

# Wiring your Hume

Kevin Hammond\*      Greg Michaelson†

November 29, 2001

## 1 Normal Wiring

Wiring in Hume is used to define the connections between boxes within a Hume program, and the connection of that program to external streams of data. The normal syntax for wiring is currently:

```
<wiredecl> ::= "wire" <boxid> <ins> <outs>
```

where <ins> and <outs> are either tuples of links, or just a single link.

```
<ins>/<outs> ::=      "(" <link1> ", " ... ", " <linkn> ")"  
                | <link>
```

A link attaches inputs to outputs, or vice-versa, or attaches an input or output to a globally named stream. The links to a given box are specified positionally for the inputs/outputs of that box.

```
<link> ::=      <boxid> "." <id>  
                |      <streamid>
```

The first form defines a box input or output, the latter attaches a box input or output to a stream. The predefined streams are given below

Stream	Attached
StdIn	standard input
StdOut	standard output

For example

---

\*School of Computer Science, University of St Andrews, North Haugh, St Andrews, UK.  
**Email:** kh@dcs.st-and.ac.uk

†Dept. of Computing and Electrical Engineering, Heriot-Watt University, **Email:** greg@cee.hw.ac.uk

```

box parity in ( input, par :: short ) out ( output, par' :: short )
match
  ( i, v ) -> let res = i + v in ( res, if even res then 1 else 0 )
;

```

```
wire parity ( StdIn, b.par' ) ( StdOut, b.par );
```

Initially-clauses may be attached to inputs. This is useful to initialise streams or values that are internal to a box as `par/par'` above.

```
wire b ( StdIn, b.par' initially 0 ) ( StdOut, b.par );
```

## 2 Templates and Replication

Box templates are defined similarly to boxes:

```

template <templateid> <ins> <outs>
<body>
;
\begin{verbatim}

```

where `\verb=<ins>=` and `\verb=<outs>=` are tuples of inputs and outputs respectively. Unlike a box, a template never generates a runtime process, and may be polymorphic. A template may be instantiated one or more times to yield concrete boxes.

```

\begin{verbatim}
template t in ( input, par :: short ) out ( output, par' :: short )
match
  ( i, v ) -> let res = i + v in ( res, if even res then 1 else 0 )
;

```

```

instantiate t as b1;
instantiate t as b2;

```

```

wire b1 ( StdIn, b1.par' initially 0 ) ( b2.input, b1.par );
wire b2 ( b1.input, b2.par' initially 0 ) ( StdOut, b2.par );

```

Boxes may also be replicated to give other (identical) boxes.

```
replicate box1 as box2;
```

A common requirement is to replicate a box or instantiate a template several times. A shorthand form can be used for this:

```
instantiate <templateid> as <boxid>*<exp>;
```

where `exp` yields a compile-time constant integer,  $n$ , where  $n \geq 1$ . For some free variable, `i`, this is exactly equivalent to the following form:

```
for i = 1 to <exp>
    instantiate <templateid> as <boxid>{i}
;
```

Wiring declarations may be included within loops. The general form of a wiring loop is:

```
for <id> = <expr1> to <expr2>
  [ except ( <eexpr1>, ..., <eexprn> ) ]
  <wiredecl>;
```

Within the loop, annotations may be used to index variants of names. The annotations (which are enclosed in braces – “” ... ””) are constant integer expressions formed from wiring loop variables, integer literal constants, integer addition, subtraction, multiplication, division, and remainder operations, named compile-time constants declared in constant declarations, and expansions of macros that have been declared in macro declarations.

```
constant bmax = 8;

macro next n = n + 1;
macro prev n = n - 1;

instantiate t as b*bmax;

for i = 2 to bmax-1
    wire b{i} ( b{prev i}.output, b{i}.par' ) ( b{next i}.input, b{i}.par );

wire b1 ( StdIn, b1.par' ) ( b2.input, b1.par );
wire b{bmax} ( b{prev bmax}.output, b{bmax}.par' ) ( StdOut, b{bmax}.par );
```

It is possible to nest for loops, and loop variables may be used within inner wiring declarations.

## 2.1 Except-Clauses

Except-clauses are used to exclude some values from the loop variable. For example

```
constant min = 1;
constant max = 8;
constant bypass = 3;

instantiate t as b*max;
```

```

for i = min+1 to max-1 except ( bypass )
    wire b{i} ( b{i-1}.output, b{i}.par' ) ( b{i+1}.input, b{i}.par );

wire b{min} ( StdIn, b{min}.par' ) ( b{min+1}.input, b{min}.par );
wire b{max} ( b{max-1}.output, b{max}.par' ) ( StdOut, b{max}.par );

```

### 3 Wiring Macros

Wiring macros allow generic wiring templates to be defined. Within the wiring macro, the parameters may be used to specify the names of boxes or input/output links. For example,

```

constant RingSize = 8;

macro predR i = (i-1) % RingSize;
macro succR i = (i+1) % RingSize;

wire Track ( this, prev, next ) =
    wire {this} ( {this}.value', {prev}.outval, {next}.inctl )
                ( {this}.value, {next}.outctl, {prev}.inval )
;

for i = 0 to RingSize
    wire Track ( Ring{i}, Ring{predR(i)}, Ring{succR(i)})
;

```

The syntax of a wiring macro definition is:

```
<wiredecl> ::= "wire" <macroid> <ids> "=" <wiredecl>
```

Wiring macros are instantiated by passing in concrete parameters to the macro

```
<wiredecl> ::= "wire" <macroid> <args>
```

Depending on usage, these arguments may be either box names or names of inputs/outputs. At present, it is not possible to use unrestricted values such as integers as arguments to wiring macros.

*Note to Greg: the implementation is not completely debugged at present and may loop or give odd effects in certain cases!*

### 4 Possible Extensions and Changes

A more flexible wiring primitive might be:

```
<wiredecl> ::= "wire" <linko> "to" <linki>
```

meaning that the output of <linko> should be wired to the input of <linki>. This primitive would allow more flexible wiring. For example, a wiring loop could set up a number of wires.

The track layout example makes extensive use of named records to set initial values. For example,

```
initial Ring{Train1Pos} { value = Just "Train1" };
```

Such uses can considerably simplify initialisation. This has not yet been implemented. The full syntax would be:

```
<wiredecl> ::= "initial" <boxid> "{" <wireinits> "}"  
<wireinits> ::= <id> "=" <expr> [ "," <wireinits> ]
```

## Acknowledgments

We would like to acknowledge the input provided by numerous visits to B+Q, Wickes, and C.G. Anderson, Cupar.