The TRAKR Codebook

A Crash Course in C Programming for the Spy Video TRAKR™

CONTENTS

SO YOU WANNA WRITE AN APP FOR THE TRAKR

Well, you've come to the right place. The TRAKR Codebook wont turn you into a programmer overnight, but you will be able to build some basic apps for your TRAKR. When you're ready to learn more about the world of C programming, go online and take advantage of free resources like Wikipedia or the spytrakr.com forums. Remember — the capabilities of the TRAKR are as vast as your imagination!

TRAKR TIP: If this information is too complex, try the App BUILDR at www.spytrakr.com.

Read on for a crash course in TRAKR coding basics: TRAKR-specific software functions, some code examples, and even a few implementation guidelines. Pretty soon that genius app idea will be out of your brain and on your TRAKR.

```
if(obvious_meaning()==true){goToSection(examples||reference);}
```

Uh, What?!?

No, we're not writing in some robot language from the future — that's a snippet of C, the programming language in which the TRAKR's software is written. Looks cool? Then keep reading.

Jargon

Jargon is made-up words or regular words which have a different meaning to experts than to the rest of us. Experts use jargon as shorthand for complex ideas. Learning jargon is a big part of becoming an expert on anything. For example, if you didn't know anything about baseball and heard a radio announcer talking about "knuckle balls" and "bunts", you wouldn't have a clue as to what he meant unless you knew some baseball jargon. As you read through these pages and try some of the examples, you'll start using some jargon, too. Jargon saves typing and makes things clear by associating complex processes with short words.

Syntax

Syntax refers to the rules for making sentences in a language. Just like French or Spanish, programming languages use syntax to make sure terms are ordered in a way that can be understood.

There are many special characters used to make syntactical distinctions in programming. For instance, a semicolon ";" is interpreted in C to be the end of a statement. You can separate two lines by a return, but they may be read by the machine as being one statement without a semicolon. If you omit a semicolon, you will be looking for a "syntax error" when you read through your code checking that your statements all have ends.

Most errors in your programs will be syntax errors. They are the easiest to make and the hardest to find. As you master the syntax, you'll find that coding gets a lot more fun!

Functions

In the C language, a function is a piece of code which runs when it is requested by another piece of code. Often times, the code which calls a function is expecting to get some data back from the function. The type of data expected should match the type of data the function will generate.

For instance, if you had created a function called "TheCurrentTime()", you would expect to get some data back which represents the time of day.

Sometimes a function does not return anything to the code which called it. It just runs its own code and then returns to the code which called it. You might expect a function named "Sleep()" to pause for a while and then return to the code without producing any data. The function "Sleep()" is, in fact, a function which the TRAKR understands.

Parameters

Many functions expect parameters to work with. A parameter is anything passed on to the function from the code which called it. The parentheses after a function's name are where the parameters are specified and multiple parameters can be separated by a comma within the parentheses.

In the example of the Sleep() function, there is the essential parameter of "...for how long?" which needs to be passed when you call it. You could write "Sleep(for 3 seconds)" but that is too wordy and not what the function expects. All you need to send is a number which the function can use. You could write "Sleep(3)" but this function expects the parameter to be in milliseconds. To get 3 seconds of sleeping, you should write "Sleep(3000)". This parameter is an integer (a whole number) which corresponds to the number of 1/1000ths of a second to pause your program.

Data Types

Data is sent to a function when it is called and it needs to meet the expectations of the function. The main characteristic to regard is the type of data. "3 seconds", ,"3" "3000"ms, "3.0" or "00000011" could all be different ways of expressing the same time to Sleep(), but only the "3000" will get the expected result. Checking that you are sending the right kind of data in your function calls will be your first tactic in debugging.

TRAKR TIP: Check the TRAKR Function Reference to find the correct types for parameters.

Reference Manuals

No one could make any progress if they tried to remember every detail of every function they had used and made, so it's helpful to store the vital statistics in a list to refer back to as needed. When you look up Sleep() in the TRAKR Function Reference, you might find something like the following:

Sleep()

Description:
Pauses the program for the amount of time (in milliseconds) specified as parameter.
(There are 1000 milliseconds in a second.)

Syntax:
Sleep (uint32 us)

Parameters:
uint32 us: the number of milliseconds to pause (unsigned long, 0 through 4,294,967,295)

Returns:
nothing
Example:

```
#include “svt.h”

void Start()
{
  // nothing to do here
}

bool Run()
{
  Sleep(1000);
  // do something
  Sleep(1000);
  // do something else
  return true;
}

void End()
{
  // do nothing
}
```

You can see from this example reference page that Sleep() expects to get a PARAMETER, which in this case is the number of milliseconds to wait before returning to the program.

TRAKR TIP: Write TRAKR code more easily! Copy sample code from the TRAKR Function Reference into your own code.

Defining Functions

The TRAKR only calls 3 functions when it powers on. These functions are; `Start(), Run()`and `End()`. The TRAKR will call each function in this order. Each TRAKR program you create must have these functions defined by your code. How you define them determines what your program will do. Here is the simplest TRAKR program that can be written, which does nothing but define the 3 mandatory functions and load the svt.h file (which is the file that explains all the default TRAKR functions):

```
#include “svt.h”
void Start(){}
bool Run(){return true;}
void End(){}
```

The following code functions the same as the above but is much easier to read. Sometimes coders add extra returns and spaces to reveal the logical structure as well as the order in which various operations occur.

It is helpful to understand which sections are contained within each other just by how indented the text is. It might not look much better now but, later on, your ability to debug code will be largely dependent on understanding the order of operations and subroutines.

```
#include “svt.h”

void Start()
{
}

bool Run()
{
  return true;
}

void End()
{
}
```

Comments

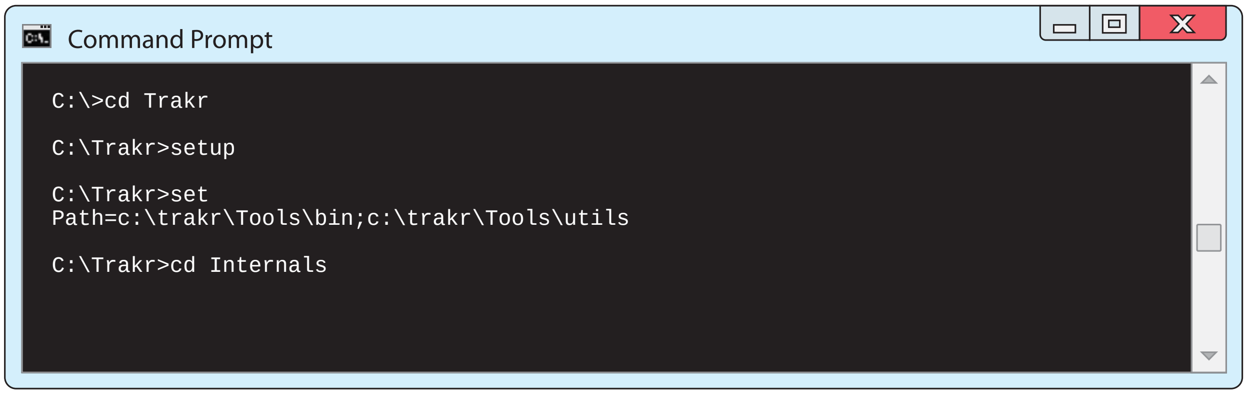
Comments are a tool that gives your code clarity. Comments do not get used by the machine but are there for you and others to help understand what your code is doing. Adding comments to your code will give you a much better understanding of how to improve and adjust it. Almost all code contains comments to help make the code legible or to keep sections of code not in use but available for reference. Write your comments clearly and revise them as you work so they remain accurate.

Comments in C start with two forward slashes and end with a return. They can be on their own line or after some functional code. Most of the following examples will use comments to explain their workings. Below is the same code as above but with some comments:

```
// we must include the svt.h file so that the compiler knows
// how to handle all the special functions for the
// Spy Video TRAKR.
#include “svt.h”
// define the Start function:
void Start()
{
  // this is where things that need to happen at
  // start-up would go.
}
// define Run function:
bool Run()
{
  // This space would contain the main body of our code.
  return true;
}
// lastly, the End function:
void End()
{
  // and here is where we can do any final tasks before
  // closing? starting again? resetting?
}
```

Compiler

A compiler is a tool to convert your written code into a real, machine readable executable program. In order for it to make the correct files and put them in the correct folders, it must be told where everything is. The program to do this is "setup". It is located in the "Trakr" folder which should be unzipped onto your C:\ drive.



```
C:\>cd Trakr

C:\Trakr>setup

C:\Trakr>set
Path=c:\trakr\Tools\bin;c:\trakr\Tools\utils

C:\Trakr>cd Internals
```

If the setup is successful, you should set the tool paths to define something like the above result.

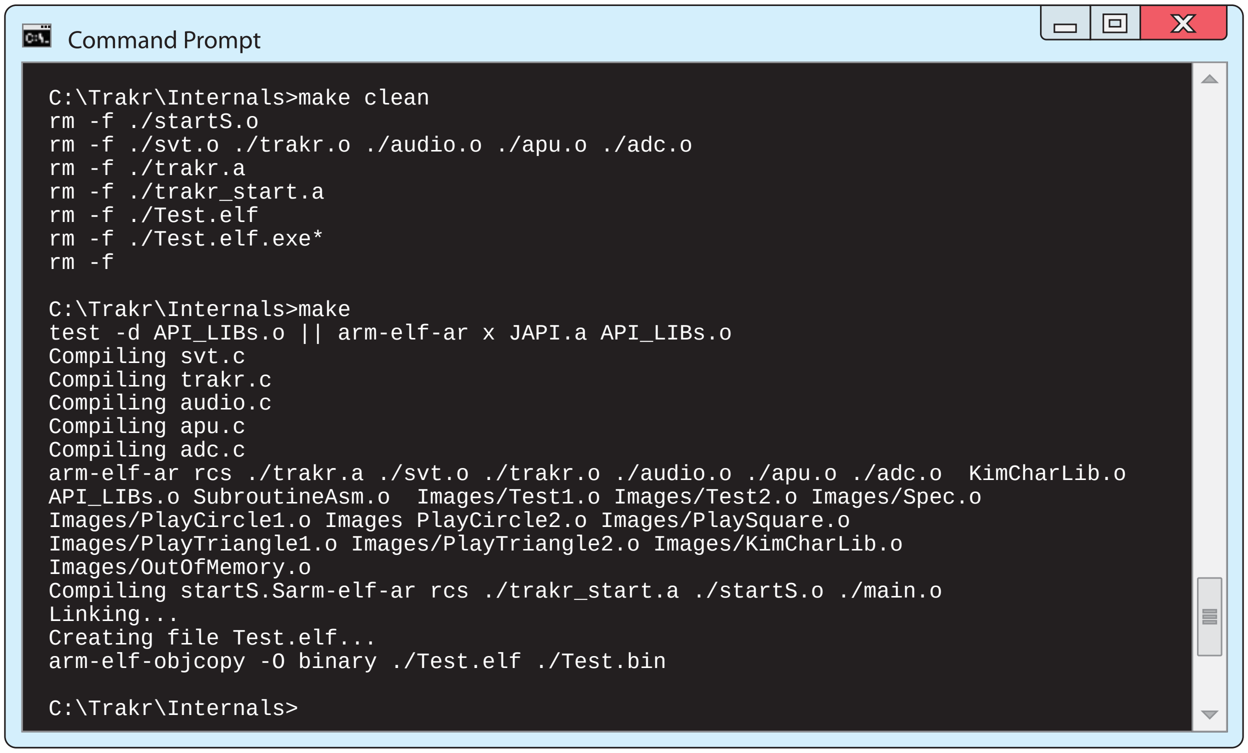
The compiler runs when you type "make" into a terminal program's command line so you need to make sure you navigate to the folder where the makefile you want to run is located. The compiler then follows any instructions in that "makefile" to convert the "app.c" file (the text file containing the source code) into a TRAKR program.

TRAKR TIP: You can start fresh and erase any old temporary files by typing "make clean".

The compiler only looks for the "app.c" file in the current folder so always name your source code app.c. Save your programs in separate folders with names which make sense. This helps to keep all the various "app.c" files organized. In the screen shot below, "make clean" is run from within C:\Trakr\Internals. It reports all the files it is removing before returning the prompt to you. "make" is then typed at the prompt and the compiler uses the "makefile" to assemble all the parts of your program (which may include many libraries, images and files specified by your code) into file to write to the TRAKR hardware. When done, the last line should look something like:

```
"arm-elf-objcopy -0 binary ./yourFileName.elf ./yourFileName.bin"
```

After that, the prompt should be returned to you.
Below is a screenshot of the Internals folder's makefile being run:



```
C:\Trakr\Internals>make clean
rm -f ./startS.o
rm -f ./svt.o ./trakr.o ./audio.o ./apu.o ./adc.o
rm -f ./trakr.a
rm -f ./trakr_start.a
rm -f ./Test.elf
rm -f ./Test.elf.exe*
rm -f

C:\Trakr\Internals>make
test -d API_LIBs.o || arm-elf-ar x JAPI.a API_LIBs.o
Compiling svt.c
Compiling trakr.c
Compiling audio.c
Compiling apu.c
Compiling adc.c
arm-elf-ar rcs ./trakr.a ./svt.o ./trakr.o ./audio.o ./apu.o ./adc.o  KimCharLib.o
API_LIBs.o SubroutineAsm.o  Images/Test1.o Images/Test2.o Images/Spec.o
Images/PlayCircle1.o Images PlayCircle2.o Images/PlaySquare.o
Images/PlayTriangle1.o Images/PlayTriangle2.o Images/KimCharLib.o
Images/OutOfMemory.o
Compiling startS.Sarm-elf-ar rcs ./trakr_start.a ./startS.o ./main.o
Linking...
Creating file Test.elf...
arm-elf-objcopy -O binary ./Test.elf ./Test.bin

C:\Trakr\Internals>
```

If the compiler encounters something it can not convert into the target format, it will stop trying and report the place at which it failed. This is often the first time you get a clue that there is a mistake in your code. By reading the compiler error and trying to identify what caused it, you can edit your code until the compiler gets through it with no errors.

TRAKR TIP: Connect the TRAKR to your computer via USB so you can run your code right away

The compiler omits all your code comments and streamlines your text into an application but it can add information as well.

The "#include" statement within your code is an instruction to the compiler to compile the file referred to. Every TRAKR application should have the line: #include "svt.h". svt stands for Spy Video TRAKR. It is the file which defines all the functions which are native to the TRAKR hardware. Without the "svt.h" file, the compiler would not know what to do with anything but regular C functions and the TRAKR would not be able to run.

STRATEGIES

Things may seem more complicated than they need to be, but consider that computers do not know anything which is not defined in advance for them. Even words which you might understand to have a single and clear meaning will have to be defined for the machine before it can do anything with them and every time we use a function or a term, it will have to look up the definition again. This means you must be VERY specific when you define something. Your definition will have to make sense in all the foreseeable instances in which it may be called. You can only use previously defined terms in your new definitions. You can then use these new parts to create even more complex structures.

Map Your Idea with a Flow Chart

When writing an app for the TRAKR, you can plan out its functions using a flow chart. When every element in the flow chart is simple enough so that you can navigate from your starting condition to the desired ending condition without any confusion, then you are ready to execute your plan.

If you drew a flow chart for drinking water, it would look like the chart in Fig. 1. This chart assumes there is water to drink, but what if there is none? You would need to add a thread for that. The chart in Fig. 2 shows the further evaluation of whether there is water or not along with the action to take in either possible state.

So, just as you can plan your own action with a flow chart, you can plan a program using elements interconnected by their logical relationships. A basic TRAKR program would look like Fig. 3. Each field would be expanded as you came to new things to evaluate or do. If you can plan your ideas like this and keep your chart updated as you work, you will be well on your way to being able to solve the most complicated programming challenges.

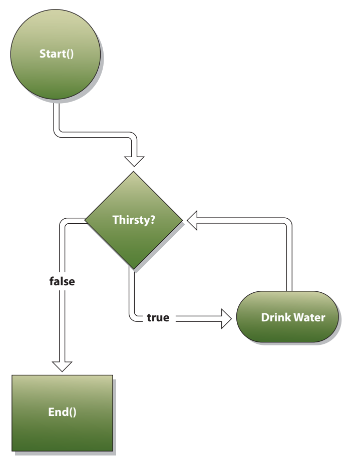


Fig. 1

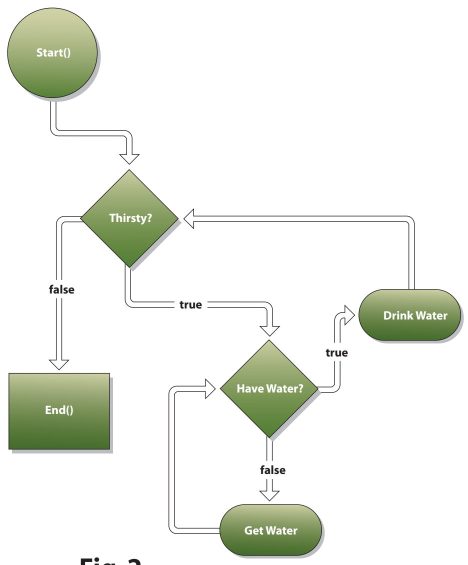


Fig. 2

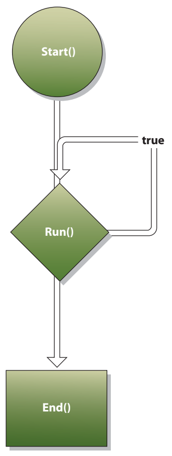


Fig. 3

Things to Remember

Before we jump into the examples, take some time to ponder a few things we have covered and how they can be used.

- Programming is Language
  - Come up with a great idea
  - Write it down in your own words
- Translate (code) your idea into C
- Computers can't infer
  - Syntax is key
  - Capitalization and spelling are important
- Stay organized
  - Names must match
  - Folders must be located as expected
- Be Patient and Relaxed
  - Be prepared to try everything a few times before it works
  - Go slow and pay attention to each step

- Take notes on what steps you have already done
- Looking for bugs is very much like looking for bugs
  - Stretch your body as often as you can remember
- That is where your brain lives
- Look away from the screen to see the forest through the trees

NOW IT'S YOUR TURN!

Ready to dig in to some code? The following examples are organized from the simple (so simple it doesn't even try to DO anything) to the complex. Even if you're new to programming, try running some code on your TRAKR by modifying these examples.

TRAKR TIP: Try storing your code on a USB flash drive. You can use data from previously written apps or save information to upload to another system later.

Examples

Sleeping: Simple Timing
Moving: Control the Motors
Motor Speed: Control the Speed of the Motors
Writing Text: Display Voltage and Current
Polling the Remote: Change the Button States
Using Night Vision: Turning the IR LED On and Off
Recording Audio: Record and Play Back
Logging to File: Writing Text to a File

Sleeping: Simple Timing

The program on the TRAKR runs as fast as it can. As soon as it finishes one operation it begins the next. It is often useful to slow things down. One way to do this is to simply stop the program for some specific amount of time. In this example, the Sleep() function stops the program by taking whatever integer value it gets and waiting that number of milliseconds (1/1000 of a second) before returning the processor to work on the next item.

```
// don't forget the svt.h file
#include "svt.h"
// define the Start function:
void Start()
{
  // nothing to do here...
}

bool Run()
{
  Sleep(1000);
  // do something
  Sleep(1000);
  // do something else
  return true;
}

void End()
{
}
```

Moving: Control the Motors

This example will show you how to program your TRAKR to move automatically. It will make the TRAKR take control away from the remote, pause for 1 second, spin counter-clockwise for 1 second, pause for 1 second, spin clockwise for 1 second and then allow the remote to control the TRAKR for 5 seconds before repeating.

To do this, you need to take and let go of control of the motors. Many of the functions which allow your code to control the TRAKR depend on you opening and closing a connection to a particular device. Sometime you can skip this step but, if you really want to be understood and get an answer the first time, it is often necessary to explicitly "open" a dialog with the OpenMotors() function. Another important reason for take control of devices is so that your processes will not be interrupted.

```
#include "svt.h"

void Start()
{
  // Take control of the motors:
  OpenMotors();
}
bool Run()
{
    // Pause 1 second
    Sleep(1000);
    // Set the Right motor speed to go full forward
    SetRightMotor(10000);
    // Set Left motor to go full backwards
    SetLeftMotor(-10000);

    // Pause 1 second
    Sleep(1000);
    // Stop both motors
      SetMotors(0, 0);

    // Pause 1 second
    Sleep(1000);
    // Set the Right motor speed to go backwards
    SetRightMotor(-10000);
    // Set Left motor to go forwards
    SetLeftMotor(10000);

    // Pause 1 second
    Sleep(1000);
    // Stop both motors
    SetMotors(0, 0);

  // Release the motors so the remote can regain control
  CloseMotors();
  // Do nothing for 5 seconds:
  Sleep(5000);
  // Take back control of the motors:
  OpenMotors();
  return true;
}
void End()
{
  // remember to let go before you finish...
  CloseMotors();
}
```

Motor Speed and Declaring a Variable

A **variable** is a name given to a place in the computer's memory where you can record and recall data. It is important to tell the machine what kind of data and how much of it you need to store so that enough room is set aside for it in advance. If you try to put a value into a variable which is not of the correct type, you will get an error, but even worse, you might lose data and not even know it.

This example will also declare a variable to store an integer. An integer is a positive or negative whole number. It can have no decimal point. A 32bit integer is a whole number between –2,147,483,648 and 2,147,483,647. You will use the variable to keep track of the last speed value and add or subtract or invert it as appropriate.

As you have seen, by setting the motors to 0 you are telling them to stop, 10,000 is full ahead and -10,000 is full reverse. As you might guess, 5,000 is half speed forward. In fact, any integer between -10,000 and 10,000 corresponds to a speed from fast to slow to fast again in the opposite direction.

This example uses the for() function to count from one value to another. This is a useful C function which will be a part of most any program. You can learn more about it from any C reference guide. It will serve you well. By counting from 0 to 1000 with a tiny pause between each iteration, we can make the motors go from stopped to full speed in a smooth ramp. This program below will set the TRAKR spinning faster and faster until it reaches full speed and then start slowing down until stopped. Then it will wait 5 seconds before repeating the program.

```
#include “svt.h”

void Start()
{
  // Take control of the motors:
  OpenMotors();
}
bool Run()
{
  // create and integer variable called “i”
  int i;
  // put a count from 0 to 10000 in “i”
  for (i=0; i<=10000; i++)
  {
    // Pause 1 millisecond
    Sleep(1);
    // Set the Right motor speed to the current count
    SetRightMotor(i);
   // Set Left motor to the negative of the current count
    SetLeftMotor(-i);
  }
  // count from 0 to 10000
  for (i=10000; i>=0; i--)
  {
    // Pause 1 millisecond
    Sleep(1);
    // Set the Right motor speed to the current count
    SetRightMotor(i);
    // Set Left motor to the negative of the current count
    SetLeftMotor(-i);
  }
  // Release the motors so the remote can regain control
  CloseMotors();
  // Do nothing for 5 seconds:
  Sleep(5000);
  // Take back control of the motors:
  OpenMotors();
  return true;
}
void End()
{
  // remember to let go before you finish...
  CloseMotors();
}
```

Writing Text: Display Voltage and Current

This example will take control of the remote display to show the data returned by the analog to digital converters (ADC) on board the TRAKR. The screen on the remote has a resolution of 160 pixels across by 80 pixels high. Everything you put on the screen needs to be placed by referring to the top left corner where we want to start drawing or writing as a set of coordinates. These coordinates are commonly referred to as "x" and "y"."x" is the number of pixels from a spot to the left side of the screen and "y" is the number of pixels from that spot to the top of the screen.

By giving your function these parameters, you can tell each function call to place its data at a specific place.

In the case of the DrawText() function, this data is text which is drawn starting from the first letter's top left corner at the "x,y" coordinates you specify and flowing to the right until it reaches the edge of the screen at which point it will wrap down and to the left and continue.

This example also incorporates the idea of checking for change to avoid needlessly re-writing the same data over and over. To do this you use two variables for each kind of data you want to evaluate. One variable to read the data into and another variable to store the last value. This way, by comparing them, you know when something has changed and can take action.

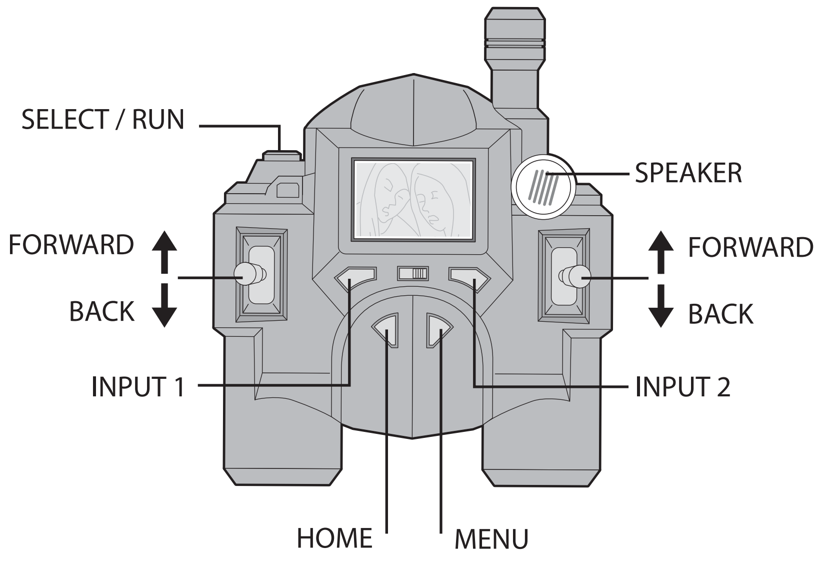
```
// This example will use the graphics functions to write
// the values of the ADCs to the Remote screen
#include “svt.h”

//declare some variables to use:
int mVolts;
int lastmVolts;
int LMC;
int lastLMC;
int RMC;
int lastRMC;

void Start()
{
  //take control of the display:
  OpenGraphics();
}

bool Run()
{
  // read values of ADCs:
  LMC=GetLeftMotorCurrent();
  RMC=GetRightMotorCurrent();
  mVolts=GetBatteryVoltage();
  //if values have changed:Update screen
  if (LMC!=lastLMC || RMC!=lastRMC||mVolts!=lastmVolts)
  {
    //clear graphcs buffer:
    ClearScreen();
    //load the buffer with some new drawing instructions:
    DrawText(5,5,”Battery: %d mV”, mVolts);
    DrawText(5,25,”Motor current”);
    DrawText(5,45,”in miliampres”);
    DrawText(5,65,”left: %d”, LMC);
    DrawText(85,65,”right: %d”, RMC);
    //Update buffer to screen:
    Show();
    //reset “last” values:
    lastLMC=LMC;
    lastRMC=RMC;
    lastmVolts=mVolts;
  }
  //if no change, wait a little while before checking again:
  else
  {
    Sleep(100);
  }
  return true;
}
```

```
void End()
{
CloseGraphics();
}
```

Polling the Remote: Change the Button States

This example checks all the buttons on the TRAKR remote and uses the display to show which are ones are pressed. The two functions which return button states are; GetRemoteKeys() or GetRemoteKeyStatus().

This example uses the GetRemoteKeys() function which sets an 8 bit variable to represent the button states at that moment. Remember; a bit is a single place to store a 1 or a 0 and 8 bits are in a byte. Bytes are often represented as decimal integers where each bit is a digit in a binary number. TRAKR takes advantage of this easy conversion to be able to store every possible combination of button presses as a single number value between 0 and 255. You can then use a bit comparison (which is the ampersand logogram, "&") between that number and a value of the "KEY_*" constants to check if that bit is high or low.



```
#include "svt.h"
int LastKeyState;

void Start(){
      OpenGraphics();
}

void End(){
      CloseGraphics();
}
// consolidating code by declaring a custom function:
void display(int keyState)
{
      ClearScreen();
      DrawText( 5, 0, “Key: %d”, keyState );
      //show the 8 bits of key         State as an integer
      int y=30; // variable to increment text display downward

      if(keyState&KEY_LEFT_BACK){
      // if bit in “keyState” associated with “KEY_LEFT_BACK” is high
      DrawText( 5, y, “LEFT BACK” );// draw text
      y=y+15; //move text “pen” down
}

if(keyState&KEY_LEFT_FORWARD){
      DrawText( 5, y, “LEFT FORWARD” );
      y=y+15;
}

if(keyState&KEY_RIGHT_BACK){
      DrawText( 5, y, “RIGHT BACK” );
      y=y+15;
}

if(keyState&KEY_RIGHT_FORWARD){
      DrawText( 5, y, “RIGHT FORWARD” );
      y=y+15;
}

if(keyState&KEY_INPUT1){
      DrawText( 5, y, “LEFT INPUT” );
      y=y+15;
}

if(keyState&KEY_INPUT2){
      DrawText( 5, y, “RIGHT INPUT” );
      y=y+15;
}

if(keyState&KEY_RUN){
      DrawText( 5, y, “GO” );
      y=y+15;
}

if(keyState&KEY_MENU){
      DrawText( 5, y, “MENU” );
      y=y+15; //move draw “pen” down
}

if(keyState==0){
      DrawText( 5, 30, “NONE PRESSED” );
}
Show();
```

```
}

bool Run()
{
  int KeyState = GetRemoteKeys();// load button states
  if(LastKeyState!=KeyState)// if buttons are in new state...
{
  display(KeyState);// show the names of pressed buttons
  LastKeyState=KeyState;// update the old state to the new
}
  Sleep(100); no need to keep the radio working constantly.
  return true;
}
```

Using Night Vision: Turn the IR LED On and Off

The TRAKR camera can perceive light beyond the range which is visible to us into the Infrared (IR) spectrum. The TRAKR has an IR light next to its camera. When it is active, there will be a faint red glow visible to the naked eye, but to the camera it is a bright flood of light. In this example, if you drive the TRAKR in the dark, the image on the TRAKR's remote control screen will appear rendered in gray tones with the brightness representing the amount of IR light being detected. If you were to drive the TRAKR in the light, the image would appear in full color.

```
#include "svt.h"
int BlinkSpeed=3000; // Variable to set frequency of blinking
bool state; // variable to toggle IR LED state

void Start()
{
  OpenIR(); // Take control of IR LED
}

bool Run()
{
  //As long as the INPUT1 key is not pressed...
  while (GetRemoteKeyStatus(KEY_INPUT1)==0)
  {
    if (ReadTimer()>BlinkSpeed)// If enough time has passed
    {
      state = !state;// toggle the value of "state"
      SetIR(state);// Set IR to new state
      ResetTimer();// reset timer to 0
    }
  }
// If the INPUT1 key is pressed:
  return false;
}

// When The Run() function returns false, the TRAKR calls the End()
```

```
function
void End()
{
  CloseIR();
}
```

Recording Audio: Record and Play Back

To record audio, a file needs to be created which will be written to. Since audio can take up a lot of space in memory very quickly, it works best with an SD card inserted in the TRAKR (you will write the file to the SD card). In this example, the audio will start recording 3 seconds of audio when the remote "GO" button is pressed. It will then play whatever was recorded when the "GO" button is pressed again.

```
// As always
#include "svt.h"

// substitutes "AudioFile" with a file path
#define AudioFile "A:\\Test\\Audio.wav"

int length = 10000; // recording length

void Start()
{
  OpenGraphics(); // we'll need the display
}

bool Run()
{
  // display some instructions:
  ClearScreen();
  DrawText( 35, 5, "Press Go" );
  DrawText( 35, 30, "to record" );
  DrawText( 25,55, "%d seconds", length/1000 );
  DrawText( 30, 80, " of Audio");
  Show();
  //wait for the keypress
  while (!GetRemoteKeyStatus(KEY_RUN))
  {
    Sleep(10);
  }
  // start recording
  ClearScreen();
  DrawText( 15, 30, "Recording" );
  Show();
  StartAudioRecording( AudioFile );
  // creates or overwrites file to write in

  ResetTimer();
  while( ReadTimer() < length )
  {
    Sleep( 10 );// wait while coolecting some sound
    WriteAudioData(); // write whatever has been recorded
  }

  StopAudioRecording(); //close recorded file
  // display finished message
  ClearScreen();
  DrawText( 15, 30, “DONE” );
  Show();
  Sleep(1500);
  // display instruction
  ClearScreen();
  DrawText( 35, 5, “Press Go” );
  DrawText( 25, 30, “to playback” );
  DrawText( 15,55, “Audio tecording” );
  Show();
  // wait for key press
  while (!GetRemoteKeyStatus(KEY_RUN))
  {
    Sleep(10);
  }
  // play back file
  ClearScreen();
  DrawText( 15, 30, “Playing” );
  Show();
  CloseGraphics();
  // free up our processor to do audio digitizing
  StartAudioPlayback( AudioFile ); // begin playing file
  // wait for audio to finish
  while( IsAudioPlaying() )
  {
    Sleep( 500 );
  }
  //report finished
  OpenGraphics();// open the display again
  ClearScreen();
  DrawText( 5, 80, “DONE” );
  Show();
  Sleep(2000);
  //run again
  return true;
}

void End()
{
  CloseGraphics();
}
```

Logging Your Code: Write Text to a File

A log is a list of entries made as events occur. Just as weather and the passage of time is logged in a ship's logbook, a program can use a text file to record events by writing to it as things happen. It is very useful to know that a section of code has run and in what order, or what a variable's value is at a specific time. In this example, you will count the iterations of Run() as well as log each press of the remote keys. A text file for logging is created on the flash memory card as soon as the Log() function is first called. The file is named "Trakr.log".

```
#include "svt.h"

int RunCounter=1; // variable to store numbr of Run() calls
int RemoteState; // integer to store RemoteState() into
int LastRemoteState; // remember the last to compare to
int secondCount=1; // counting seconds passed

void Start()
{
  ResetTimer();
  Log("Starting..."); //The first line of the "A:\Test\Trakr.log"
}

bool Run()
{
  //note as the seconds pass
  if (ReadTimer() >= secondCount*1000)
  {
    Log("%d Seconds", secondCount);
    secondCount+=1;
  }
  // Log when the Remote Keys Change
  RemoteState=GetRemoteKeys();
  if (LastRemoteState!=RemoteState)
  {
    Log("Run count %d, Remote state: %d", RunCounter, RemoteState);
    LastRemoteState=RemoteState;
  }
  // Check if time is up
  if (ReadTimer()>5000)
  {
    Log("Time is up");
    return false;
  }
  // Wait a moment and then increment the run count before going again
  Sleep(10);
  RunCounter+=1;
  return true;
}

void End()
{
  //On last Log Entry
  Log("Signing off.");
}
```

TRAKR Code Snippets

Theses TRAKR Code Snippets are an easy way to build an app using pre-written code for the TRAKR. The following code block does not work as a whole, but instead contains commonly used parts of code for you to copy into your own programs and modify.

```
// Go Forward, Stop
SetMotors(10000, 10000);
// Turn the left & right motors to full forward 10000
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

// Go Backward, Stop
SetMotors(-10000, -10000); // Turn the left & right motors to 10000
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

// Spin Clockwise
SetMotors(10000, -10000);
// Turn the left motor forward & right motors back
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

// Spin Counter Clockwise
SetMotors(-10000, 10000);
// Turn the left motor back & right motors forward
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

//Arc Left Forward
SetMotors(5000, 10000);
// Turn the right motor forward a little faster than the left
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

// Arc Right Forward
SetMotors(10000, 5000);
// Turn the left motor forward a little faster than the right
Sleep([Duration]); // Wait for the right number of milliseconds`
SetMotors(0, 0); // Turn the left & right motors off

// Arc Left Backward
SetMotors(-5000, -10000);
// Turn the right motor backward a little faster than the left
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off
```

```
// Arc Right Backward
SetMotors(-10000, -5000);
// Turn the left motor backward a little faster than the right
Sleep([Duration]); // Wait for the right number of milliseconds
SetMotors(0, 0); // Turn the left & right motors off

// Display Text
ClearRectangle(20, 10, 140, 40); // First clear the text region
DrawText(20, 10, “[Text]”); // Draw the text
Show(); // Show the graphics

// Display Image
ClearRectangle(40, 60, 120, 110); // First clear the image region
DrawImage(40, 60, [Image]); // Draw the image
Show(); // Show the graphics

// Display Battery Voltage
ClearScreen(); // First Clear the Screen
v = GetBatteryVoltage(); // Read the battery voltage into variable v
if (v < 5000) // Check the battery. If it’s low… (less than 5 volts,
//5000 millivolts)
Color RED; //Define the colors using RGB
Red.R=225;
Red.G=0;
Red.B=0;
Red.Transparent=0;
Color GREEN;
Green.R=0;
Green.G=225;
Green.B=0;
Green.Transparent=0;
Color WHITE;
White.R=225;
White.G=225;
White.B=225;
White.Transparent=0;
{
  SetTextColor( Red ); // Set the color to RED
}
else // else
{
  SetTextColor( Green ); // Set the color to GREEN
}
DrawText(20, 10, “Batt : %d”, v);
// Draw the fixed text, with the number v replacing the %d
Show(); // Show the graphics
SetTextColor(WHITE); // Put the text color back to white
```

```
// Wait for a Key
while(! GetRemoteKeys() = 0)
// Keep doing the wait until the key is pressed
{
  Sleep(100); // Wait for 100ms, then loop again!
}

// Wait for a Key to Release
while(ReadKey([Key]))
// Keep doing the wait until the key is NOT pressed
{
  Sleep(100); // Wait for 100ms, then loop again!
}

// Go Back
SetMotors(-1000, -1000);
// Turn the left & right motors to full forward 100

// Stop //
SetMotors(0, 0); // Turn the left & right motors off

// Wait
Sleep([Duration]); // Wait for the right number of milliseconds

// Add Message
DrawText(20, 10, “[text]”); // Draw the text at screen co-ord (20, 10)

// Clear Screen
ClearScreen(); // Clear the display

// Clear Image Region
ClearRectangle(40, 60, 120, 110); // Clear the image region

// Clear Text Region
ClearRectangle(20, 10, 140, 40); // Clear the text region

// Show Screen
Show(); // Show all graphics since last Show

// Read Battery
v = GetBatteryVoltage(); // Read the battery voltage into variable v

// Read Timer
v = ReadTimer(); // Read the timer value into variable v

// Reset Timer
ResetTimer(); // Reset the timer
```

```
// Display Number
DrawText(20, 10, “[Text] : %d”, v);
// Draw the fixed text, with the number v replacing the %d
```

Data Types
Hardware and Software

- TRAKR and remote control
- USB cable
- Computer with one of the following operating systems: Windows, Mac, Linux
- Software: text editor, command line terminal program
- Files: TRAKR folder

Function	Description	Size	Range
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int`	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int`	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1bit	true or false
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 or 4 bytes	1 wide character

- The TRAKR's on-board computers are a pair of ARM 9 processors (one in the TRAKR and one in the remote) with 8MB of RAM. These microprocessors are preloaded with the TRAKR firmware which allows you to code in C and use the library of TRAKR functions.
- The computer you use to upload code to the TRAKR must support USB. You must have administrative privileges so that you can move, write and erase files in the root folder.
- You should use a text editor which is able to save files as plain text. there are many editors for coding which have extra features for organizing your ideas and formatting your code but the free editors which come with most computers are fine as well.
- In order to get your code from a text editor on your computer to the TRAKR, you will need to compile your text file into a binary file (.bin) and then upload it via a USB cable connection. The program which does this is run from a command line prompt in a terminal program. In Windows, this is the "Command Prompt" program in the start menu. The compiler is run from the command prompt by typing "make" while in the folder where your app is located. The compiler then reads the "makefile" in that folder for instructions on how to name and where to place the new .bin file it will create.
- All the files you need are available to download from the spytrakr.com website. The download, called the App Primer, includes installation instructions, templates for making your own apps and all the example apps used in this introduction.
- Follow all the instructions contained in the download for your operating system. Pay special attention to the location of the folders and the order of each operation.

TRAKR Function Reference

The TRAKR Function Reference is an essential guide to all the native TRAKR functions.

TRAKR TIP: Download all of these tools here.

Function	Description	Syntax	Parameters	Returns
ClearRectangle()	Clears an area in the buffer defined by top/left and bottom/right coordinates	ClearRectangle(int, int, int, int)	4 ints, top/left x,y and bottom/right x,y	none
ClearScreen()	Clear all graphics from screen buffer	ClearScreen()	none	none
CloseFile()	Closes the specified file	CloseFile(File f)		none
CloseFileSystem()	Shuts down the file system	CloseFileSystem()	none	none
CloseGraphics()	Shuts down the graphic system	CloseGraphics()	none	none
CloseImageRegister()	Completes the Image Register function	CloseImageRegister()	none	none

Function	Description	Syntax	Parameters	Returns
CloseIR()	Releases control of the InfraRed LED on the Trakr car	CloseIR()	none	none
CloseMotors()	Releases Motor control	CloseMotors()	none	none
CreateFile(String pathName)	Creates and opens the named file	long CreateFile(String pathName)	path name of the file	
DeleteFile(String pathName)	Deletes the named file	long DeleteFile(String pathName)	path name of the file	
DrawImage()	Renders an image into buf-fer starting at coordinates and applying transparency	DrawImage(int, int, int, Color.transparent)	int imageIndex, int x, int y, Color transparent	none
DrawRectangle()	Draws a rectangle from top/left to bottom/right coordinates	DrawRectangle(int, int, int, int, Color rgba)	lt, ty, rx, by	none
DrawText()	Begins writing text at coordinates	DrawText(int, int, String)	int x, int y, String fmt, ...	?
FlushFile()	Completes all previous file operations	long FlushFile(File f)		
GetBatteryVoltage()	Returns the voltage level in millivolts from the Car Batteries	GetBatteryVoltage()	none	int
GetLeftMotorCurrent()	Returns the Left Motor cur-rent use in milliamps	GetLeftMotorCurrent()	none	int
GetRemoteKeys()	Queries all the remote buttons	GetRemoteKeys()		int
GetRemoteKeyStatus()	Queries a specified remote button	GetRemoteKeyStatus(int key)	int	bool
GetRightMotorCurrent()	Returns the Right Motor current use in milliamps	GetRightMotorCurrent()	none	uint16
IsAudioPlaying()	Checks status of audio file playback	IsAudioPlaying()		bool
OpenFile(String filename)	Opens a specified file for reading or writing to	File OpenFile(String filename)	path name of the file	
OpenFileSystem()	Opens the file system for use	long OpenFileSystem()	none	
OpenGraphics()	Opens the buffer for the display	OpenGraphics()	none	none
OpenImageRegister()	Signals the beginning of an operation to register images	OpenImageRegister()	none	none
OpenIR()	Takes control of InfraRed LED on Trakr car	OpenIR()	none	none
OpenMotors()	Takes control of motors	OpenMotors()	none	none

Function	Description	Syntax	Parameters	Returns
ReadFile()	Reads (at current file position) up to a specified number of bytes into a buffer, and writes the number of bytes read into a variable	long ReadFile(File f, void* buffer, bufferLength)		
ReadTimer()	Returns the number of millisecond since last ResetTimer() call	ReadTimer()	none	int
RegisterImage()	Registers an image and gets back a handle	int RegisterImage(void* image, int size)	pointer to bitmap, length	int
ResetTimer()	Sets value of internal timer clock to 0	ResetTimer()	none	none
SeekFile()	Sets pointer to a place in file (byte count)	long SeekFile(File f, long pos)		
SetIR()	Sets the state of the Infra-Red LED on the Trakr car	SetIR(bool)	bool, 1 for on, 0 for off	none
SetLeftMotor()	Sets speed of left motor. -10,000=full reverse, 0=stop, 10,000=full ahead	SetLeftMotor(int speed)	int (-10,000 to 10,0000)	none
SetLineWidth()	Sets the pixel width of lines used in drawing	SetLineWidth(int)	int, number of pixels	none
SetMotors()	Define speed of left and right motors at once with 2 integers	SetMotors(int leftSpeed, int rightSpeed)	int, int (-10,000 to 10,0000)	none
SetRectangle()	Draws a solid rectangle	SetRectangle(int lx, int ty, int rx, int by, Color rgba)	4 ints, top/left x,y and bottom/right x,y AND a Color	none
SetRightMotor()	Sets speed of right motor. -10,000=full reverse, 0=stop, 10,000=full ahead	SetRightMotor(int speed)	int (-10,000 to 10,0000)	none
SetScreen()	Set screen color and transparancy	SetScreen(Color)	Color variable structure	none
SetTextColor()	Sets color and transparency of text	SetTextColor(Color)	Color variable structure	none
Show()	Updates the display with all graphics in buffer	Show()	none	none
Sleep()	Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)	Sleep(uint32 us)	the number of mil-liseconds to pause	none
StartAudioPlayback()	Begins playing file at specified address	StartAudioPlayback(char* filename)	the address of a WAV file	bool
StartAudioRecording()	Creates file at specified location, and starts writing audio to buffer internal	StartAudioRecording(char* filename)	path name of the file	bool

Function	Description	Syntax	Parameters	Returns
StopAudioRecording()	Stops writing audio data to the buffer, and closes file	StopAudioRecording()		none
WriteAudioData()	Moves recorded data from buffer to file, clears buffer	WriteAudioData()		bool
WriteFile()	Writes bytes in buffer to file at current file position	long WriteFile(File f, void* buffer, uint32 bufferLength, uint32* bytesRead)		