Lists.php

```php
<?php

namespace dokuwiki\Parsing\Handler;

class Lists extends AbstractRewriter
{
    protected $listCalls = array();
    protected $listStack = array();

    protected $initialDepth = 0;

    const NODE = 1;

    /** @inheritdoc */
    public function finalise()
    {
        $last_call = end($this->calls);
        $this->writeCall(array('list_close',array(),
$last_call[2]));

        $this->process();
        $this->callWriter->finalise();
        unset($this->callWriter);
    }

    /** @inheritdoc */
    public function process()
    {

        foreach ($this->calls as $call) {
            switch ($call[0]) {
                case 'list_item':
                    $this->listOpen($call);
                    break;
                case 'list_open':
                    $this->listStart($call);
                    break;
                case 'list_close':
                    $this->listEnd($call);
                    break;
                default:
                    $this->listContent($call);
                    break;
            }
        }

        $this->callWriter->writeCalls($this->listCalls);
        return $this->callWriter;
```

```
48.     }
49.
50.     protected function listStart($call)
51.     {
52.         $depth = $this->interpretSyntax($call[1][0], $listType);
53.
54.         $this->initialDepth = $depth;
55.         //                   array(list type, current depth, index
    of current listitem_open)
56.         $this->listStack[] = array($listType, $depth, 1);
57.
58.         $this->listCalls[] =
    array('list'.$listType.'_open',array(),$call[2]);
59.         $this->listCalls[] =
    array('listitem_open',array(1),$call[2]);
60.         $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
61.     }
62.
63.
64.     protected function listEnd($call)
65.     {
66.         $closeContent = true;
67.
68.         while ($list = array_pop($this->listStack)) {
69.             if ($closeContent) {
70.                 $this->listCalls[] =
    array('listcontent_close',array(),$call[2]);
71.                 $closeContent = false;
72.             }
73.             $this->listCalls[] =
    array('listitem_close',array(),$call[2]);
74.             $this->listCalls[] = array('list'.$list[0].'_close',
    array(), $call[2]);
75.         }
76.     }
77.
78.     protected function listOpen($call)
79.     {
80.         $depth = $this->interpretSyntax($call[1][0], $listType);
81.         $end = end($this->listStack);
82.         $key = key($this->listStack);
83.
84.         // Not allowed to be shallower than initialDepth
85.         if ($depth < $this->initialDepth) {
86.             $depth = $this->initialDepth;
87.         }
88.
89.         if ($depth == $end[1]) {
90.             // Just another item in the list...
```

```
91.             if ($listType == $end[0]) {
92.                 $this->listCalls[] =
    array('listcontent_close',array(),$call[2]);
93.                 $this->listCalls[] =
    array('listitem_close',array(),$call[2]);
94.                 $this->listCalls[] =
    array('listitem_open',array($depth-1),$call[2]);
95.                 $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
96.
97.                 // new list item, update list stack's index into
    current listitem_open
98.                 $this->listStack[$key][2] =
    count($this->listCalls) - 2;
99.
100.                // Switched list type...
101.            } else {
102.                $this->listCalls[] =
    array('listcontent_close',array(),$call[2]);
103.                $this->listCalls[] =
    array('listitem_close',array(),$call[2]);
104.                $this->listCalls[] =
    array('list'.$end[0].'_close', array(), $call[2]);
105.                $this->listCalls[] =
    array('list'.$listType.'_open', array(), $call[2]);
106.                $this->listCalls[] = array('listitem_open',
    array($depth-1), $call[2]);
107.                $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
108.
109.                array_pop($this->listStack);
110.                $this->listStack[] = array($listType, $depth,
    count($this->listCalls) - 2);
111.            }
112.        } elseif ($depth > $end[1]) { // Getting deeper...
113.            $this->listCalls[] =
    array('listcontent_close',array(),$call[2]);
114.            $this->listCalls[] = array('list'.$listType.'_open',
    array(), $call[2]);
115.            $this->listCalls[] = array('listitem_open',
    array($depth-1), $call[2]);
116.            $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
117.
118.            // set the node/leaf state of this item's parent
    listitem_open to NODE
119.            $this->listCalls[$this->listStack[$key][2]][1][1] =
    self::NODE;
120.
121.            $this->listStack[] = array($listType, $depth,
    count($this->listCalls) - 2);
```

```
122.         } else { // Getting shallower ( $depth < $end[1] )
123.             $this->listCalls[] =
    array('listcontent_close',array(),$call[2]);
124.             $this->listCalls[] =
    array('listitem_close',array(),$call[2]);
125.             $this->listCalls[] =
    array('list'.$end[0].'_close',array(),$call[2]);
126.
127.             // Throw away the end - done
128.             array_pop($this->listStack);
129.
130.             while (1) {
131.                 $end = end($this->listStack);
132.                 $key = key($this->listStack);
133.
134.                 if ($end[1] <= $depth) {
135.                     // Normalize depths
136.                     $depth = $end[1];
137.
138.                     $this->listCalls[] =
    array('listitem_close',array(),$call[2]);
139.
140.                     if ($end[0] == $listType) {
141.                         $this->listCalls[] =
    array('listitem_open',array($depth-1),$call[2]);
142.                         $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
143.
144.                         // new list item, update list stack's
    index into current listitem_open
145.                         $this->listStack[$key][2] =
    count($this->listCalls) - 2;
146.                     } else {
147.                         // Switching list type...
148.                         $this->listCalls[] =
    array('list'.$end[0].'_close', array(), $call[2]);
149.                         $this->listCalls[] =
    array('list'.$listType.'_open', array(), $call[2]);
150.                         $this->listCalls[] =
    array('listitem_open', array($depth-1), $call[2]);
151.                         $this->listCalls[] =
    array('listcontent_open',array(),$call[2]);
152.
153.                         array_pop($this->listStack);
154.                         $this->listStack[] = array($listType,
    $depth, count($this->listCalls) - 2);
155.                     }
156.
157.                     break;
158.
```

```
159.                     // Haven't dropped down far enough yet.... ( $end[1] > $depth )
160.                 } else {
161.                     $this->listCalls[] = array('listitem_close',array(),$call[2]);
162.                     $this->listCalls[] = array('list'.$end[0].'_close',array(),$call[2]);
163.
164.                     array_pop($this->listStack);
165.                 }
166.             }
167.         }
168.     }
169.
170.     protected function listContent($call)
171.     {
172.         $this->listCalls[] = $call;
173.     }
174.
175.     protected function interpretSyntax($match, & $type)
176.     {
177.         if (substr($match, -1) == '*') {
178.             $type = 'u';
179.         } else {
180.             $type = 'o';
181.         }
182.         // Is the +1 needed? It used to be count(explode(...))
183.         // but I don't think the number is seen outside this handler
184.         return substr_count(str_replace("\t", '  ', $match), '  ') + 1;
185.     }
186. }
```