" Define a preprocessor macro.
2. "#undef" Remove a macro definition.
3. "#include" Insert text from another file.
4. "#if"
 - Conditionally include some text, based on the value of a constant expression.
5. "#ifdef"
 - Conditionally include some text, if a macro name is defined.
6. "#ifndef"
 - Conditionally include some text, if a macro name is not defined.
7. "#else"
 - Alternatively include some text, if the previous #if, #ifdef, or #ifndef condition was false.
8. "#endif" Terminate conditional text.
9. "#line" Supply a line number for compiler messages.
10. "#elif"
 - Alternatively include text based on the value of another constant expression.

The preprocessor removes all preprocessor command lines from the source file and make additional transformations on the source file as directed by the commands.

1.1.3 Organization of V

A V program is a set of V application declarations, V application definitions, and V application assignments.

We will use YACC syntax notation to describe the V syntax. References can be consulted for a precise definition of this syntax. YACC syntax resembles Backus-Naur Form (BNF).

The syntax below shows the organization of a V-Program.

```
.PProgram      : vProgramElement
                | vProgram vProgramElement
                ;
.PProgramElement : vApplication
                | channelAssignment
                ;
vApplication    : applicationDeclaration appBaseDir statementList
                ;
applicationDeclaration : "appdef" string
                ;
appBaseDir      : "basedir" string
                ;
statementList   : statement
                | statementList statement
                ;
channelAssignment : "assign" number string
                ;
```

¡number¿, is a decimal integer. ¡string¿ is a (possibly empty) sequence of characters enclosed in double quotes. ¡statement¿ is described in the "V Statement" section of this publication.

V Application

```
vApplication    : applicationDeclaration appBaseDir statementList
                ;
applicationDeclaration : "appdef" string
                ;
appBaseDir      : "basedir" string
                ;
```

A V Application (vApplication) is an application declaration, followed by the declaration of a base directory for the application followed by a list of V statements. Application Declaration consists of the keyword "appdef" followed by a string (application-name). Application-name identifies the application. It can be used to assign the application to one or more Phone-Port. A base directory can be specified, all voice messages within this application will be selected from this base directory.

Channel Assignment

```
channelAssignment : "assign" number string
                ;
```

After an application has been declared and defined, it can be assigned to one or more Phone-Ports. Once an application has been assigned to a Phone-Port, it will perform the statements defined by the application.

1.1.4 V Statement

The following are valid statements in V.

answer

```
answerStatement : "answer"
                ;
```

The answer statement results in the Phone-Port going Off-Hook, once the specified number of rings has been detected.

play0

```
play0Statement : "play0" string statementList
               ;
```

`;string;` specifies the name of a voice message. The play0 statement results in the specified voice message being played over the Phone-Port. The voice message will be played until the end of the voice message is reached. All DTMFs (Dual Tone Multiple Frequency) detected during this statement are ignored.

play1

```
play1Statement : "play1" string statementList "endplay"
               ;
noKeyLabel    : "nokey"
               ;
keyLabel      : "key" string
               ;
otherKey      : "otherKey"
               ;
breakStatement : "break"
               ;
```

`;string;` specifies the name of a voice message. The play1 statement results in the specified voice message being played over the Phone-Port. The voice message will be played until the end of voice message is reached or a DTMF is detected. The play1 statement is a multiway branch based on the value of an input. Implementation of the play1 statement in V is very similar to the implementation of the switch statement in C. It is a "computed goto". noKeyLabel, keyLabel and otherKey labels are used as local labels that control can be transferred to.

A play1 statement is executed as follows:

1. The voice message is played until the end of message is reached or a DTMF is detected.
2. If no DTMF was detected, program control is transferred to the point indicated by the noKeyLabel as if by a goto statement. The statement labeled by the noKeyLabel is executed next.

3. If a DTMF was detected, and it was equal to a value specified by one of the keyLabels, then program control is transferred to the point indicated by the keyLabel as if by a goto statement. The statement labeled by the keyLabel is executed next.
4. If a DTMF was detected, and it was not equal to any value specified by the keyLabels and there is an otherKeyLabel, then program control is transferred to the point indicated by the otherKeyLabel as if by a goto statement. The statement labeled by the otherKeyLabel is executed next.
5. If a DTMF was detected, and it was not equal to any value specified by the keyLabels and there is no otherKeyLabel, then no statement of the body of the play1 statement is executed. Program control is transferred to whatever follows the play1 statement.

After control is transferred to a nokey, key or otherkey label, execution continues through successive statements, ignoring any additional nokey, key, otherkey labels that are encountered, until the end of the play1 statement is reached, or control is transferred out of the play1 statement by a break statement.

hangup

```
hangupStatement : "hangup"
                ;
```

The hangup statement results in the Phone-Port going On-Hook.

1.2 V Application Examples

The following sections describe the implementation of simple Voice Response applications in V.

1.2.1 Snow Reports

There are a few Ski Resorts in the Seattle area. Some of the more famous ski resorts are Crystal Mountain, White Pass, and Stevens. Let us develop a simple voice response application that prompts the caller for a choice of available reports and depending on the caller's input provides the snow report for the selected ski resort. Let this Voice Response Application be called "snow". Let the directory under which all voice messages related to this application reside be called "c:/snow". Let the voice message that prompts the caller for a selection be called "choose.vox". Let the voice message that thanks the caller be called "thankyou.vox". Let the voice message that contains the snow report for Crystal resort be called "crystal.vox". Let the voice message that contains the snow report for Stevens resort be called "stevens.vox". Let the voice message that contains the snow report for White resort be called "white.vox".

If the caller does not enter a touch-tone key or if an unexpected touch-tone key is detected, reports for all three resorts will be played.

Figure (2) is a V program that implements the above mentioned voice response application.

```
/*
 * File Name: snow.v
 * Snow Report Application.
 * The user is prompted for a number of Ski resorts,
 * Once a selection is made the snow condition for
 * that resort is reported.
```

```

*
*/
#define SNOW_APP "snow" /* Pre-Processor Commands */
#define SNOW_DIR  "/snow"

appdef SNOW_APP
basedir SNOW_DIR
answer
play1 "choose.vox"
key "1"
    play0 "white.vox"
    break
key "2"
    play0 "stevens.vox"
    break
key "3"
    play0 "crystal.vox"
    break
nokey
otherkey
    /* nokey and otherkey do the same thing */
    play0 "white.vox"
    play0 "stevens.vox"
    play0 "crystal.vox"
    break
endplay
play0 "thankyou.vox"
hangup

```

Once an application has been defined, it can be assigned to one or more Phone-Port. This can be done in a file other than the one that contains the definition of the application. Several applications can independently be executed within the system. Let the name of the file that contains the definition of snow application be "snow.v". Figure (3) presents a file that includes "snow.v" and then assigns the snow application to Phone-Ports one and four.

```

/*
* File Name: config.v
*
* Include the applications of interest and
* assign them to telephone ports.
*/

#include "snow.v"
assign 1 SNOW_APP
assign 4 SNOW_APP

```