

FIDE

Language Reference

(v1.6)

© 2008-2009 AIS Cube. All rights reserved.

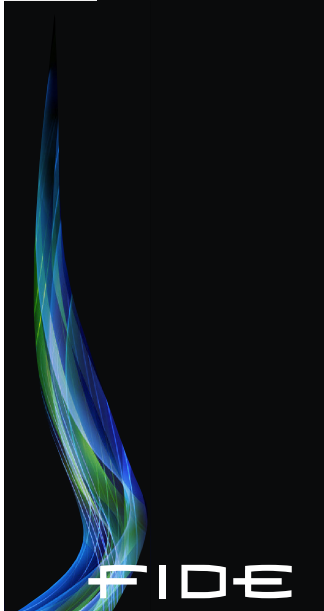
The FlamingICE(FI) and FIDE are either registered trademarks or trademarks of AIS Cube in Singapore and/or other countries.

Microsoft, Windows, Visual Basic and Visual C# are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Adobe and Acrobat are trademarks of Adobe Systems Incorporated.

Quick Overview

FIDE is a Windows-based Integrated Development Environment. It is an IDE for the FlamingICE(FI) Chip.



With the **FIDE** you can:

- Create Projects for the FI chip
- Create Modules for the FI chip
- Write code in Visual Basic™ for your Project Modules
- Compile and Download your project to the FI Chip

FIDE is a user-friendly and intuitive environment:

- Code Editor features Syntax Highlighting, Intellisense, Code Completion, Code Assistant, Parameters Assistant, Auto Correction for common typos and Code Templates.
- Error Window displays all errors detected during compiling and linking.
- Help files are syntax and context sensitive.

As with any modern Windows-oriented program, you may customize the layout of **FIDE** to best suit your needs.

Program Organization

FIDE Projects

A **project** is the collection of files you use to build an application. To create a project, start **FIDE** and select **New Project**.

A **project** consist the following:

- One or more **Modules**
- One or more **Sub Functions**
- **Controls** (Optional)
- **Components** (Optional)

The **FIDE** will create a folder that with the same name as the project name. All files pertaining to this **project** will be saved in this folder. The project file will have an extension “.FIF”. Other information of the **project** is stored in a file with the extension “.dtC”

After all of the components and controls in a **project** have been assembled and the code written, compile and download your **project** to the **Flaming ICE(FI)** chip.

Modules

A FIDE project must have at least one module. The module is where all the source code resides. Modules consist of:

- Declaration statement
- Procedures - Subs and Functions

The module file has an extension “.vbC”.

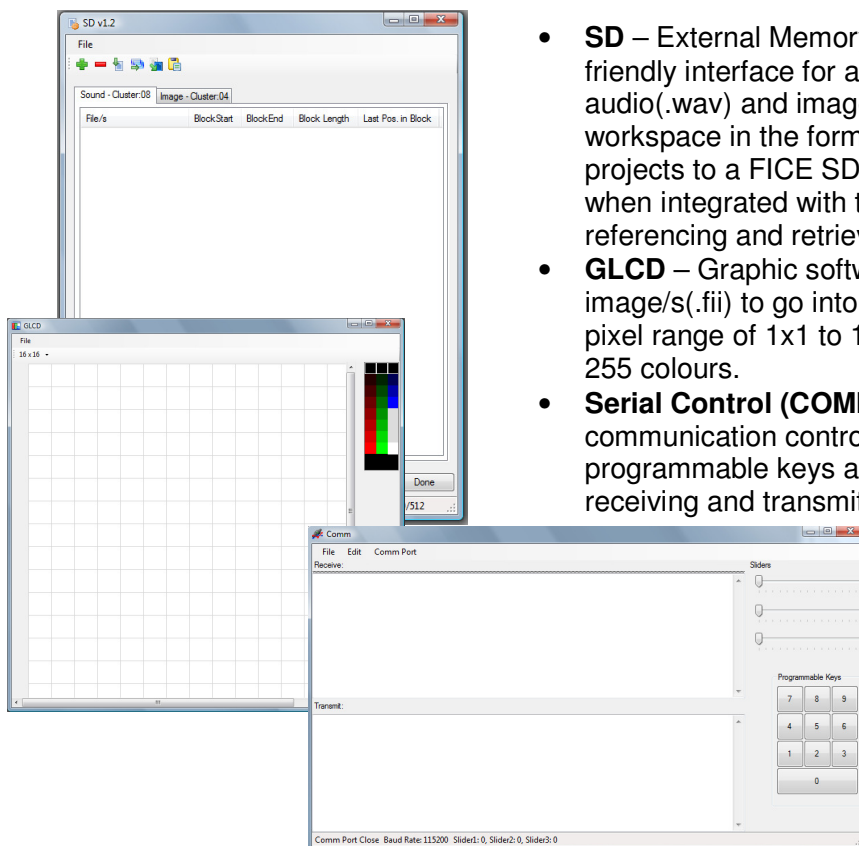
Modules allow you to manage your code better when writing a large program. It allows you to reuse and share your code, making it easy to control the visibility of your subprograms and data.

A FIDE project must have a procedure call Main() in one of its module. The Main() procedure is where the program starts.

FIDE Tools

FIDE features and boasts of a range of tools that provide users visual control over areas pertaining to getting your project up and running quickly and easily.




Tools available:



- **SD** – External Memory Tool featuring a user-friendly interface for adding and deleting audio(.wav) and image(.fii) files, saving workspace in the form of a project and writing projects to a FICE SD card formatted for use when integrated with the FI chip for easy referencing and retrieval of data.
- **GLCD** – Graphic software for designing the image/s(.fii) to go into your graphic LCD with a pixel range of 1x1 to 128x128 that supports 255 colours.
- **Serial Control (COMM)** - Complete serial communication control through user programmable keys and sliders, making receiving and transmitting data an easy task.

SD Tool

Features

- Add and delete sound(.wav) and image(.fii) items
 - Multi-select supported
- Save and load workspace as a project
-  Writing to the FICE SD card
-  Define specific cluster saving
- View detailed information of size, start, end, and last position of files in external memory for easy calling of files during programming
-  Copy entire tab information to clipboard
 - Paste contents to excel or any other editor for easy reference when programming. (Works best in excel.)

Writing to external memory (FICE SD card)

External Memory Drive must be set before commencing write operation. This can be done in **FIDE > Project > External Memory Setting.**

Example,



In this case, the drive would be 'H'.

Formatting the FICE SD

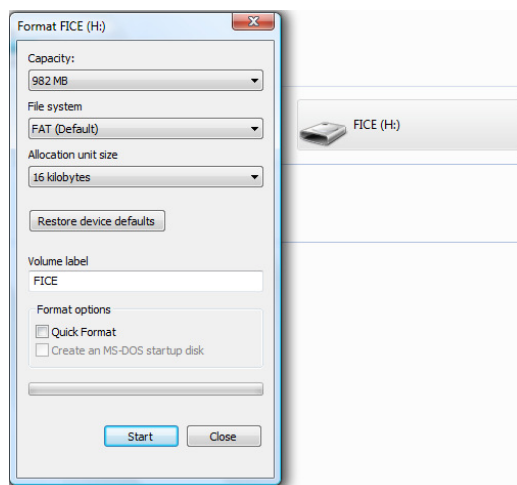
Note: This feature is only available to SD tool (v1.2) onwards

The FICE SD is shipped 'ready-to-use' with preloaded soundtracks, numbers, alphabets and the first 100 commonly used English words in 16Khz 16 bit Mono wave file formats.¹ However, should a need arise to reformat the FICE SD card, please take note and follow the instructions below carefully. You will need to do the actual formatting yourself.

Here's how:


Windows Explorer > My Computer > Right Click (in this case) FICE (H:) > Format...

You should see something like the figure below



File System: FAT
Allocation Unit Size: 16kb.
Volume Label: FICE

Once you're ready, click 'Start'.

Upon successful formatting of the drive, open the SD tool from FIDE, click on the "SD image transfer"  icon and you're done.

FICE SD: The FICE SD card consists of 53 clusters of 16MB each, with a FAT header area of 8192bytes in each cluster. Data storage starts after the 16-sector FAT header. Each cluster consists of 32768 sectors of 512bytes. FI OS takes clusters 0-3 and these clusters are read-only. Users will not be able to write into it. Images are stored in the range of clusters 4 to 7, while audio is stored in the range of clusters 8 to 52.

FICE SD Preloaded Files¹:

Cluster 8 = 2 songs

Cluster 9 = Numbers

Cluster 10 = Alphabets

Cluster 11 = Hundred words (refer to APPENDIX A)

GLCD Canvas

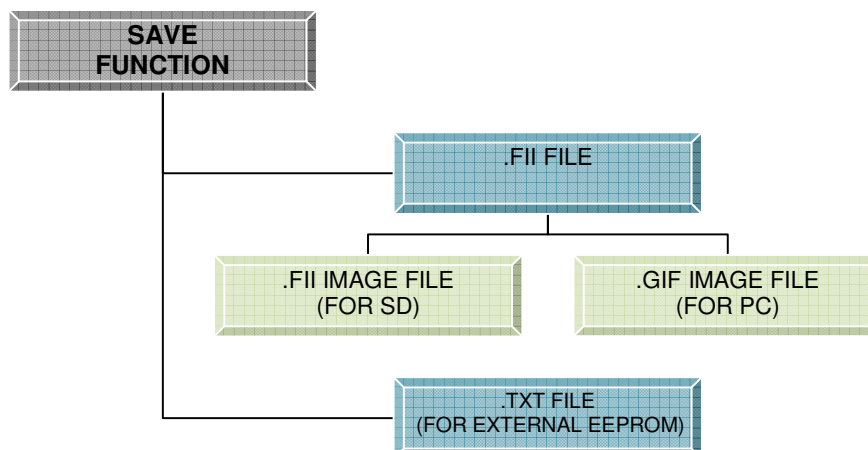
- (v1.4 onwards)

Features

- Copy and paste to and from external image sources
- Loads using an image file (*GIF, PNG, JPEG, BMP, TIFF...*)
- Converts image to 8bpp equivalent automatically as would you would see portrayed in the GLCD
- Uses GIF Colour Palette on screen
- Code generator for External EEPROM array declarations.

Saves image in 3 formats

Saving in .fii would automatically generate 2 files, the .fii image for SD storage, and the .gif image file for loading up again in the tool the next time around. Saving in text format generates a .txt file containing the colour values in auto-generated 16bit word code array for easy copy and paste operations to the FIDE CodeData Module, which will be written to the External EEPROM for a quick preview of the image on the GLCD.



Embedded Palette: 8-bit RGB palette in BGR format (**BBB/GGG/RR**).

Data COMM (Serial Control)

Features

- Receive and transmit data through serial port
- 10 user-programmable keys
- Save and load hotkey sessions
- 3 sliders that sends out data every 100ms
 - Especially useful for servo control
- Slider value range: 0 – 255
- Define COM port and baud rate values
- Open and close COM port
- Hexadecimal, Decimal and Ascii representations of data received and transmitted

Language Reference

Comment format

An apostrophe is used to denote a comment.

Example:

```
I = 32767 ' Comment
```

All characters to the right of the apostrophe are usually ignored by the compiler, unless the apostrophe is embedded inside a string literal.

Identifiers

FIDE identifiers must start with a letter, and all other characters must be letters, digits or underscores.

An identifier can be up to 255 characters long, and all characters are significant. Identifiers are not case sensitive. For example, identifiers xyz, XYZ and xYz are equivalent.

Sub Main

FIDE project must have a procedure "Main()" in the project. This is starting point of the project.

Example:

```
Public Sub Main()  
  
End Sub
```

Procedures

Procedures and functions, referred to collectively as routines, are self-contained statement blocks that can be called from different locations in a program.

Function is a routine that returns a value when it is executed. Procedure is a routine that does not return a value.

Once these routines have been defined, you can call them once or multiple times. Procedure is called upon to perform a certain task, while function is called to compute a certain value.

Sub procedures

```
[Private|Public] Sub procedure_name (arguments)
    [statements]
End Sub
```

You can exit a procedure by using an Exit Sub statement.

Example:

```
Private Sub GetPosition(ByRef X As Byte)
    ReadDevice X
    If (X > 100) Then
        Exit Sub
    End If
    X = X + X
End Sub
```

Functions

```
[Private|Public] Function function_name (arguments) As type
    [statements]
    [Return Variable/Value]
End Function
```

The function returns a value. This is achieved by using the 'Return' command followed by the value to return within the Function.

You can also exit a function by using an Exit Function statement.

Example:

```
Public Function F(ByVal i As Integer) As Integer
    If (i = 3) Then
        Return 92
        Exit Function
    End If
    Return i + 1
End Function
```

Structures

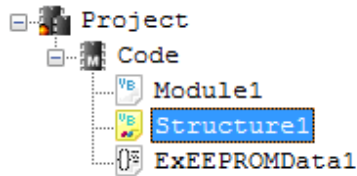
```
[Public | Private] Structure structname
    [Public | Dim] variable As type
End Structure
```

A structure declaration starts with the `Structure` Statement and ends with the `EndStructure` statement. The Structure statement supplies the name of the structure, which is also the identifier of the data type the structure is defining. Other parts of the code can use this identifier to declare variables, parameters, and function return values to be of this structure's data type.

The declarations between the `Structure` and `EndStructure` statements define the members of the structure.

NOTE: Structures in FIDE are declared in a self contained structure file. To use structures in FIDE, you will first have to add a FIDE VB Structure File to the project.

Project > Add New Structure



Example:

```
'Structure1

Public Structure struct_MyStructure
    Public Var1 As Integer
    Public Var2 As Integer
    Public Var3 As Integer
End Structure
```

Once you're done declaring your structure in the structure file, you can make use of it in the Modules by declaring variables of your structure type.

Example:

```
'In Module1

Dim MyStructure As struct_MyStructure

Public Sub Main()
    'assigning values to the variable in the structure
    MyStructure.Var1 = 100
End Sub
```

Syntax Helper

```
0 'Module1
1
2 Dim MyStructure As struct_MyStructure
3
4 Public Sub Main()
5
6 MyStructure.|
7
8 End Sub
9
10
```

A syntax helper popup window is shown over the code. It contains the following text: 'Public Var1 As Integer', 'Public Var2 As Integer', and 'Public Var3 As Integer'. The popup has a light blue border and a white background.

TIP: You can make use of the syntax helper to see the variables declared in your user-defined structures.

Passing parameters to subprograms

Parameters can be passed to a subprogram by reference (`ByRef`) or by value (`ByVal`).

Pass by reference -- if you pass a parameter by reference, any changes to the parameter will propagate back to the caller. Pass by reference is the default.

Pass by value -- if you pass a parameter by value, no changes are allowed to propagate back to the caller. With a few exceptions, if an argument is passed by value, a copy is made of the argument, and the called subprogram operates on the copy. Changes made to the copy have no effect on the caller.

One exception is for string parameters passed by value. For efficiency reasons, a copy of the string is not made. Instead, the string is write-protected in the called subprogram, which means you can neither assign to it nor pass it by reference to another subprogram. You are allowed to pass it by value to another subprogram, however.

The other exception is for types `Unsigned Integer` and `Unsigned Long`, which are treated similarly – these parameters are write-protected in called subprograms.

Actual vs. formal parameters -- the type and number of the actual parameters must match that of the "formal" parameters (the formal parameters appear in the subprogram declaration). If there is a type mismatch, the compiler will declare an error. It will not do implicit type conversions.

Example:

```
Sub Collate(ByVal I As Integer, ByRef X As Single, ByVal B As Byte)
```

Restrictions on passing mechanisms:

- Scalar variables and array elements can be passed by value or by reference.
- Since array are global they are cannot be passed into a function or subroutine.
- Numeric expressions and numeric literals can be passed by value but not by reference. The same applies to boolean expressions and boolean literals.

Data types

Type Storage Range

Boolean	8 bits	True or False
Byte	8 bits	0 to 255
Integer	16 bits	-32 768 to 32 767

Declarations

All variables must be declared before they're used. Implicit declarations are not allowed. For variables declared in module-level code:

```
[Public | Private | Dim] variable As type
```

Public variables are global and visible throughout the entire program. Private variables are visible only in the module in which they appear. The default is private – that is, if Dim is used for a module level variable, the variable is private.

For variables declared inside a subprogram:

```
Dim variable As type
```

Variables declared inside a subprogram are visible only inside the subprogram.

Example:

```
Public Distance As Integer ' Module-level variable, global
Private Temperature As Single ' Module-level variable, local to 'module
Sub ReadPin()
    Dim PinNumber As Byte ' Variable is local to this 'subprogram
End Sub
```

Constants

For constants declared in module-level code:

```
[Public | Private] Const constant_name As type = literal
```

The default is private.

Numeric literals

Decimal integer examples:

1

-1

10

255

Note that integer numeric literals are not allowed to have a decimal point.

Labels

Labels serve as targets for the 'goto' statements. Mark the desired statement with label and colon like this:

```
label_identifier : statement
```

Arrays

An array represents an indexed collection of elements of the same type (called the base type).

- Max dimension is 3
- Array must be Public
- Only Integer Arrays are allowed

NOTE: While the maximum dimension for declaring an array is 3, please take note that the maximum length allowed for the 2nd and 3rd dimension e.g. (*array_length_2*, *array_length_3*) is 255. There is no limit for the 1st dimension of the array, but onboard SRAM memory pertaining to the respective chip you are using should be taken to mind. The array length should not exceed the SRAM Memory available.

	FI28	FI40	FI80
SRAM	2048	2048	6144

Above: Table showing SRAM Memory available onboard the FI Series.

Declaration :

```
Dim ArrayName(array_length) As Integer '1 dimension
Dim ArrayName(array_length_1, array_length_2) As Integer '2 dimensions
Dim ArrayName(array_length_1, array_length_2, array_length_3) As
Integer '3 dimensions
```

Strings

A string represents a sequence of characters equivalent to an array of char.

Strings are always declared 32 characters long

Declaration :

```
Dim StringName As String
```

String library

CSTR
MID
GETCHAR
Plus Operator

CSTR

Convert data type to String

Example:

```
Dim i As Integer
i = 13
Debug.Print "The Number is: ";CSTR(i)
```

Output :

```
The Number is 13
```

MID

Returns a string containing a specified number of characters from a string.

MID (String, Start, Length)

Example:

```
Dim MIDtest As String = "First Middle Last"
Dim a As String = Mid(MIDtest, 1, 5)
Dim b As String = Mid(MIDtest, 7, 6)
Dim c As String = Mid(MIDtest, 7)
```

```
Debug.Print "a = ";a
Debug.Print "b = ";b
Debug.Print "c = ";c
```

Output :

```
a = First
b = Middle
c = Middle Last
```

GETCHAR

Returns a Char value representing the character from the specified index in the supplied string.

GETCHAR (String, Index)

Example:

```
Dim aStr As String = "ABCDE"
Dim bChr As Char
bChr = GetChar(aStr, 4)
```

```
Debug.Print CSTR(bChr)
```

Output :

```
D
```

Plus Operator

A plus operator can be used to append strings.

Example:

```
Dim a As String = "Hello,"  
Dim b As String = "World!"  
Dim c As String = ""
```

```
c = a + b  
Debug.Print c
```

Output:

```
Hello, World!
```

Operators

Operators describe and perform an operation between two or more values.

Arithmetic Operators

Arithmetic operators are used to perform mathematical computations. They have numerical operands and return numerical results.

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), modulus (Mod), negation (-) and exponentiation (^).

Order of precedence for arithmetic operators follow the rules of precedence from basic math, which is, left to right.

Operator	Operation
^	Exponentiation
-	Negation
*, /	Multiplication and division
\	Integer Division
Mod	Modulus
+, -	Addition and subtraction

Table above lists operators and operations from highest precedence to the lowest.

Relational Operators

Use relational operators to test equality or inequality of expressions. All relational operators return TRUE or FALSE.

Operator	Operation
=	equal
<>	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal

All relational operators associate from left to right.

Bitwise Operators

Use the bitwise operators to modify the individual bits of numerical operands.

Operator	Operation
and	bitwise AND; compares pairs of bits and generates a 1 result if both bits are 1, otherwise it returns 0
or	bitwise (inclusive) OR; compares pairs of bits and generates a 1 result if either or both bits are 1, otherwise it returns 0
xor	bitwise exclusive OR (XOR); compares pairs of bits and generates a 1 result if the bits are complementary, otherwise it returns 0
not	bitwise complement (unary); inverts each bit
<<	bitwise shift left; moves the bits to the left, it discards the far left bit and assigns 0 to the right most bit.
>>	bitwise shift right; moves the bits to the right, discards the far right bit and if unsigned assigns 0 to the left most bit, otherwise sign extends

Boolean Operators

Operator	Operation
and	logical AND
or	logical OR
xor	logical exclusive OR (XOR)
not	logical negation

Boolean operators associate from left to right. The negation operator '*not*' associates from right to left.

Overall Operator Precedence

(Highest)	[1] Not
	[2] * \ Mod And
	[3] + - Or Xor &
(Lowest)	[4] = > < <> <= >=

Expressions

An expression is a sequence of operators, operands, and punctuators that returns a value.

```
ABS(variable)
SETBIT(variable, position)
CLRBIT(variable, position)
GETBIT(variable, position)
```

```
HIGH(variable)
LOW(variable)
```

Statements

Statement format

A statement begins at the beginning of a line of text and terminates at the end of a line of text. In FIDE you can have at most 1 statement per line.

IF Statement

Conditionally executes a group of statements, depending on the value of an expression.

```
If <condition> Then
    [ statements ]
[ ElseIf <condition> Then
    [ statements ] ]
[ Else
    [ statements ] ]
End If
-or-
If condition Then [ statements ]
```

Select Case Statement

Runs one of several groups of statements, depending on the value of an expression.

```
Select Case <test expression>
Case <expression>
    [ statements ]
[ Case <expression>
    [ statements ] ]
[ Case Else
    [ statements ] ]
End Select
```

Iteration Statements (Loops)

For Statement

Repeats a group of statements a specified number of times.

```
For counter = start To end [ Step value ]  
    [ statements ]  
    [ Exit For ]  
    [ statements ]  
Next
```

Example:

```
Dim i As Integer  
  
For i = 0 To 3  
    Debug.Print CSTR(i)  
Next
```

Output :

```
0  
1  
2  
3
```

Do Statement

Repeats a block of statements while a Boolean condition is True or until the condition becomes True.

```
Do { While | Until } <condition>  
    [ statements ]  
    [ Exit Do ]  
    [ statements ]  
Loop  
-or-  
Do  
    [ statements ]  
    [ Exit Do ]  
    [ statements ]  
Loop { While | Until } <condition>
```

Example:

```
Dim i As Integer  
i = 0  
  
Do  
    Debug.Print Cstr(i)  
    i = i + 1  
Loop Until i = 3
```

Output :

0
1
2
3

¹ Sound track files are subject to change.

APPENDIX A

Songs

File/s	BlockStart	BlockEnd	Block Length	Last Pos. in Block
01 KT Tunstall - Other Side Of The World.wav	16	13434	13419	134
Avril Lavigne - Girlfriend.wav	13435	26917	13483	222

Numbers

	Words	File/s	BlockStart	BlockEnd	Block Length	Last Pos. in Block
0	zero	N0000.wav	16	54	39	0
1	one	N0001.wav	55	80	26	0
2	two	N0002.wav	81	107	27	0
3	three	N0003.wav	108	137	30	0
4	four	N0004.wav	138	162	25	0
5	five	N0005.wav	163	195	33	0
6	six	N0006.wav	196	230	35	0
7	seven	N0007.wav	231	260	30	0
8	eight	N0008.wav	261	289	29	0
9	nine	N0009.wav	290	326	37	0
10	ten	N0010.wav	327	352	26	0
11	eleven	N0011.wav	353	389	37	0
12	twelve	N0012.wav	390	427	38	0
13	thirteen	N0013.wav	428	470	43	0
14	fourteen	N0014.wav	471	513	43	0
15	fifteen	N0015.wav	514	560	47	0
16	sixteen	N0016.wav	561	612	52	0
17	seventeen	N0017.wav	613	667	55	0
18	eighteen	N0018.wav	668	708	41	0
19	nineteen	N0019.wav	709	751	43	0
20	twenty	N0020.wav	752	782	31	0
21	thirty	N0021.wav	783	825	43	0
22	fourty	N0022.wav	826	863	38	0
23	fifty	N0023.wav	864	902	39	0
24	sixty	N0024.wav	903	941	39	0
25	seventy	N0025.wav	942	983	42	0
26	eighty	N0026.wav	984	1024	41	0
27	ninety	N0027.wav	1025	1063	39	0
28	hundred	N0028.wav	1064	1094	31	0
29	thousand	N0029.wav	1095	1132	38	0
30	million	N0030.wav	1133	1163	31	0
31	billion	N0031.wav	1164	1190	27	0
32	plus	N0032.wav	1191	1215	25	0

33	minus	N0033.wav	1216	1252	37	0
34	negative	N0034.wav	1253	1290	38	0
35	product	N0035.wav	1291	1326	36	0
36	dollars	N0036.wav	1327	1367	41	0
37	cents	N0037.wav	1368	1400	33	0
38	a.m	N0038.wav	1401	1430	30	0
39	p.m	N0039.wav	1431	1465	35	0
40	o'clock	N0040.wav	1466	1509	44	0
41	hour	N0041.wav	1510	1533	24	0
42	minute	N0042.wav	1534	1558	25	0
43	second	N0043.wav	1559	1593	35	0
44	first	N0044.wav	1594	1631	38	0
45	third	N0045.wav	1632	1659	28	0
46	point	N0046.wav	1660	1687	28	0
47	half	N0047.wav	1688	1715	28	0
48	quarter	N0048.wav	1716	1742	27	0
49	over	N0049.wav	1743	1765	23	0

Alphabets

	Alphabets	File/s	BlockStart	BlockEnd	Block Length	Last Pos. in Block
1	a	A0001.wav	16	49	34	0
2	b	A0002.wav	50	64	15	0
3	c	A0003.wav	65	95	31	0
4	d	A0004.wav	96	134	39	0
5	e	A0005.wav	135	157	23	0
6	f	A0006.wav	158	190	33	0
7	g	A0007.wav	191	226	36	0
8	h	A0008.wav	227	263	37	0
9	i	A0009.wav	264	283	20	0
10	j	A0010.wav	284	317	34	0
11	k	A0011.wav	318	341	24	0
12	l	A0012.wav	342	367	26	0
13	m	A0013.wav	368	394	27	0
14	n	A0014.wav	395	409	15	0
15	o	A0015.wav	410	438	29	0
16	p	A0016.wav	439	467	29	0
17	q	A0017.wav	468	499	32	0
18	r	A0018.wav	500	527	28	0
19	s	A0019.wav	528	559	32	0
20	t	A0020.wav	560	585	26	0
21	u	A0021.wav	586	619	34	0
22	v	A0022.wav	620	644	25	0

23	w	A0023.wav	645	684	40	0
24	x	A0024.wav	685	717	33	0
25	y	A0025.wav	718	755	38	0
26	z	A0026.wav	756	787	32	0

Hundred Words

	VOCAB	File/s	BlockStart	BlockEnd	Block Length	Last Pos. in Block
0	the	W0000.wav	16	34	19	0
1	of	W0001.wav	35	63	29	0
2	and	W0002.wav	64	84	21	0
3	a	W0003.wav	85	104	20	0
4	to	W0004.wav	105	130	26	0
5	in	W0005.wav	131	151	21	0
6	is	W0006.wav	152	182	31	0
7	you	W0007.wav	183	216	34	0
8	that	W0008.wav	217	240	24	0
9	it	W0009.wav	241	257	17	0
10	he	W0010.wav	258	280	23	0
11	was	W0011.wav	281	303	23	0
12	for	W0012.wav	304	328	25	0
13	on	W0013.wav	329	350	22	0
14	are	W0014.wav	351	377	27	0
15	as	W0015.wav	378	409	32	0
16	with	W0016.wav	410	431	22	0
17	his	W0017.wav	432	456	25	0
18	they	W0018.wav	457	480	24	0
19	I	W0019.wav	481	501	21	0
20	at	W0020.wav	502	525	24	0
21	be	W0021.wav	526	544	19	0
22	this	W0022.wav	545	572	28	0
23	have	W0023.wav	573	602	30	0
24	from	W0024.wav	603	628	26	0
25	or	W0025.wav	629	655	27	0
26	one	W0026.wav	656	681	26	0
27	had	W0027.wav	682	696	15	0
28	by	W0028.wav	697	717	21	0
29	word	W0029.wav	718	739	22	0
30	but	W0030.wav	740	757	18	0
31	not	W0031.wav	758	793	36	0
32	what	W0032.wav	794	817	24	0
33	all	W0033.wav	818	842	25	0
34	were	W0034.wav	843	858	16	0
35	we	W0035.wav	859	875	17	0

36	when	W0036.wav	876	902	27	0
37	your	W0037.wav	903	932	30	0
38	can	W0038.wav	933	957	25	0
39	said	W0039.wav	958	984	27	0
40	there	W0040.wav	985	1004	20	0
41	use	W0041.wav	1005	1033	29	0
42	an	W0042.wav	1034	1059	26	0
43	each	W0043.wav	1060	1094	35	0
44	which	W0044.wav	1095	1111	17	0
45	she	W0045.wav	1112	1137	26	0
46	do	W0046.wav	1138	1160	23	0
47	how	W0047.wav	1161	1181	21	0
48	their	W0048.wav	1182	1198	17	0
49	if	W0049.wav	1199	1216	18	0
50	will	W0050.wav	1217	1245	29	0
51	up	W0051.wav	1246	1266	21	0
52	other	W0052.wav	1267	1288	22	0
53	about	W0053.wav	1289	1316	28	0
54	out	W0054.wav	1317	1329	13	0
55	many	W0055.wav	1330	1354	25	0
56	then	W0056.wav	1355	1379	25	0
57	them	W0057.wav	1380	1399	20	0
58	these	W0058.wav	1400	1443	44	0
59	so	W0059.wav	1444	1480	37	0
60	some	W0060.wav	1481	1520	40	0
61	her	W0061.wav	1521	1537	17	0
62	would	W0062.wav	1538	1551	14	0
63	make	W0063.wav	1552	1573	22	0
64	like	W0064.wav	1574	1595	22	0
65	him	W0065.wav	1596	1614	19	0
66	into	W0066.wav	1615	1656	42	0
67	time	W0067.wav	1657	1692	36	0
68	has	W0068.wav	1693	1722	30	0
69	look	W0069.wav	1723	1757	35	0
70	two	W0070.wav	1758	1785	28	0
71	more	W0071.wav	1786	1808	23	0
72	write	W0072.wav	1809	1843	35	0
73	go	W0073.wav	1844	1868	25	0
74	see	W0074.wav	1869	1900	32	0
75	number	W0075.wav	1901	1939	39	0
76	no	W0076.wav	1940	1969	30	0
77	way	W0077.wav	1970	2007	38	0
78	could	W0078.wav	2008	2021	14	0

79	people	W0079.wav	2022	2048	27	0
80	my	W0080.wav	2049	2069	21	0
81	than	W0081.wav	2070	2093	24	0
82	first	W0082.wav	2094	2132	39	0
83	water	W0083.wav	2133	2165	33	0
84	been	W0084.wav	2166	2189	24	0
85	call	W0085.wav	2190	2223	34	0
86	who	W0086.wav	2224	2245	22	0
87	oil	W0087.wav	2246	2283	38	0
89	now	W0089.wav	2284	2307	24	0
90	find	W0090.wav	2308	2335	28	0
91	long	W0091.wav	2336	2360	25	0
92	down	W0092.wav	2361	2395	35	0
93	day	W0093.wav	2396	2414	19	0
94	did	W0094.wav	2415	2436	22	0
95	get	W0095.wav	2437	2454	18	0
96	come	W0096.wav	2455	2487	33	0
97	made	W0097.wav	2488	2510	23	0
98	may	W0098.wav	2511	2546	36	0
99	part	W0099.wav	2547	2569	23	0