NXT File System

- Just like we're able to store multiple programs and sound files to the NXT, we can store text files that contain information we specify.
  - Text files can contain any type of data (string, int, char, float, etc)
  - This can be useful for sharing data across multiple programs or for data logging purposes.

NXT File System

- How it works:
  - ROBOTC opens a text file for reading or writing.
    - OpenWrite or OpenRead
  - A unique “handle” is generate from this command to allow us to know the address of the file we’re working with.
    - A special variable type called “TFileHandle” is used to store the handle
  - Every command referencing file I/O begins with requesting the file’s “handle” – this is to ensure that if you have multiple files open that we are working with the correct one.
  - Once we’re done working with the file, we call the close command to finish working with the file and make it available for access elsewhere.

Opening Files

- OpenWrite(hFileHandle, nIoResult, sFileName, nFileSize);
  - Opens a file for writing to the NXT's file system.
  - **hFileHandle** - Returns a handle for future access to the file
    - Declare this variable as type TFileHandle
  - **nIoResult** – Returns a result code to determine if the file operation was successful.
    - Declare this variable as type TFileIOResult
  - **sFileName** – Input parameter of the name of the file we want to write to. If this file doesn't exist, the openWrite command will create it.
    - Filename is passed as a string (example: string myFileName "myFile.txt")
  - **nFileSize** – Input parameter to determine the maximum size of the file.
    - Declare this as an integer to specify the size of the file.

Opening Files

- OpenRead(hFileHandle, nIoResult, sFileName, nFileSize);
  - Opens a file for reading from the NXT's file system.
  - **hFileHandle** - Returns a handle for future access to the file
    - Declare this variable as type TFileHandle
  - **nIoResult** – Returns a result code to determine if the file operation was successful.
    - Declare this variable as type TFileIOResult
  - **sFileName** – Input parameter of the name of the file we want to open.
    - Filename is passed as a string (example: string myFileName "myFile.txt")
  - **nFileSize** – Return parameter with the size of the contents of the text file. Use for knowing the length of the file when reading back data.
    - This does not mean the maximum size of the file, but rather the actual size of the contents.
    - Declare this as an integer to specify the size of the file.

Closing Files

- Close(hFileHandle, nIoResult);
  - nFileHandle – the Handle that was created by the "open" command. Used to make sure we close the right file.
  - nIoResult - Returns a result code to determine if the file operation was successful.

Example Code

```
task main()
{
  TFileHandle myFileHandle;
  TFileIOResult IOResult;
  string myFileName = "myFile.txt";
  int sizeOfFile = 200;

  OpenWrite(myFileHandle, IOResult, myFileName, sizeOfFile);

  Close(myFileHandle, IOResult);

  OpenRead(myFileHandle, IOResult, myFileName, sizeOfFile);

  Close(myFileHandle, IOResult);
}
```

Write to a File

- Now that we have created our text file, we can write data to it.
  - The NXT has 6 different “write” commands to write data to a file.
    - **WriteByte(hFileHandle, nIoResult, nParm);**
    - **WriteFloat(hFileHandle, nIoResult, fParm);**
    - **WriteLong(hFileHandle, nIoResult, nParm);**
    - **WriteShort(hFileHandle, nIoResult, nParm);**
    - **WriteString(hFileHandle, nIoResult, sParm);**
    - **WriteText(hFileHandle, nIoResult, sParm);**

Writing Values

- WriteFloat, WriteLong, WriteShort
- Parameters - (hFileHandle, nIoResult, nParm);
  - hFileHandle – Handle to the file we want to write to.
  - nIoResult – Returned result variable to know if the write was successful
  - nParm – the value that we want to write (example: 52.43, 1234567, 459)
- These values are written in a machine language to the text file.
- Values written using these commands will not be human readable if the text file is opened in a “notepad” application.

Writing Text

- WriteByte, WriteString, WriteText
- Parameters - (hFileHandle, nIoResult, nParm);
  - hFileHandle – Handle to the file we want to write to.
  - nIoResult – Returned result variable to know if the write was successful
  - nParm – the value that we want to write (example: ‘A’, “My String!”, “My Text!”)
- Values written to a text file as text and will be shown as normal text in a notepad like application.

Numbers vs. Text

- Why choose one over the other?
  - Values written as **numbers** are not human readable, but are easier to import into ROBOTC to be used as numbers in your program.
    - Example – Storing encoder counts and then recalling the number of counts to be used with nMotorEncoderTarget
  - Values written as **text** are human readable, but are not able to be easily used by your ROBOTC program
    - Example – The string “3.1415” is a human readable number
    - However, it is not equal to the number 3.1415

Numbers vs. Text

- Are all numbers the same?
  - Is 42 the same as “42”?
  - Characters and strings are different from numbers.
  - A string is formally a sequence of characters
    - Typically made of an array of characters
    - Each character has a value of its own, based off of the ASCII standard

ASCII Character:	3	.	1	4	1	5
Integer Value:	51	46	49	52	49	53

ASCII Table

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

String Conversion

- Because of this, we need a special command to convert numbers from a string to an integer.
- atoi(inputString)
  - Converts a string of numbers into an integer number
- Usage:
  - `String myString = “523”`
  - `int myNumber = 0;`
  - `myNumber = atoi(myString)`
- myNumber now equals 523 (as a number)
- atof – Converts string to a float
- atol – Converts string to a long integer.

Dec	Hx	Oct	Html	Chr
32	20	040		Space
33	21	041	!	!
34	22	042	"	"
35	23	043	#	#
36	24	044	$	$
37	25	045	%	%
38	26	046	&	&
39	27	047	'	'
40	28	050	(	(
41	29	051	)	)
42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9

Numbers vs. Text

- Because of the conversion tools we have, it's generally recommended to write values as "text" instead of "numbers" when using file reads and writes.
- This provides more flexibility and also gives you the power to debug your file IO command because you will be able to open the text file and see if the correct values are being printed.
- Keep in mind that strings are limited to 19 characters in ROBOTC

Writing to a File

- Sample Code:

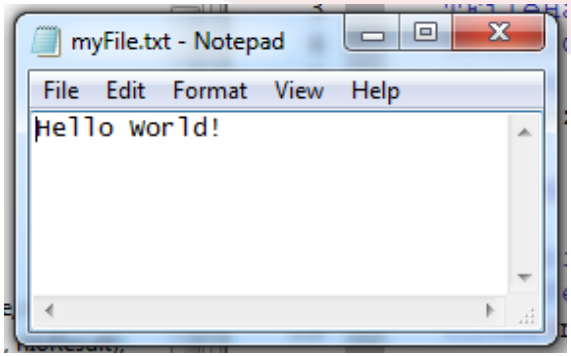
```
task main()
{
  TFileHandle myFileHandle;
  TFileIOResult IOResult;
  string myFileName = "myFile.txt";
  int sizeOfFile = 20;

  string Outgoing = "Hello World!";

  OpenWrite(myFileHandle, IOResult, myFileName, sizeOfFile);
  WriteText(myFileHandle, IOResult, Outgoing);
  Close(myFileHandle, IOResult);
}
```

Getting our File

- We can use the NXT File Management Utility to "Upload" our file to the PC.
  - Robot -> NXT Brick -> File Management
- Find your text file in the list and save it to your computer.
- Navigate to the file on your computer and open it.



Reading from a File

- ROBOTC has 4 commands for reading from a file:
  - **ReadByte(hFileHandle, nIoResult, nParm);**
  - **ReadFloat(hFileHandle, nIoResult, fParm);**
  - **ReadLong(hFileHandle, nIoResult, nParm);**
  - **ReadShort(hFileHandle, nIoResult, nParm);**
- But wait…
- Where is the “Read String” command?
  - Remember that strings are only character arrays in nice, easy to use packages (data type).

Reading from a File

- Remember our OpenRead command:
  - OpenRead(hFileHandle, nIoResult, sFileName, nFileSize);
  - The nFileSize will tell us how many characters there are in the file
  - We can use this data to create a For Loop to parse through each character.
  - As we receive each character, we can store it to a string.

Reading from a File

```
task main()
{
  TFileHandle myFileHandle;
  TFileIOResult IOResult;
  string myFileName = "myFile.txt";
  int sizeOfFile = 0;

  string Incoming = "";
  char IncomingChar;

  OpenRead(myFileHandle, IOResult, myFileName, sizeOfFile);

  for(int i = 0; i < sizeOfFile; i++)
  {
    ReadByte(myFileHandle, IOResult, IncomingChar);
    Incoming += IncomingChar;
  }

  Close(myFileHandle, IOResult);
}
```

Global Variables

Index	Variable	Value
0	myFileHandle	1
1	IOResult	ioRsltSuccess
2H	myFileName	"myFile.txt"
12	sizeOfFile	12
13H	Incoming	"Hello World!"
23H	IncomingChar	33 ('!')
24	i	12

Writing and Reading Values

- We can use commands like StringFormat in order to package up variable values to be saved as text.
  - StringFormat(outgoing, “Var1: %d, myVar);
- Keep in mind when we read values back as strings, we’ll have use our “ato*” commands to process the number.
  - atoi(“54”) = returns a value of 54 as a number

Writing Multiple Lines

- We're able to write to multiple lines by adding in return characters to what we write.
  - Example: Write 5 lines that read "Writing Line #1", "Writing Line #2", etc…
- To do this effectively, we'll need to use two things:
  - Our ASCII Table
  - A For Loop
- Don't forget to increase the size of your file, as writing more lines means we need more space!

Writing Multiple Lines

- If we search about how a computer processes a “return” or “enter key”, it’s not an invisible command that magically happens
- Every characters and non-character (spaces, tabs, returns, etc) has an ASCII value.
  - But nothing’s ever easy, right?

Writing Multiple Lines

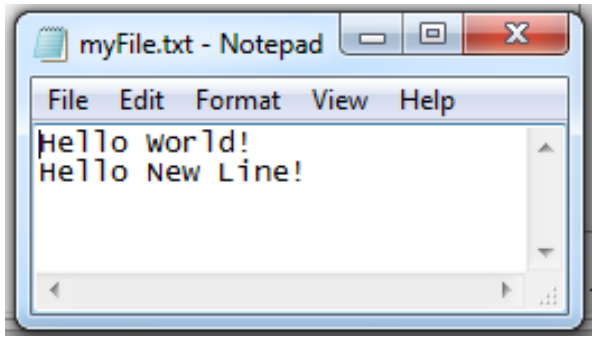
- Systems based on ASCII or a compatible character set use either…
  - Line Feed (LF - 0x0A, 10 in decimal)
  - Carriage return (CR - 0x0D, 13 in decimal)
  - CR followed by LF (CR+LF, '\r\n', 0x0D and 0x0A).
- These characters are based on printer commands:
  - The line feed indicated that one line of paper should feed out of the printer
  - Carriage return indicated that the printer carriage should return to the beginning of the current line.
- CR+LF: Microsoft Windows
- LF: Unix and Unix-like systems (Linux, Mac OS X)

Writing Multiple Lines

- Using the “WriteByte” command, we can write the commands LF (10) and CR (13) to our text file to make a new line.

```
string Outgoing = "Hello World!";

OpenWrite(myFileHandle, IOResult, myFileName, sizeOfFile);
WriteText(myFileHandle, IOResult, Outgoing);
WriteByte(myFileHandle, IOResult, 13);
WriteByte(myFileHandle, IOResult, 10);
WriteText(myFileHandle, IOResult, "Hello New Line!");
Close(myFileHandle, IOResult);
```



Reading Multiple Lines

- The same idea applies…
  - Now we have to build logic in to ask when reading in our file if we see a LF and CR command.
  - When we do, we'll have to start writing to new string.
- Quick Quiz: When we need multiples of the same variables and data… what do we use?

Reading Multiple Lines

```
int nLineCounter = 0;
string incomingArray[5];
OpenRead(hFileHandle, nIoResult, sFileName, nFileSize);

for (int index = 0; index < nFileSize; ++index)
{
  ReadByte(hFileHandle, nIoResult, incomingChar);

  if(incomingChar == 13 || incomingChar == 10)
  {
    if(incomingChar == 10)
      nLineCounter++;
  }
  else
  {
    incomingString[nLineCounter] += incomingChar;
  }
}
Close(hFileHandle, nIoResult);
```

Reading Multiple Lines

- First we ask “do I see a LF or a CR?”
  - If not, we write the current character to the current string in our array.
- Next, we ask if the incoming character is CR we because we know that CRs come after LFs.
  - When I see the “CR”, increment our Line Counter.
  - When we see a “LF”, we will fail the “CR” check - this is ideal. We don’t want to print the LF to our string.

```
if(incomingChar == 13 || incomingChar == 10)
{
  if(incomingChar == 10)
    nLineCounter++;
}
else
{
  incomingString[nLineCounter] += incomingChar;
}
```

Parsing Data

- Note: String are displayed with double quotes, normal values/numbers are not.

```
task main()
{
  string sFloat = "219.323";
  float returnFloat;

  string nLong = "432525432";
  long returnLong;

  string nInt = "12349";
  int returnInt;

  returnInt = atoi(nInt);
  returnLong = atol(nLong);
  returnFloat = atof(sFloat);
}
```

Global Variables

Index	Variable	Value
204H	sFloat	"219.323"
214	returnFloat	219.323
216H	nLong	"432525432"
226	returnLong	432525432
228H	nInt	"12349"
238	returnInt	12349

Errors | Global Variables

Data Logging

- To create a text file that works like a Data Log, you can create a CSV file
  - CSV – Comma Separated Values
- CSV files have the following format:

Raw CSV File

```
Time Elapsed,EncoderB,EncoderC
0,0,0
100,232,233
200,421,454
300,643,632
```

Opened In Excel

	A	B	C
1	Time Elapsed	EncoderB	EncoderC
2	0	0	0
3	100	232	233
4	200	421	454
5	300	643	632

CSV Formatting

- To save a string in CSV format:
  - **StringFormat(CurrentString, “%d,%d,%d,”, Var1, Var2, Var3)**
- For the first line with the subject titles, you may want to write this to the text file in 3 “writes”
  - “Heading 1” – Write #1
  - “,Heading 2” – Write #2
  - “,Heading 3” – Write #3
  - Then don’t forget your LF and CR bytes!

Other File Tools

- **Delete(sFileName, nIoResult);**
  - Delete the file from the NXT
  - This is useful to add to your program before you write to a file to ensure that you're not mixing old data with new.
- **Rename(sFileName, nIoResult, sOriginalFileName);**
  - Rename sOriginalFileName to sFileName
  - Useful for making backups of your files before you want to create a new one.