CONTACT: *A Symbiotic Preprocessor Facility for the Propeller IDE*

Phil Pilgrim (propeller@phipi.com)

Introduction

The goal of this project is to integrate a preprocessor facility into the Propeller IDE in order to let the user incorporate new features into Spin and Assembler in as transparent a fashion as possible. Such features can include conditional compilation, macros, large memory model assembly, new Spin data structures, and any other enhanced source code features translatable into code the Propeller IDE can compile. Although true integration is impossible without access to the Propeller IDE's source code, it's still possible to create a "symbiotic" add-on that emulates how a truly integrated facility would look and behave. I called it "CONTACT" because, in the days before electric starters, that's what the pilot would shout to the person spinning the prop, once he'd engaged the ignition and before the propeller could be spun. It also reflects the relationship of the program's single control to the IDE.

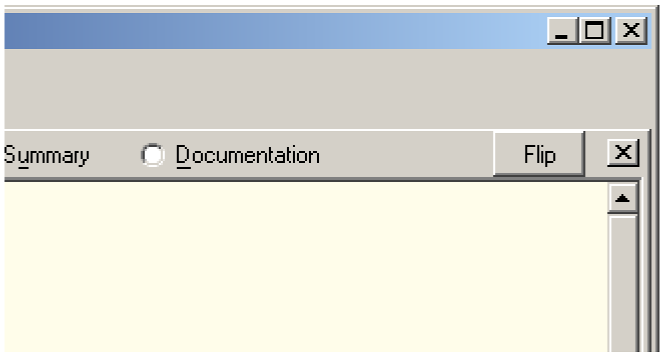
Installation

The ZIP file includes three **exe** files, which should all be extracted into the same directory:

- **contact.exe**: The preprocessor "symbiote".
- **uppercase.exe**: A demo preprocessor program that converts all text to upper case.
- **nobozos.exe**: A demo preprocessor that generates an error upon encountering "bozo" in the text.

Startup

Contact.exe can be started either before or after the Propeller IDE has been started. If the IDE is not running, **contact** will look for the latest version on the user's hard drive and start it up. Once this has been done, a button will appear above the IDE's edit window, as shown here:



The purpose of this button is to flip back and forth between the original code shown in the window and a preprocessed version that the IDE can compile. The button isn't actually attached to the IDE, but sticks by it in the same relative position, as the IDE window is moved about onscreen. Occasionally, if the IDE is not the topmost window on the desktop, the button will disappear, only to reappear instantly when the IDE is brought to the fore. This behavior is why the program is called "symbiotic" rather than "integrated": it clings to its host like a remora to a shark but isn't really part of the IDE *per se*.

Defining the Preprocessor

In order for CONTACT to know what to do when the Flip button is pressed, there needs to be a section in the user's source that defines the desired action. This is found in the **PRE** section, which must be the first section in the user's program, ahead of all **CON**, **VAR**, **OBJ**, **PUB**, **PRI**, and **DAT** sections. In fact, *the word **PRE** must begin on the first line and in the first column of the user's program for CONTACT to recognize it, and its list of commands must be indented at least one space*. The reason for specifying the preprocessor's action here is that each source program may require a different preprocessor to handle its particular needs. What the **PRE** section defines is a batch program to invoke the user's preprocessor. Any programs listed here may have their paths omitted if they're in the same directory as **contact.exe**. Otherwise, the path information must be included. Here's an example:

```
PRE

  @echo off
  preprocess.exe %1 %2
```

The program **preprocess.exe** will receive two arguments on its command line, which are the names of temporary files. The first file, designated "%1", contains the unprocessed source code. The second file, "%2" is where the processed code (or an error message) needs to be written. Preprocessors that accept source code on STDIN and output on STDOUT can also be accommodated, as in the following example:

```
PRE

  @echo off
  type %1 | preprocess.exe >> %2
```

The output of the **type** command, which is normally used to display a file's contents, is piped to **preprocess.exe** through STDIN. The output of **preprocess.exe** (via STDOUT) is then written to the output file. In this case, **preprocess.exe** doesn't need to know anything about either file.

By using STDIN and STDOUT, it's possible to chain several preprocessors together in sequence. Suppose, for example, that there are different preprocessors for conditional compilation and for macro expansion. You could use both, in sequence, like this:

```
PRE

  @echo off
  type %1 | conditionals.exe | macros.exe >> %2
```

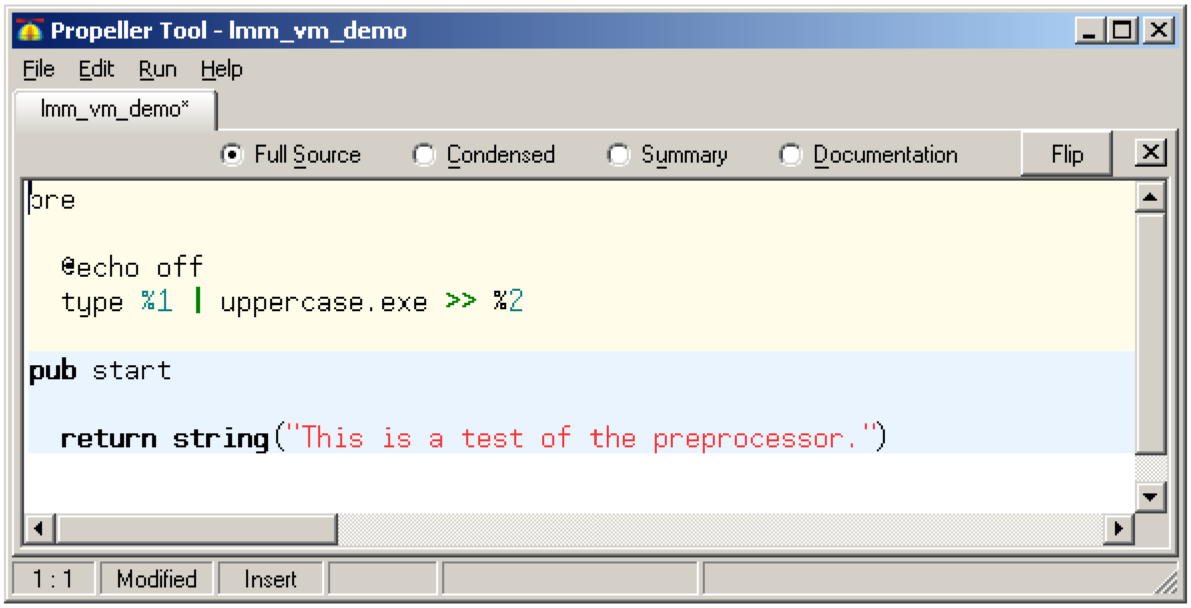
Here, the output of **conditionals.exe** is piped into **macros.exe**. In short, anything that's permitted in a batch file is permitted here.

Operation of the Flip Button

The Flip button takes the code in the current edit window and either sends it to the defined preprocessor or undoes the prior preprocessing so the original code can be edited. It is able to keep track of the code's state, since CONTACT, once it receives preprocessed code, tacks the original code onto the end in a special block comment. If it sees this comment when Flip is pressed, it simply jettisons the processed code and restores the original code from the comment. Therefore, the code in an edit window can be saved in either state, since the code itself contains all the information necessary to go back and forth.

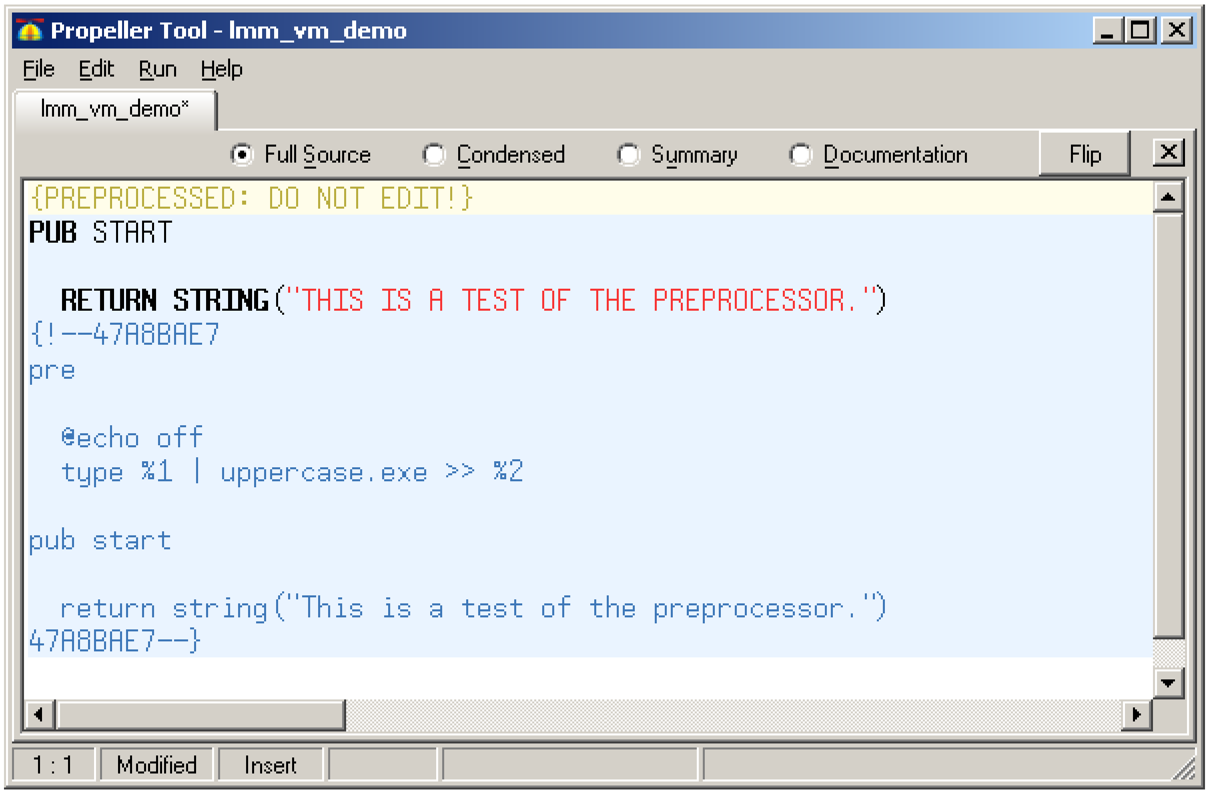
Note: All code sent to a preprocessor is stripped of carriage returns. The normal Windows carriage return/linefeed sequences are replaced, instead, by linefeeds (newlines) only.

Here's an example. In the first window, you can see the original, unprocessed code, along with the preprocessor definition of a program (included as a demo) that converts all lower-case letters to upper case:



```
pre

  @echo off
  type %1 | uppercase.exe >> %2

pub start

  return string("This is a test of the preprocessor.")
```

After Flip is pressed, a program that the Propeller IDE can compile appears in its place:



```
{PREPROCESSED: DO NOT EDIT!}
PUB START

  RETURN STRING("THIS IS A TEST OF THE PREPROCESSOR.")
{!--47A8BAE7
pre

  @echo off
  type %1 | uppercase.exe >> %2

pub start

  return string("This is a test of the preprocessor.")
47A8BAE7--}
```

Notice that the **PRE** section has been removed from the preprocessed code. (In fact, it's removed from the code sent *to* the preprocessor.) Also note that CONTACT has inserted a warning at the top of the code and has included the original code (with the PRE section intact) at the end, in a specially-formatted block comment. If one were to press Flip again, this original code would be restored. The hex number in the block comment represents the date and time that the conversion was performed. Its main purpose, though, is to create unique, matching comment delimiters for CONTACT to recognize.

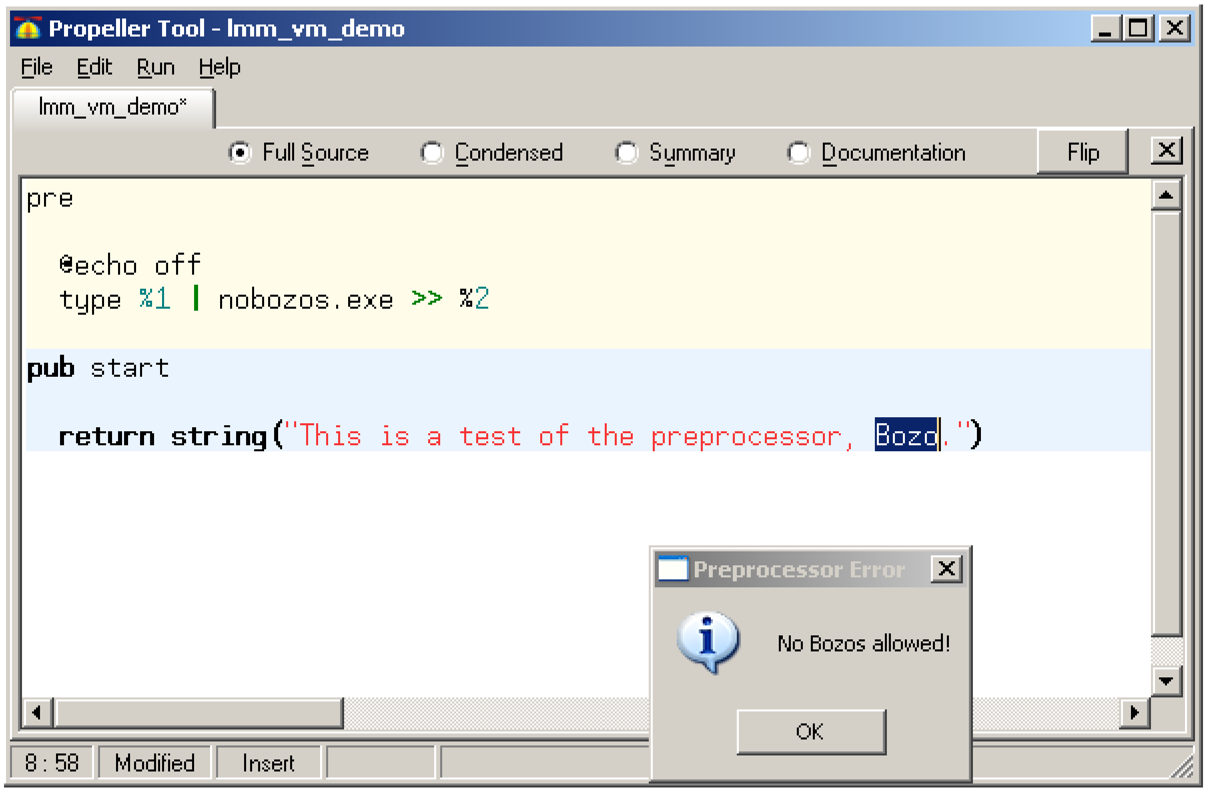
Error Reporting

If the preprocessor detects an error in the unprocessed code and can't continue, it can abort with an error message instead of (or in addition to) sending or writing processed code. The error message must be of the following form:

```
{PREPROCESSOR ERROR: <row>,<col>,<len>,<message>}
```

Here, **<row>** is an integer specifiying which row of the original source (beginning with row 1) the error occurred in; **<col>** is the column number (beginning at column 1) where the offending text begins; and **<len>** is its length in characters. **<message>** is any string not containing braces {} that describes the error. When CONTACT sees a block comment like this coming back from the preprocessor, it will not alter the original source but, rather, highlight the offending text, per the data in the error block, and display the error message in a popup window.

In the following example, the preprocessor (included as as demo) searches for any occurrence of the character sequence "bozo" (case-insensitive). If it finds one, it flags it as an error. Here's what the screen would look like when it does:



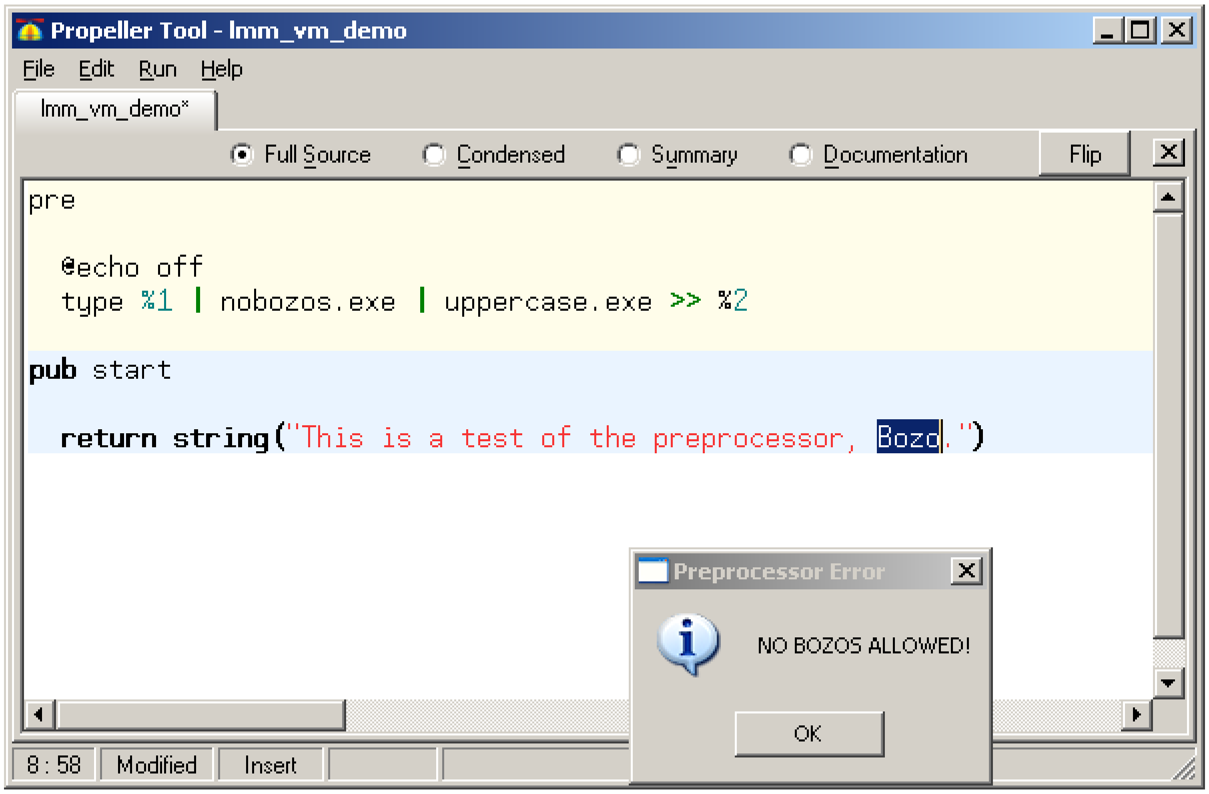
```
pre

  @echo off
  type %1 | nobozos.exe >> %2

pub start

  return string("This is a test of the preprocessor, Bozo.")
```

No Bozos allowed!

In this example, the preprocessor returned the following string embedded in the partially processed code:

```
{PREPROCESSOR ERROR: 8, 54, 4, No Bozos allowed!}
```

If you chain multiple preprocessors together, you may want to make sure that those “down the line” can deal with errors generated further upstream. Here’s an example where that isn’t the case:



```
pre

  @echo off
  type %1 | nobozos.exe | uppercase.exe >> %2

pub start

  return string("This is a test of the preprocessor, Bozo.")
```

NO BOZOS ALLOWED!

The program **uppercase.exe** processes the output from **nobozos.exe**. But that output is an error message. Nonetheless, **uppercase.exe**, being oblivious, converts the error message to upper case!

Example Preprocessor Code

Both **uppercase.exe** and **nozos.exe** are written in Perl (free from www.activestate.com) and converted to exe files via **perl2exe** (available from www.indigostar.com). Here is **uppercase.pl**:

```
undef $/;             #Set up to snarf the input as one big string.
print uc <STDIN>      #Output the input as uppercase.
```

And here is **nobozos.pl**:

```
my $lineno = 0;
while (<STDIN>) {
  $lineno++;
  if (m/bozo/i) {
    print "{preprocessor error: $lineno, ", length($`) + 1, ',', length($&), ',',
    "No Bozos allowed!}"
  } else {
    print $_
  }
}
```

Issues without Answers

There are a couple issues with CONTACT which, though not fatal, beg to be solved. The first is its handling of Unicode. In short, it doesn't. Any of Spin's special graphics characters will be converted to question marks, both in the original code and in the code sent to the preprocessor. The latter is probably preferred, since not all preprocessor writers will want to be bothered with Unicode. But it would be nice to be able to preserve it in the original.

The second is the appearance of an annoying console window during preprocessing. It seems as if it has to be there for Windows to process a DOS batch file, even if there's nothing to display in it. If the console window is omitted or iconified, the batch processing simply does not take place.

Future Directions

Part of the incentive for writing CONTACT is to show what *might* be accomplished if preprocessor hooks were integrated directly into the Propeller IDE. The user interface demonstrated here is one of many possibilities but includes some necessary features:

- Complete freedom for the user to select and/or write his own preprocessors.
- Integration of the preprocessor definition into the source code (via the **PRE** section), so different sources can use different preprocessors without having to switch them manually.
- Complete source code recovery without the use of auxiliary files, by means of an appended comment block.
- Ability to report and highlight preprocessor errors in the original source code.
- Simple, one-button user interface.

A facility like this, if included in the Propeller IDE itself, could add immeasurably to its value.