Programming Guide

Sean D'Epagnier

1

Published by Sean D'Epagnier

Table of Contents

1 The dataparser

1.1 What is it

The dataparser generates code compiled onto the embedded target. It is a complete target-side reflection interface. This allows various versions of the target software to be used with the same host viewer application (dataviewer or dataclient).

1.2 Building

The dataparser runs in two modes, compile and link. To compile, run with no arguments, and pass the preprocessed source code to it via stdin, the output via stdout is the generated code to be passed in for a link step. This can be stored to a file, and can be concatinated with other output files to make libraries. To link run dataparser with arguments of all of the generated files. This will produce datagen.c and datagen.h. The build system fully supports the dataparser. The datagen scheme contains accessors and operators which can be defined within ordinary c code with the macros DATA_ACCESSOR and DATA_OPERATOR.

1.3 Accessors

For each accessor the parser generates a function that is inserted into a table. An example of an accessor to expose a uint8_t:

```
uint8_t x;
DATA_ACCESSOR(name=x type=uint8_t mem=sram varname=x)
```

NOTE: x must not be declared static since the generated code needs to access it After the code is recompiled the name x will be available through the external interface.

1.4 Operators

The parameters to the macro are called attribute pairs. The name=value pairs are used in a matching algorithm to determine which operators are applied to the data for a given operation. Operators define small sections of code that should be performed. As an example, an operator that increments a variable:

```
DATA_OPERATOR(inc, varname,
    varname++;
)
```

Within the body of the operator, the attributes expand to their values. In this case the operator will be provided for any ACCESSORS that have a varname attribute. Assuming we don't want values greater than 1000, We could make the operator more robust by warning the user about overflow:

```
DATA_OPERATOR(inc, varname,
    if(varname > 1000)
        printf("Warning, overflow\n");
    varname++;
```

```
)
```

The operator will only be used if the varname attribute is set for the accessor. If a more specific operator is defined, the more specific operator is favored. eg:

```
DATA_OPERATOR(inc, mem=eeprom type=int16_t varname,
    int16_t val = eeprom_read_word(&varname);
    val++;
    eeprom_write_word(&varname, val);
)
```

The values for operator attributes are treated as extended POSIX regular expressions. This way, something like "type=u?int16_t" would match uint16_t as well as int16_t.

1.5 Standard Attributes

Because there is no cost in having operators that are not used, it is useful to have a library of operators. The parser does not care what names are used for attributes, however certain names are chosen for the basic library to aid in code reuse. If the only operations required are in the basic library there is no need to define any additional operators.

1. type – basic types: float, char stdint.h types: uint8_t, int8_t, uint16_t, int16_t, uint32_t, int32_t
2. mem – sram, eeprom, flash, etc..
3. varname – This attribute specifies the variable defined in the code that contains the data, it is not specified for accessors with no underlying data.
4. arraylen – Used by arrays to specify the length
5. writable – enable write operations, not possible for flash.

1.6 Standard Operators

The standard operators are typically applied remotely and are used to read and write data. name – Request the name of the data, if this attribute is not defined, it will be hidden from typical user interfaces. In the definition it is defined as _name which hides it from the user but is still accessible through DATA_OPERATOR_name in other operators.

1. get – Get specified data
2. set – Set specified data
3. display – Typical operation implemented for data that should display as soon as it is read (eg sensor data)
4. ops – Query supported operations
5. values – List possible values, if not implemented then there is no such list. This is typically implemented for types like enumerants or booleans.
6. clear – clear or reset the accessor to a default value

1.7 Matching

Attributes define which operations are used. Each attribute may be a name=value pair, or only the name. If only the name is given, it is defined with no value. For the purposes of

matching, an operator attribute without a value matches any value. An accessor attribute without a value only matches by an operator attribute without a value. Operator attributes may contain posix regular expressions to aid in complex matching situations.

1.8 Directories

Directories are designated by two attributes: dir and dirname. The dir attribute determines what directory the element is in; the dirname attribute defines an attribute as a directory. If the dir attribute is not set, or set to root, then the accessor appears in the root directory.

1.9 Protocol

It is typically useful to be able to jump to the bootloader, the driver listens for \e, and on the next character jumps to the bootloader section. No data requests should include this character. Because it is often useful to send data other than simple replies to data requests, for example asyncronous sensor updates. To distinquish between asyncronous and regular data, look for a leading '\0', this is the start of a reply terminated by the prompt "$-> ". This way it is possible to use normal console communication without problem. You may use the DEBUG macro to send debugging information. Use DATA_OUTPUT to send asyncrounous data, unless it is in the context of a request. These macros can be used the same as printf.

2 The menu system

2.1 Limitations

The menu system is ment to give the user a set of possible states each one starting from the main menu and working up. You will always exit to the page from which you entered. It is possible to spoof this, but it is not the overall design.

2.2 Proto threads

The menu system makes heavy use of protothreads. Understanding them is required to be able to modify this code. Protothreads are elegantly implemented as C macros which basically emit switch or goto statements. The gcc extention addrlabels is used so that we don't lose the ability to use swtich statements within protothreads, be aware of this.

Protothreads provide the ability to write non-blocking code with linear execution. This way the menu system can be run one frame at a time, and return control to other functions.

2.3 Benefits

The protothreads essentially log in static variables the position in each function they are in. This way the functions can return, and when they are called again, they jump to wherever they were last. The added benefit is before the jump each function can execute its rendering technique. This is evident in any thread function which code before PT_BEGIN. This allows the lcd screen to be easily subdivided and rearranged assigning each part of the screen to a different thread while allowing them to be updated using their own render routine.