



1.1 XML Parsing with SAX and DOM :: A code comparison – by Leif Eriksen

1.1.1 Introduction

There are two main paradigms for parsing and processing XML documents

- **SAX - the Simple API for XML**
- **DOM - the Document Object Model**

Early in the design process for code that will be processing XML, a decision will need to be made on the style of parser that will be used.

Each paradigm brings with it advantages and caveats.

1.1.2 SAX-based Parsing

SAX treats an XML document as a stream.

It generates events as each structural element of an XML document is encountered, for example

- **Start and end of elements**
- **CDATA and PCDATA**
- **Processing Instruction**
- **Comment**

Programs can subscribe to (or receive) these events

When a program subscribes to an event, it identifies the piece of its code that it wants to have handle that event. So a program may have one piece of code handle the 'start-of-element' event, another may handle the 'end-of-element' event. Events that are not subscribed to are typically dropped, though we may also establish a default handler.

Each event is independent, though the context of the event is available. This means you can distinguish between an event for the <Name> elements in <Ship><Name>Titanic</Name></Ship> and <Person><Name>DiCaprio</Name></Person>.

The order of the events is driven by the order of the elements in the XML document. You cannot direct the SAX-parser to generate events in an order different to that of the structure of the document. They handle the document as a stream, they start at one end and produce events till they get to the end.

If you need to produce output from the content of an XML document, and that output needs to be completely independent in structure from the XML document, this can be hard to achieve with a SAX processor

1.1.3 DOM-based Parsing

DOM-based parsers produce a new representation of an XML document, one that expresses the relationships between XML elements of the original document

Generally, this is an in-memory representation. For example, it may be a tree, or a hash, or some other data structure capable of expressing parent/child/sibling relationships.

Generally, the in-memory representation is a composition of different containers and object collections - a hash of arrays, a hash of hashes etc.

Programs use the DOM API, or the container/collection API's, to navigate, filter and transform the DOM representation.

If the document is 'never-ending' (say a stock exchange feed), this is not an appropriate choice

1.1.4 Perls Parsers

Perl has a lot of XML parsers available, some come with the Perl distribution, others are available from CPAN. Only two are dealt with in this talk.



1.1.4.1 XML::Parser - a SAX parser

XML::Parser is an extremely powerful parser, with a number of great features

- **it comes with Perl**
- **it is very fast/stable**
- **it has a simple API**

We will demonstrate XML::Parser using one of its 'styles'. When you create a new XML::Parser instance you declare the style you want (my \$p = XML::Parser->new(Style => 'XXX');)

The style we choose is 'Subs'. Each element of the XML document will result in a subroutine of the same name being called e.g. <House>For Sale</House> will generate two events

- **House - this event fires when the <House> element is detected.**
- **House_ - this event fires when the </House> element is detected.**

In both cases, XML::Parser will look for two subroutines, in your program, by the same name. you can direct XML::Parser to look in a particular namespace instead. We use this feature to place all our event-handling code in one module.

If you dont have a subroutine written of the appropriate name, the event is 'dropped' - no action takes place.

1.1.4.2 XML::Sax - a DOM parser

XML::Simple parses the complete XML document and produces an in-memory hash-of-hashes representation of the document.

XML::Simple has a rich API and feature set, so it may at first glance appear misnamed. The simplicity is in the extremely simple in-memory representations it can produce - they are Perl native data structures, and very easy to navigate.

Some things to note about XML::Simple are :-

- **it produces perl data structures - a plus**
- **it is extremely configurable - sometimes good, sometimes bad**
- **it is on CPAN - again this is good and bad**
- **it is slow to produce a representation. But you save hours/days of development time, because you can solve you XML problem quickly. But for large XML documents (>500K) it tkes several seconds to produce the internal representation, so high volumes of these will cause a problem. The internal representation is also linked to the size of the source document - large document, lots of memory used to represent it.**

An example of an in-memory representation of an XML document is :-

Our XML is :-

```
<config logdir="/var/log/foo/" debugfile="/tmp/foo.debug">
  <server name="sahara" osname="solaris" osversion="2.6">
    <address>10.0.0.101</address>
    <address>10.0.1.101</address>
  </server>
  <server name="gobi" osname="irix" osversion="6.5">
    <address>10.0.0.102</address>
  </server>
  <server name="kalahari" osname="linux" osversion="2.0.34">
    <address>10.0.0.103</address>
    <address>10.0.1.103</address>
  </server>
</config>
```

This can be parsed by XML::Simple to become an in-memory hash :-



```
{
  'logdir'      => '/var/log/foo/',
  'debugfile'  => '/tmp/foo.debug',
  'server'     => {
    'sahara'    => {
      'osversion' => '2.6',
      'osname'    => 'solaris',
      'address'   => [ '10.0.0.101', '10.0.1.101' ]
    },
    'gobi'      => {
      'osversion' => '6.5',
      'osname'    => 'irix',
      'address'   => '10.0.0.102'
    },
    'kalahari' => {
      'osversion' => '2.0.34',
      'osname'    => 'linux',
      'address'   => [ '10.0.0.103', '10.0.1.103' ]
    }
  }
}
```

This allows us to write code like :-

```
foreach my $server (sort keys %{$xml->{server}} ) {
  push @{$os{$xml->{server}->{$server}->{osname}}}, $server; # track which servers map
  to the same os
}
```

1.1.5 Code Comparison

What follows is two families of code to process the same input XML file, and produce the same output. One is based on a DOM parser, the other a SAX parser.

It is based on a real customers requirements, and the input data is a real client data file, obscured for security and privacy.

The DOM version is running in production - though it has changed slightly from the version presented here as new requirements have been added since this paper was prepared.

The important thing to note in this talk is the extremely different nature of the code presented.

1.1.6 Code

1.1.6.1 General Files

1.1.6.1.1 MANIFEST

```
ASIC490_085946461.xml
dom.pl
DOM/InvoiceStatement.pm
DOM/SchemeStatement.pm
dom_log.conf
Makefile.PL
MANIFEST          This list of files
sax.pl
SAX/InvoiceStatement.pm
SAX/SchemeStatement.pm
SAX/t/InvoiceStatement.t
SAX/t/SchemeStatement.t
sax_log.conf
Utils.pm
VERSION.pm
xsl.pl
```

1.1.6.1.2 Makefile.PL



```
use ExtUtils::MakeMaker;

require 5.006;

WriteMakefile(
    NAME      => 'XML',
    VERSION_FROM => 'VERSION.pm',
    test      => {TESTS => './SAX/t/*.t'},
    PREREQ_PM => {
        Log::Log4perl => 0.36,
    },
);
```

1.1.6.1.3 VERSION.pm

```
use vars qw($VERSION $RCS_ID);

$VERSION = sprintf("%d.%02d", q$Revision: 1.1 $ =~ /(\d+)\.(\d+)/);
$RCS_ID = q$Id: VERSION.pm,v 1.1 2004/03/15 03:09:02 le99007 Exp $;
$NAME = q$Name: HPA_PERLLIBS_1_21 $
```

1.1.6.1.4 Utils.pm

```
package Utils;

use strict;

use Exporter;

our @ISA = qw(Exporter);

our @EXPORT = ();
our @EXPORT_OK = qw(lt
                    cp);
our %EXPORT_TAGS = (
    std => \@EXPORT,
    test => \@EXPORT_OK
);

sub cp {
    my ($fh, $id, $collator, $postcode, $nsp, $gatemark, $page_counter,
        $impression_counter) = @_;

    # setup default values
    $id           ||= ' ';
    $collator     ||= ' ';
    $postcode     ||= '0';
    $nsp          ||= '0';
    $gatemark     ||= 'NNNNNN';
    $page_counter ||= 0;
    $impression_counter ||= 0;

    $fh->print(sprintf("<CP>%-20.20s%-6.6s%04.4d%-3.3d%-6.6s%06.6d%06.6d\n", $id,
        $collator, $postcode, $nsp, $gatemark, $page_counter, $impression_counter));
}

1;
```

1.1.6.2 DOM Processing Files

1.1.6.2.1 dom.pl



```
#!/usr/bin/perl -w

use strict;

use Data::Dumper;
use IO::File;
use Carp;
use Getopt::Long;
use File::Spec;

use XML::Simple;
use Log::Log4perl qw(:levels);

my $VERSION = sprintf("%d.%02d", q$Revision: 1.2 $ =~ /(\d+)\.(\d+)/);

GetOptions(
    'log=s'    => \my $log,
    'files=s'  => \my @files,
    'dirs=s'   => \my @dirs,
    'out=s'    => \my $out,
);

#canonicalise
@files = split(/,/, join(',', @files));
@dirs  = split(/,/, join(',', @dirs));

die "must specify at least one entry in either --files or --dirs on the command line" if
(! @files and ! @dirs);
die "must specify the path to the logging configuration file in --log" if (! $log);
die "must specify the path to the output directory in --out" if (! $out);

Log::Log4perl->init_and_watch($log, 10);
my $logger = Log::Log4perl->get_logger('DOM')
    or die "Cannot open logger\n";

# we get each xml file in the directories in @dirs, and add it to @files
foreach my $dir (@dirs) {
    opendir(DIR, $dir)
        or $logger->error("cannot open $dir :: $!") && next;
    foreach my $entry (readdir(DIR)) {
        $entry = $dir . "/" . $entry;
        next unless(-e $entry and
            -s $entry and
            -f $entry and
            -r $entry and
            $entry =~ /xml$/
        );
        push @files, $entry;
    }
}

$logger->info("Beginning DOM processing at ", scalar localtime);

foreach my $clfn (@files) {
    my $xs = XML::Simple->new();

    $logger->info("About to parse $clfn");
    my $xml;

    eval {
        $xml = $xs->XMLin($clfn,
            KeepRoot    => 1,
            ForceArray => [
```



```
        'n0:postalAddr',
        'n0:officer',
        'n0:officesheld',
        'n0:shareClass',
        'n0:member',
        'n0:owner',
        'n0:holding',
        'n0:statementLines',
        'n0:lineDescription',
        'postalAddr',
        'officers',
        'officesheld',
        'shareClass',
        'member',
        'holding',
        'interestClass',
        'holder',
        'owner',
        'statementLines',
        'lineDescription',
        'openingBalanceText',
        'accountBalance',
        'accountBalanceText',
    ]);

};

if ($@) { # most relevant detail is line 1 of the exception
    carp "cannot parse $clfn - $@";
    next;
}

STDERR->print(Dumper($xml));

my ($ns_part, $letter_type) = ((keys %$xml)[0] =~ m/^([:]*:)?(.*?)$/);

$ns_part = '' if ! defined $ns_part;

$logger->debug("ns_part == $ns_part, leter_type == $letter_type");
my $class = "DOM::" . $letter_type;

eval "require $class";

if ($@) {
    carp "cannot load $class for $clfn - document encoder possibly not implemented
\n$@";
    next;
}

use File::Basename;
my $ofn = basename($clfn, ".xml") . ".dom";
$ofn = File::Spec->join($out, $ofn);

my $ofh = IO::File->new($ofn, O_RDWR|O_CREAT|O_TRUNC)
    or die "cannot open $ofn :: $!";

$logger->debug("generate header record for $letter_type");

use POSIX qw(strftime);

$ofh->print('<HDR>DOM::', $letter_type, ' XML ', $clfn, ' ', strftime("%d\%m\%Y",
(localtime)), "\n");
$logger->debug("create LT data in $ofn");
{
    no strict 'refs';
```



```
&{$class . '::generate'}($ns_part, $xml, $ofh); # the base name is the id field of
the CP record
}

print $ofh "<TLR>" . sprintf("%08.8d%08.8d%08.8d\n", 1, 0, 0);

close($ofh);
}

# and we're done
exit 0;
```

1.1.6.2 dom_log.conf

```
# set your logger here
log4perl.logger=INFO, A1

#configure SCREEN logger

log4perl.appender.A1=Log::Log4perl::Appender::Screen
log4perl.appender.A1.min_level=debug
log4perl.appender.A1.stderr=1
log4perl.appender.A1.layout=Log::Log4perl::Layout::PatternLayout
log4perl.appender.A1.layout.ConversionPattern=%d %-5.5p - %M - %m%n
```

1.1.6.3 DOM/InvoiceStatement.pm

```
package DOM::InvoiceStatement;

use strict;

use Exporter;

our @ISA      = qw(Exporter);

our @EXPORT   = ();
our @EXPORT_OK = qw(generate);
our %EXPORT_TAGS = (std => \@EXPORT,
                    test => \@EXPORT_OK);

our $VERSION = sprintf("%d.%02d", q$Revision: 1.1 $ =~ /(\d+)\.(\d+)/);

use Log::Log4perl qw(get_logger);
use Data::Dumper;

sub generate {
    my ($ns, $xml, $fh) = @_;

    my $logger = get_logger((caller(0))[3]);

    {
        my $logger = get_logger((caller(0))[3] . '::DumpXML');

        $logger->debug(Dumper($xml)) if $logger->is_debug();
    }

    my $header = $xml->{$ns . 'statementHeader'};

    # Invoice Statement
```



```
_lt($fh, 4000);
_lt($fh, 4010, $header->{$sns . 'statementDate'});
_lt($fh, 4020, $header->{$sns . 'debtorType'});
my $formatted = $header->{$sns . 'debtorAccount'};
$formatted =~ s/(?<=\\d)(?=(?:\\d{3})+)$/ /g;
_lt($fh, 4030, $formatted);
_lt($fh, 4040, $header->{$sns . 'debtorLedger'});
_lt($fh, 4050, $header->{$sns . 'debtorLedger'} . ' ' . $header->{$sns .
'debtorAccount'});
_lt($fh, 4060, $header->{$sns . 'debtorName'});
my $base__lt = 4090;
foreach my $debtoraddress (@{$header->{$sns . 'debtorNameAndAddress'}}) {
    _lt($fh, $base__lt += 10, ref($debtoraddress) =~ m/HASH/ ? '' : $debtoraddress);
}
_lt($fh, 4170);

_lt($fh, 4180, $header->{$sns . 'openingBalance'});
_lt($fh, 4190, ref($header->{$sns . 'openingBalanceText'}->[0]) =~ m/HASH/ ? '' :
$header->{$sns . 'openingBalanceText'}->[0]);
_lt($fh, 4200, ref($header->{$sns . 'accountBalance' }->[0]) =~ m/HASH/ ? '' :
$header->{$sns . 'accountBalance'}->[0]);
_lt($fh, 4210, ref($header->{$sns . 'accountBalanceText'}->[0]) =~ m/HASH/ ? '' :
$header->{$sns . 'accountBalanceText'}->[0]);
_lt($fh, 4220, $header->{$sns . 'newCharges'});
_lt($fh, 4230, $header->{$sns . 'paymentsAndCredits'});
_lt($fh, 4240, $header->{$sns . 'amountPayable'});

if ($header->{$sns . 'amountPayableNow'} > 0) {
    _lt($fh, 4250, $header->{$sns . 'amountPayableNow'});
    _lt($fh, 4260);
}

if ($header->{$sns . 'amountPayableFuture'} != 0) {
    _lt($fh, 4270, $header->{$sns . 'amountPayableFuture'});
    _lt($fh, 4280);
}

if ($header->{$sns . 'amountPayableFuture'} != 0) {
    _lt($fh, 4290, $header->{$sns . 'futureDebtDueDate'});
    _lt($fh, 4300);
}

_lt($fh, 4310, $header->{$sns . 'bpayReference'});
_lt($fh, 4320, $header->{$sns . 'austpostReference'});
_lt($fh, 4320, $header->{$sns . 'asicReference'});
_lt($fh, 4330, $header->{$sns . 'statementReferenceNo'});
_lt($fh, 4340, $header->{$sns . 'reviewTransRef'});
_lt($fh, 4380);

my $statementLines;
$logger->debug("statement lines are ", Dumper($xml->{$sns . 'statementLines'}));
if (exists $xml->{$sns . 'statementLines'}[0]{'statementLine'}) {
    $statementLines = $xml->{$sns . 'statementLines'};
    _lt($fh, 4400);
    foreach my $statementline (@$statementLines) {
        _lt($fh, 4410);
        _lt($fh, 4420, $statementline->{$sns . 'statementLine'}{$sns . 'lineDate'});
        _lt($fh, 4430, $statementline->{$sns . 'statementLine'}{$sns . 'lineReference'});
        _lt($fh, 4440, $statementline->{$sns . 'statementLine'}{$sns . 'lineAmount'});
        foreach my $linedescription (@{$statementline->{$sns . 'statementLine'}{$sns .
'lineDescription'}}) {
            $logger->debug(Dumper($linedescription));
            _lt($fh, 4450);
            _lt($fh, 4460, ref($linedescription) =~ m/HASH/ ? '' : $linedescription);
            _lt($fh, 4470);
        }
    }
}
```



```
    _lt($fh, 4480);
}

    _lt($fh, 4500);
}

sub _lt {
    my ($fh, $lt, $data) = @_;

    print $fh "<LT$lt>";

    print $fh "$data" if $data;

    print $fh "\\n";
}

1;
```

1.1.6.3.1 DOM/SchemeStatement.pm

```
package DOM::SchemeStatement;

use strict;

use Exporter;

our @ISA      = qw(Exporter);

our @EXPORT   = ();
our @EXPORT_OK = qw(generate);
our %EXPORT_TAGS = (std => \@EXPORT,
                    test => \@EXPORT_OK);

our $VERSION = sprintf("%d.%02d", q$Revision: 1.1 $ =~ /(\d+)\.(\d+)/);

use Log::Log4perl qw(get_logger);
use Data::Dumper;

use Utils qw(cp);

use DOM::InvoiceStatement;

sub generate {
    my ($ns, $xml, $fh, $id) = @_;
    # note - currently Scheme does not use a namespace component, however that may change
    # if it does, no code change is required - :-)

    my $logger = get_logger((caller(0))[3]);

    {
        my $logger = get_logger((caller(0))[3] . '::DumpXML');

        $logger->debug(Dumper($xml)) if $logger->is_debug();
    }

    my $scheme = $xml->{$ns . 'SchemeStatement'};

    cp($fh, $id);
}
```



```
# Scheme Letter

_lt($fh, 2100);
_lt($fh, 2110, $scheme->{$ns . 'messageHeader'}{$ns . 'messageType'});
_lt($fh, 2120, $scheme->{$ns . 'messageHeader'}{$ns . 'messageVersion'});
_lt($fh, 2140, $scheme->{$ns . 'despatchDetails'}{$ns . 'addressee'});
if (exists $scheme->{$ns . 'despatchDetails'}{$ns . 'postalAddr'}) {
    my $base__lt = 2140;
    foreach my $line (@{$scheme->{$ns . 'despatchDetails'}{$ns . 'postalAddr'}}) {
        _lt($fh, $base__lt += 10, ref($line) =~ m/HASH/ ? ' ' : $line);
    }
}
my $sch = $scheme->{$ns . 'scheme'};
_lt($fh, 2220, $sch->{$ns . 'idType'});
_lt($fh, 2230, $sch->{$ns . 'idNumber'});
_lt($fh, 2240, $sch->{$ns . 'schemeName'});
_lt($fh, 2250, $sch->{$ns . 'reviewDate'});
_lt($fh, 2280);

# Scheme Statement

if (exists $scheme->{$ns . 'company'}) {
    my $company = $scheme->{$ns . 'company'};
    _lt($fh, 2300);
    _lt($fh, 2310, $company->{$ns . 'extractDate'});
    _lt($fh, 2220, $company->{$ns . 'idType'});
    _lt($fh, 2230, $company->{$ns . 'idNumber'});
    _lt($fh, 2320, $company->{$ns . 'reviewDate'});
    _lt($fh, 2330, $company->{$ns . 'corporateKey'});
    _lt($fh, 2350);
}

# Statement Details - Responsible Entity

if (exists $scheme->{$ns . 'respEntity'}) {
    my $entity = $scheme->{$ns . 'respEntity'};
    _lt($fh, 2400);
    _lt($fh, 2410, $entity->{$ns . 'reHeading'}{$ns . 'reRef'});
    _lt($fh, 2420, $entity->{$ns . 'reHeading'}{$ns . 'reLabel'});
    _lt($fh, 2430, $entity->{$ns . 'reIdType'});
    _lt($fh, 2440, $entity->{$ns . 'reIdentifier'});
    _lt($fh, 2450, $entity->{$ns . 'reName'});
    _lt($fh, 2460);
    if (exists $entity->{$ns . 'reMessage'}{$ns . 'reText'}) {
        _lt($fh, 2470);
        _lt($fh, 2480, $entity->{$ns . 'reMessage'}{$ns . 'reText'});
    }
    _lt($fh, 2490);
}

# Statement Details - Issued Interests Details

if (exists $scheme->{$ns . 'issuedInterests'}) {
    my $interests = $scheme->{$ns . 'issuedInterests'};
    _lt($fh, 2500);
    _lt($fh, 2510, $interests->{$ns . 'intHeading'}{$ns . 'intRef'});
    _lt($fh, 2520, $interests->{$ns . 'intHeading'}{$ns . 'intLabel'});

    if (exists $interests->{$ns . 'unitTrusts'}) {
        my $units = $interests->{$ns . 'unitTrusts'};
        _lt($fh, 2530);
        _lt($fh, 2540, $units->{$ns . 'unitHeading'}{$ns . 'unitRef'});
        _lt($fh, 2550, $units->{$ns . 'unitHeading'}{$ns . 'unitLabel'});
        _lt($fh, 2560);
        foreach my $interest (@{$units->{$ns . 'interestClass'}}) {
            _lt($fh, 2570);
        }
    }
}
```



```
        _lt($fh, 2580, $interest->{$ns . 'classCode'});
        _lt($fh, 2590, $interest->{$ns . 'title'});
        _lt($fh, 2600, $interest->{$ns . 'numberIssued'});
        _lt($fh, 2610, $interest->{$ns . 'totalAmountPaid'});
        _lt($fh, 2620, $interest->{$ns . 'totalAmountUnPaid'});
        _lt($fh, 2630, $interest->{$ns . 'numberOptions'});
        _lt($fh, 2640, $interest->{$ns . 'optionPrice'});
        _lt($fh, 2650);
    }
    _lt($fh, 2660);
}

if (exists $interests->{$ns . 'nonUnitTrusts'}) {
    my $nonunits = $interests->{$ns . 'nonUnitTrusts'};
    _lt($fh, 2730);
    _lt($fh, 2740, $nonunits->{$ns . 'nintHeading'}{$ns . 'nintRef'});
    _lt($fh, 2750, $nonunits->{$ns . 'nintHeading'}{$ns . 'nintLabel'});
    _lt($fh, 2760);
    foreach my $interest (@{$nonunits->{$ns . 'interestClass'}}) {
        _lt($fh, 2770);
        _data_lt($fh, 2780, $interest->{$ns . 'classCode'});
        _data_lt($fh, 2790, $interest->{$ns . 'title'});
        _data_lt($fh, 2800, $interest->{$ns . 'numberIssued'});
        _data_lt($fh, 2810, $interest->{$ns . 'totalAmountPaid'});
        _data_lt($fh, 2820, $interest->{$ns . 'totalAmountUnpaid'});
        _data_lt($fh, 2830, $interest->{$ns . 'numberOptions'});
        _data_lt($fh, 2840, $interest->{$ns . 'optionPrice'});
        _lt($fh, 2850);
    }
    _lt($fh, 2860);
}

if (exists $interests->{$ns . 'intMessage'}) {
    _lt($fh, 2900);
    _lt($fh, 2910, $interests->{$ns . 'intMessage'}{$ns . 'intText'});
    _lt($fh, 2920);
}

_lt($fh, 2950);
}

if (exists $scheme->{$ns . 'interestHolders'}) {

    _lt($fh, 3000);

    my $holders = $scheme->{$ns . 'interestHolders'};
    _lt($fh, 3010, $holders->{$ns . 'ihHeading'}{$ns . 'ihRef'});
    _lt($fh, 3020, $holders->{$ns . 'ihHeading'}{$ns . 'ihLabel'});
    if (exists $holders->{$ns . 'holder'}) {
        $logger->debug("holders is ", Dumper($holders->{$ns . 'holder'}));

        _lt($fh, 3040);

        # Scheme Statement - Interest Holder Details
        foreach my $holder (@{$holders->{$ns . 'holder'}}) {
            _lt($fh, 3050);
            foreach my $owner (@{$holder->{$ns . 'owner'}}) {
                _lt($fh, 3055);
                _lt($fh, 3060, $owner->{$ns . 'onrName'});
                _lt($fh, 3070, $owner->{$ns . 'onrACN'});
                _lt($fh, 3080, $owner->{$ns . 'onrAddress'});
                _lt($fh, 3085);
            }
            _lt($fh, 3087);
        }
    }
}
```



```
        foreach my $holding (@{$holder->{$ns . 'holding'}}) {
            _lt($fh, 3090, $holding->{$ns . 'hdgClass'});
            _lt($fh, 3091, $holding->{$ns . 'numberHeld'});
            _lt($fh, 3092, $holding->{$ns . 'fullyPaid'});
            _lt($fh, 3094);
        }
        _lt($fh, 3096);
    }
}
_lt($fh, 3100);
}

if (exists $scheme->{$ns . 'interestHolders'}{$ns . 'ihMessage'}) {
    _lt($fh, 3110);
    _lt($fh, 3120, $scheme->{$ns . 'interestHolders'}{$ns . 'ihMessage'}{$ns .
'ihText'});
}

if (exists $scheme->{$ns . 'qspStatement'}) {
    _lt($fh, 3150);
    $logger->debug("Passing ", Dumper($scheme->{$ns . 'qspStatement'})) if $logger-
>is_debug();
    DOM::InvoiceStatement::generate('', $scheme->{$ns . 'qspStatement'}, $fh);
} else {
    _lt($fh, 3155);
}
}

sub _lt {
    my ($fh, $lt, $data) = @_;

    print $fh "<LT$lt>";

    print $fh "$data" if $data;

    print $fh "\\n";
}

sub _data_lt {
    my ($fh, $lt, $data) = @_;

    _lt(@_) if ($data);
}

1;
```

1.1.6.4 SAX Processing Files

1.1.6.4.1 sax.pl

```
#!/usr/bin/perl -w

use strict;

use Data::Dumper;
use Getopt::Long;
use IO::File;
use Carp;
use File::Basename;
use File::Spec;
use POSIX qw(strftime);

use XML::Parser;
```



```
use Log::Log4perl qw(:levels);

use SAX::SchemeStatement;

my $VERSION = sprintf("%d.%02d", q$Revision: 1.1 $ =~ /(\d+)\.(\d+)/);

GetOptions(
    'log=s'    => \my $log,
    'files=s' => \my @files,
    'dirs=s'  => \my @dirs,
    'out=s'   => \my $out,
);

#canonicalise
@files = split(/,/, join(',', @files));
@dirs  = split(/,/, join(',', @dirs));

die "must specify at least one entry in either --files or --dirs on the command line" if
(! @files and ! @dirs);
die "must specify the path to the logging configuration file in --log" if (! $log);
die "must specify the path to the output directory in --out" if (! $out);

Log::Log4perl->init_and_watch($log, 10);
my $logger = Log::Log4perl->get_logger('SAX')
    or die "Cannot open logger $!\n";

$logger->info("Beginning SAX processing at ", scalar localtime);

# we get each xml file in the directories in @dirs, and add it to @files
foreach my $dir (@dirs) {
    opendir(DIR, $dir)
        or $logger->error("cannot open $dir :: $!") && next;
    foreach my $entry (readdir(DIR)) {
        $entry = $dir . "/" . $entry;
        next unless(-e $entry and
            -s $entry and
            -f $entry and
            -r $entry and
            $entry =~ /xml$/
        );
        push @files, $entry;
    }
}

my $p = XML::Parser->new(Pkg          => 'SAX::SchemeStatement',
                        Style        => 'Subs',
                        ErrorContext => 1,
                        Handlers     => {
                            Init => \&SAX::SchemeStatement::Init,
                            Final => \&SAX::SchemeStatement::Final,
                        },
                        )
    or die "cannot create parser :: $!";

foreach my $clfn (@files) {
    $logger->info("About to parse $clfn");

    my $ofn = basename($clfn, ".xml") . ".sax";
    $ofn = File::Spec->join($out, $ofn);

    my $ofh = IO::File->new($ofn, O_RDWR|O_CREAT|O_TRUNC)
        or die "cannot open $ofn :: $!";
```



```
SAX::SchemeStatement->set_output($ofh);
eval {
    $p->parsefile($clfn);
};

$logger->logdie($@) if $@;

SAX::SchemeStatement->reset_holder();

$ofh->print("<TLR>" . sprintf("%08.8d%08.8d%08.8d\n", scalar(@files), 0, 0));

close($ofh);
}

$logger->info("Processed @{{[scalar @files]} files in @{{[time() - $^T]} seconds\n");
# and we're done
exit 0;
```

1.1.6.4.2sax_log.conf

```
# set your logger here
log4perl.logger=INFO, A1

#configure SCREEN logger

log4perl.appender.A1=Log::Log4perl::Appender::Screen
log4perl.appender.A1.min_level=debug
log4perl.appender.A1.stderr=1
log4perl.appender.A1.layout=Log::Log4perl::Layout::PatternLayout
log4perl.appender.A1.layout.ConversionPattern=%d %-5.5p - %M - %m%n
```

1.1.6.4.3SAX/InvoiceStatement.pm

```
package SAX::InvoiceStatement;

use strict;

use Data::Dumper;
use Carp;

use Log::Log4perl qw(get_logger);

our $VERSION = sprintf("%d.%02d", q$Revision: 1.3 $ =~ /(\d+)\.(\d+)/);

our $ofh;

BEGIN {
    $ofh = \*STDOUT;
}

my %lt = (
    statementDate      => 4010,
    debtorType         => 4020,
    debtorAccount      => 4030,
    debtorLedger       => 4040,
    debtorName         => 4060,
    debtorNameAndAddress => 4100,
    openingBalance     => 4180,
    openingBalanceText => 4190,
    accountBalance     => 4200,
```



```
        accountBalanceText => 4210,
        newCharges         => 4220,
        paymentsAndCredits => 4230,
        amountPayable      => 4240,
        amountPayableNow   => 4250,
        amountPayableFuture => 4270,
        futureDebtDueDate  => 4290,
        bpayReference      => 4310,
        austpostReference  => 4320,
        asicReference      => 4320,
        statementReferenceNo => 4330,
        reviewTransRef     => 4340,
        statementHeader_   => 4380,
        statementLines     => 4400,
        statementLine      => 4410,
        statementLine_    => 4480,
        lineDate           => 4420,
        lineReference      => 4430,
        lineAmount         => 4440,
        lineDescription    => 4450,
        lineDescription_   => 4470,
    );

our $AUTOLOAD;

sub AUTOLOAD {
    my $p = shift;

    my $logger = get_logger((caller(0))[3]);

    (my $elem = $AUTOLOAD) =~ s/.*::(\w+)/$1/;

    if ($elem =~ m/_$/ ) {
        $ofh->print(_format_lt($lt{$elem}), "\\n") if (exists $lt{$elem});
    } else {
        _lt($p, $lt{$elem}) if (exists $lt{$elem});
    }
}

sub _lt {
    my ($p, $code) = @_ ;

    my $logger = get_logger((caller(0))[3]);

    confess() if !defined $ofh or ! defined $code;

    # set the handler for any subsequent data
    $ofh->print(_format_lt($code));

    $p->setHandlers(Char => \&_text);
}

sub _format_lt {
    my ($code) = @_ ;

    my $logger = get_logger((caller(0))[3]);

    return "<LT$code>";
}

sub _text {
    my ($p, $string) = @ ;
```



```
my $logger = get_logger((caller(0))[3]);

chomp $string;

$string =~ s/\s+//g;
$string =~ s/^\s//g;
$string =~ s/\s$//g;

$string .= "\\n";

$ofh->print($string);

$p->setHandlers(Char => undef);
}

my $post_idx = 0;
sub debtorNameAndAddress {
    my ($p, $elem) = @_;

    my $logger = get_logger((caller(0))[3]);

    _lt($p, $lt{$elem} + $post_idx);
    $post_idx += 10;
}

sub openingBalance {
    my ($p, $elem) = @_;

    my $logger = get_logger((caller(0))[3]);

    $ofh->print(_format_lt(4170));
    $ofh->print("\\n");

    _lt($p, $lt{$elem});
}

# Account Number
# this is a little complicated - we need to assemble an entry from
# two parts of the LT tree - not something SAX is good at
my $lt_string = '';

my $debtorLedger_seen = 0;
my $debtorAccount_seen = 0;

sub debtorLedger {
    my $p = shift;

    my $logger = get_logger((caller(0))[3]);

    $debtorLedger_seen++;
    # set the handler for any subsequent data
    $p->setHandlers(Char => \&_debtorLedger_text);
}

sub debtorAccount {
    my $p = shift;

    my $logger = get_logger((caller(0))[3]);

    $debtorAccount_seen++;
}
```



```
# set the handler for any subsequent data
$p->setHandlers(Char => \&_debtorAccount_text);
}

sub _debtorLedger_text {
    my ($p, $string) = @_;

    my $logger = get_logger((caller(0))[3]);

    $ofh->print(_format_lt(4040));
    $ofh->print($string . "\\n");

    $lt_string = $string . " $lt_string";

    $lt_string =~ s/\s+/ /g;
    $lt_string =~ s/^\s//g;
    $lt_string =~ s/\s$//g;

    _output_debtor_string();
    $p->setHandlers(Char => undef);
}

sub _debtorAccount_text {
    my ($p, $string) = @_;

    my $logger = get_logger((caller(0))[3]);

    # insert space every third digit...
    (my $format_string = $string) =~ s/(?<=\\d)(?=(?:\\d{3})+)$/ /g;

    $ofh->print(_format_lt(4030));
    $ofh->print($format_string . "\\n");

    $lt_string = "$lt_string $string";

    $lt_string =~ s/\s+/ /g;
    $lt_string =~ s/^\s//g;
    $lt_string =~ s/\s$//g;

    _output_debtor_string();
    $p->setHandlers(Char => undef);
}

sub _output_debtor_string {
    if ($debtorLedger_seen and $debtorAccount_seen) {
        $ofh->print(_format_lt(4050));
        $ofh->print($lt_string . "\\n");
        $lt_string = '';
        $debtorLedger_seen = 0;
        $debtorAccount_seen = 0;
    }
}
# End Account

sub amountPayableNow {
    my ($p, $elem) = @_;

    my $logger = get_logger((caller(0))[3]);

    $p->setHandlers(Char => \&_amountPayableNow_text);
}
```



```
sub _amountPayableNow_text {
  my ($p, $string) = @_;

  my $logger = get_logger((caller(0))[3]);

  if ($string > 0) {
    $ofh->print(_format_lt(4250));
    $ofh->print("$string\\n");
    $ofh->print(_format_lt(4260));
    $ofh->print("\\n");
  }
  $p->setHandlers(Char => undef);
}

my $amountPayableFuture = 0;
sub amountPayableFuture {
  my ($p, $elem) = @_;

  my $logger = get_logger((caller(0))[3]);

  $p->setHandlers(Char => &_amp;_amountPayableFuture_text);
}

sub _amountPayableFuture_text {
  my ($p, $string) = @_;

  my $logger = get_logger((caller(0))[3]);

  if ($string != 0) {
    $ofh->print(_format_lt(4270));
    $ofh->print("$string\\n");
    $ofh->print(_format_lt(4280));
    $ofh->print("\\n");
    $amountPayableFuture = $string;
  }
  $p->setHandlers(Char => undef);
}

sub futureDebtDueDate {
  my ($p, $elem) = @_;

  my $logger = get_logger((caller(0))[3]);

  $p->setHandlers(Char => &_amp;_futureDebtDueDate_text);
}

sub _futureDebtDueDate_text {
  my ($p, $string) = @_;

  my $logger = get_logger((caller(0))[3]);

  if ($amountPayableFuture != 0) {
    $ofh->print(_format_lt(4290));
    $ofh->print("$string\\n");
    $ofh->print(_format_lt(4300));
    $ofh->print("\\n");
    $amountPayableFuture = 0;
  }
  $p->setHandlers(Char => undef);
}

my $lineDescription text = '';
```



```
sub lineDescription {
  my ($p, $elem) = @_;

  my $logger = get_logger((caller(0))[3]);

  $ofh->print(_format_lt($lt{$elem}));
  $ofh->print("\\\\n");

  $p->setHandlers(Char => \&_lineDescription_text);
  $lineDescription_text = '';
}

sub _lineDescription_text {
  my ($p, $string) = @_;

  my $logger = get_logger((caller(0))[3]);

  $lineDescription_text = "$string";

  $p->setHandlers(Char => undef);
}

sub lineDescription_ {
  my ($p, $elem) = @_;

  my $logger = get_logger((caller(0))[3]);

  $ofh->print(_format_lt(4460));
  $ofh->print("$lineDescription_text\\\\n");

  $lineDescription_text = '';

  $ofh->print(_format_lt($lt{$elem . '_'}));
  $ofh->print("\\\\n");

  $p->setHandlers(Char => undef);
}

# qspStatement_ is handled by the SchemeStatement package.
# adjust the parser and have it handle the LT encoding

sub qspStatement_ {
  my ($p, @args) = @_;

  $ofh->print(_format_lt(4500));
  $ofh->print("\\\\n");

  # WARNING #####
  # we are reaching into the internals of the XML::Parser package to set
  # the Pkg attribute

  $p->{Pkg} = 'SAX::SchemeStatement'; # oh for a better API...
  SAX::SchemeStatement->set_output = $ofh;
}

sub set_output {
  my ($class, $fh) = @_;

  my $logger = get_logger((caller(0))[3]);
```



```
$logger->debug("setting output file handle");

$ofh = $fh;
}

1;
```

1.1.6.4.4SAX/SchemeStatement.pm

```
package SAX::SchemeStatement;

use strict;

use Data::Dumper;
use Carp;
use POSIX qw(strftime);

use Log::Log4perl qw(get_logger);

use SAX::InvoiceStatement;

use Utils qw(cp);

our $VERSION = sprintf("%d.%02d", q$Revision: 1.1 $ =~ /(\d+)\.(\d+)/);

my $ofh;

BEGIN {
    $ofh = \*STDOUT;
}

our %lt = (
    SchemeStatement => 2100,
    messageType => 2110,
    messageVersion => 2120,
    addressee => 2140,
    postalAddr => 2150,
    idType => 2220,
    idNumber => 2230,
    schemeName => 2240,
    scheme_ => 2280,
    idType_ => 2220,
    idNumber => 2230,
    company => 2300,
    company_ => 2350,
    respEntity => 2400,
    reRef => 2410,
    reLabel => 2420,
    reIdType => 2430,
    reIdentifier => 2440,
    reName => 2450,
    reName_ => 2460,
    reMessage => 2470,
    reText => 2480,
    respEntity_ => 2490,
    issuedInterests => 2500,
    intRef => 2510,
    intLabel => 2520,
    uintTrusts => 2530,
    uintRef => 2540,
    uintLabel => 2550,
    uintLabel_ => 2560,
    uintTrusts => 2660,
```



```
        nonUnitTrusts    => 2730,
        nintRef          => 2740,
        nintLabel       => 2750,
        nintLabel_      => 2760,
        nonUnitTrusts_  => 2860,
        intMessage      => 2900,
        intText         => 2910,
        intMessage_     => 2920,
        issuedInterests_ => 2950,
        interestHolders => 3000,
        ihRef           => 3010,
        ihLabel         => 3020,
        holder          => 3040,
        owner           => 3055,
        onrName         => 3060,
        onrACN          => 3070,
        onrAddress      => 3080,
        owner_          => 3085,
        holding         => 3087,
        hdgClass        => 3090,
        numberHeld      => 3091,
        fullyPaid       => 3092,
        holding_        => 3094,
        holder_         => 3096,
        interestHolders_ => 3100,
        ihMessage       => 3110,
        ihText          => 3120,
    );

our $AUTOLOAD;

sub AUTOLOAD {
    my $p = shift;

    my $logger = get_logger((caller(0))[3]);

    (my $elem = $AUTOLOAD) =~ s/.*::(\w+)/$/$1/;

    _lt($p, $lt{$elem}) if (exists $lt{$elem});
}

sub Init {
    my $p = shift;

    my $logger = get_logger((caller(0))[3]);

    $lt{holder} = 3040;

    $ofh->print('<HDR>', __PACKAGE__, ' XML ', $p->{Base}, ' ', strftime("%d\%m\%Y",
(localtime)), "\n");
}

sub Final {
}

sub _lt {
    my ($p, $code) = @_;

    confess() if !defined $ofh or ! defined $code;

    # set the handler for any subsequent data
    $ofh->print( format lt($code));
}
```



```
    $p->setHandlers(Char => \&_text);
}

sub _format_lt {
    my ($code) = @_;

    my $logger = get_logger((caller(0))[3]);

    $logger->debug((caller())) if (!defined $code);

    return "<LT$code>";
}

sub _text {
    my ($p, $string) = @_;

    chomp $string;

    $string =~ s/\s+//g;
    $string =~ s/^\s//g;
    $string =~ s/\s$/g;

    $string .= "\\n";

    $ofh->print($string);

    $p->setHandlers(Char => undef);
}

# Special event handlers from hereonin

# Scheme statement means we need a CP and 2100 LT

sub SchemeStatement {
    my ($p, $elem) = @_;

    cp($ofh);

    _lt($p, $lt{$elem});
}

my $qsp_seen = 0;

sub SchemeStatement_ {
    my ($p, $elem) = @_;

    if (!$qsp_seen) {
        $ofh->print(_format_lt(3155));
        $ofh->print("\\n");
    }
}

# postal address occurs 1-8 times, increasing the required LT each time
# one day we could try a self-initialising closure

my $post_idx = 0;
sub postalAddr {
    my ($p, $elem) = @_;

    lt($p, $lt{$elem} + $post_idx);
}
```



```
    $post_idx += 10;
}

# a closing despatchDetails indicates that we need to reset the post_idx, for the next
document
sub despatchDetails_ {
    $post_idx = 0;
}

# need to cater for a set of scenarios

# 1. empty holder - 3040, 3050, 3087 and 3096 output
# 2. multiple holders with owners and holdings

my $holding_seen = 0;

sub holder {
    my ($p, $elem) = @_;

    if ($lt{$elem} == 3040) {
        $ofh->print(_format_lt($lt{$elem}));
        $ofh->print("\\\\n");
        $lt{$elem} = 3050;
    }
    $ofh->print(_format_lt($lt{$elem}));
    $ofh->print("\\\\n");
}

sub holder_ {
    my ($p, $elem) = @_;

    if (! $holding_seen) {
        $ofh->print(_format_lt(3087));
        $ofh->print("\\\\n");
    }

    $ofh->print(_format_lt($lt{$elem} . '_'));
    $ofh->print("\\\\n");
    $holding_seen = 0;
}

sub holding {
    my ($p, $elem) = @_;

    if (! $holding_seen) {
        $ofh->print(_format_lt($lt{$elem}));
        $ofh->print("\\\\n");
        $holding_seen = 1;
    }
}

# some conflicts in child names for scheme and company elements - set appropriate keys
sub scheme {
    my $p = shift;

    $lt{reviewDate} = 2250;
    delete $lt{corporateKey};
    delete $lt{extractDate};
}

sub company {
    my $p = shift;
```



```
$ofh->print(_format_lt(2300));
$ofh->print("\\\\n");

$lt{extractDate} = 2310;
$lt{reviewDate} = 2320;
$lt{corporateKey} = 2330;

}

# conflicts for Unit and nonUnit trusts - set appropriate keys
sub interestClass {
  my $p = shift;

  my $type = $p->{Context}[-1];

  if ($type eq 'uintTrusts') {
    _lt($p, 2570);
    $lt{classCode} = 2580;
    $lt{title} = 2590;
    $lt{numberIssued} = 2600;
    $lt{totalAmountPaid} = 2610;
    $lt{totalAmountUnpaid} = 2620;
    $lt{numberOptions} = 2630;
    $lt{optionPrice} = 2640;
    $lt{interestClass_} = 2650;
  } elsif ($type eq 'nonUnitTrusts') {
    _lt($p, 2770);
    $lt{classCode} = 2780;
    $lt{title} = 2790;
    $lt{numberIssued} = 2800;
    $lt{totalAmountPaid} = 2810;
    $lt{totalAmountUnpaid} = 2820;
    $lt{numberOptions} = 2830;
    $lt{optionPrice} = 2840;
    $lt{interestClass_} = 2850;
  }
}

# qspStatement is handled by the InvoiceStatement package.
# adjust the parser and have it handle the LT encoding
sub qspStatement {
  my ($p, @args) = @_;

  my $logger = get_logger((caller(0))[3]);

  $ofh->print(_format_lt(3150));
  $ofh->print("\\\\n");

  $qsp_seen++;

  $ofh->print(_format_lt(4000));
  $ofh->print("\\\\n");

  $p->{Pkg} = 'SAX::InvoiceStatement';

  SAX::InvoiceStatement->set_output($ofh);
}

# class methods

sub set_output {
  my ($class, $fh) = @_;
```



```
    $ofh = $fh;
}

# I haven't worked out how to reset the holder LT between documents - hence this
#####
# HACK #
#####
sub reset_holder {
    $!t{holder} = 3040;
}

1;
```