



SAX and Perl

Kip Hampton
The Perl/XML Community Community



SAX



SAX

- Simple **A**PI for **X**ML



SAX

- **S**imple **A**PI for **X**ML
- Event-based, push-style processing model.

SAX

- Simple API for XML
- Event-based, push-style processing model.
- Document content is made available via the arguments passed to the methods in one or more event handler classes.

```
my $handler => My::Handler->new();
my $parser =
    XML::SAX::ParserFactory->parser( Handler => $handler, );

$parser->parse_uri('some.xml');

# events encountered while processing 'some.xml'
# are sent to the methods in My::Handler
```



Event Overview





Event Overview

- 3 common types of event classes:

Event Overview

- 3 common types of event classes:
- **Generators** (sometimes called **Drivers**)
 - ◆ Responsible for initializing the events in the stream.

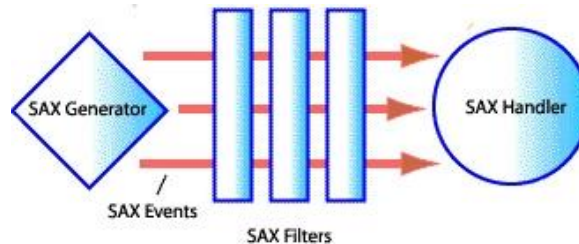


Event Overview

- 3 common types of event classes:
- **Generators** (sometimes called **Drivers**)
 - ◆ Responsible for initializing the events in the stream.
- **Handlers**
 - ◆ Receives the events sent from the Generator, provides access to the document content.

Event Overview

- 3 common types of event classes:
- **Generators** (sometimes called **Drivers**)
 - ◆ Responsible for initializing the events in the stream.
- **Handlers**
 - ◆ Receives the events sent from the Generator, provides access to the document content.
- **Filters**
 - ◆ A special class of Handler that both receives events from a Generator (or other Filter) and sends them to another Handler.



Event Overview (Continued)

```
<?xml version="1.0"?>
<doc>
  <quote>
    There are more things in heaven
    and earth, Horatio, Than are dreamt
    of in your philosophy.
  </quote>
</doc>
```

Event Overview (Continued)

```
start_document-->
    <?xml version="1.0"?>
start_element--->
    <doc>
start_element--->
    <quote>
    --->There are more things in heaven
characters-|   and earth, Horatio, Than are dreamt
    --->of in your philosophy.
    </quote>
end_element----->
    </doc>
end_element----->
end_document----->
```

Event Overview (Continued)

```
start_document-->
    <?xml version="1.0"?>
start_element--->
    <doc>
start_element--->
    <quote>
    --->There are more things in heaven
characters-|   and earth, Horatio, Than are dreamt
    --->of in your philosophy.
    </quote>
end_element----->
    </doc>
end_element----->
end_document----->
```

- Event methods are fired in the order encountered by the parser.

Event Overview (Continued)

```
start_document-->
    <?xml version="1.0"?>
start_element--->
    <doc>
start_element--->
    <quote>
    --->There are more things in heaven
characters-|   and earth, Horatio, Than are dreamt
    --->of in your philosophy.
    </quote>
end_element----->
    </doc>
end_element----->
end_document----->
```

- Event methods are fired in the order encountered by the parser.
- Data is passed through in predictable Perl data structures.

Event Overview (Continued)

```
start_document-->
                <?xml version="1.0"?>
start_element--->
                <doc>
start_element--->
                <quote>
                --->There are more things in heaven
characters-|    and earth, Horatio, Than are dreamt
                --->of in your philosophy.
                </quote>
end_element----->
                </doc>
end_element----->
end_document----->
```

- Event methods are fired in the order encountered by the parser.
- Data is passed through in predictable Perl data structures.
- Parser returns what the final Handler's end_document() method returns.



Event Handler Methods





Event Handler Methods

- There are a total of 33 handler methods that may be called during document processing.



Event Handler Methods

- There are a total of 33 handler methods that may be called during document processing.
- Don't worry: for most applications only a handful of those will actually need to be implemented:
 - ◆ `start_document()`
 - ◆ `end_document()`
 - ◆ `start_element()`
 - ◆ `end_element()`
 - ◆ `characters()`



Event Handler Method Details



Event Handler Method Details

- Document data is passed through (via @_) to the various methods as a single HASH reference of named, method-specific properties.

```
# the start_document event receives an  
# empty HASH reference by default:
```

```
sub start_document {  
    my $self = shift;  
    my $wtf = shift;  
    ...  
}
```

```
$wtf = {};
```

```
# the characters() event receives a reference with a single key
```

```
sub characters {  
    my $self = shift;  
    my $chars = shift;  
  
    my $text = $chars->{Data};  
    ...  
}
```

Event Handler Method Details (Cont'd)

```
# example start_element $element structure
# for <foo:bar/>
```

```
$element = { Name           => 'foo:bar',
              LocalName     => 'bar',
              Prefix        => 'foo',
              NamespaceURI   => 'http://some.tld/nsfoo',
              Attributes     => {},
            }
```

```
# end_element is the same, minus the Attributes property
```

```
$element = { Name           => 'foo:bar',
              LocalName     => 'bar',
              Prefix        => 'foo',
              NamespaceURI   => 'http://some.tld/nsfoo',
            }
```



Handlers

Handlers

- Handlers are the most commonly implemented SAX class.



Handlers

- Handlers are the most commonly implemented SAX class.
- A Handler is the end-point in the event stream where you actually do something with the data.



Handlers

- Handlers are the most commonly implemented SAX class.
- A Handler is the end-point in the event stream where you actually do something with the data.
- The parser will return to the application whatever you return from your Handler's `end_document()` method.

Handler Example (XML)

```
<?xml version="1.0"?>
<messages>
  <message>
    <from>claudius@elsinore.gov</from>
    <to>maddog@elsinore.gov</to>
    <subject>
      Re: [RSVP] Impromptu Theatrical Performance
      Today!
    </subject>
    <body>
      Hamlet,
      The Queen and I sincerly look forward to
      attending your play.Glad to see that
      you're feeling better.
      Your Uncle and King,
      Claudius
    </body>
  </message>
  ...
</mssages>
```

Handler Example (Perl Class)

```
package My::MailHandler;
use strict;
use XML::SAX::Base;
use Mail::Sendmail;
use vars qw( @ISA );
@ISA = qw( XML::SAX::Base );
my (%mail_args, $current_element, $message_count, $sent_count);

sub start_element {
    my ($self, $element) = @_;

    if ($element->{Name} eq 'message') {
        %mail_args = ();
        $message_count++;
    }
    elsif ($element->{Name} eq 'body') {
        $current_element = 'message';
    }
    else {
        $current_element = $element->{Name};
    }
}
```

Handler Class Example (Cont'd)

```
sub characters {
  my ($self, $characters) = @_;
  my $text = $characters->{Data};
  unless ($current_element eq 'message') {
    $text =~ s/^\s*//;
    $text =~ s/\s*$//;
  }
  $mail_args{$current_element} .= $text if $text;
}

sub end_element {
  my ($self, $element) = @_;
  if ($element->{Name} eq 'message') {
    Mail::Sendmail::sendmail(%mail_args)
      or warn "Mail Error: $Mail::Sendmail::error";

    $sent_count++ unless $Mail::Sendmail::error;
  }
}
```



Filters



Filters

- A Filter is nothing more than a Handler that sends events to another Handler.

Filters

- A Filter is nothing more than a Handler that sends events to another Handler.
- Elements can be "pruned" from the stream by not forwarding certain events, or added by simply calling the correct methods.

Filters

- A Filter is nothing more than a Handler that sends events to another Handler.
- Elements can be "pruned" from the stream by not forwarding certain events, or added by simply calling the correct methods.
- Making your Filter a subclass of XML::SAX::Base makes life much easier.

Filter Example (Filter Class)

```
# silly text transformer
package PorcusFilter;
use strict;
use XML::SAX::Base;
use vars qw( @ISA );
@ISA = qw( XML::SAX::Base );

sub characters {
    my ($self, $chars) = @_ ;
    my $out = $self->porcus($chars->{Data});
    $self->SUPER::characters( {Data => $out} );
}

sub porcus {
    my ($self, $chars) = @_ ;
    $chars =~ tr/A-Z/a-z/;
    $chars =~ s/\b([aeiou])/w$1/g;
    my $cons = q{[bcfghjklmnpqrstvwxyz]};
    $chars =~
        s/\b(qu|$cons($cons$cons?)?|[a-z])([a-z]*)/$3$1ay/g;
    return $chars;
}

1;
```

Filter Example (Usage)

```
use strict;  
use XML::SAX::ParserFactory;  
use XML::SAX::Writer;  
use My::PorcusFilter;  
  
my $writer = XML::SAX::Writer->new();  
  
my $filter =  
    My::PorcusFilter->new( Handler => $writer );  
  
my $parser =  
    XML::SAX::ParserFactory->new( Handler => $filter );  
  
$parser->parse_uri('some.xml');
```

Filter Example (Result)

```
<html>
<body>
<p>
  otay emay, oneway ofway ethay ostmay
  agonizingway aspectsway ofway anguage
  lay esignday isway omingcay upway
  ithway away usefulway ystemsay ofway
  operatorsway. otay otherway
  anguelay esignersday, isthay aymay
  eemsay ikelay away illysay ingthay
  otay agonizeway overway. afterway
  allway, ouyay ancay iewway allway
  operatorsway asway eremay yntacticsay
  ugarsay -- operatorsway
  areway ustjay unnyfay ookinglay
  unctionfay allscay.
</p>
</body>
</html>
```



Filter Pipelines and XML::SAX::Machines



Filter Pipelines and XML::SAX::Machines

- XML::SAX::Machines obviates most of the pain of building complex SAX filter chains:

```
use XML::SAX::Machines qw( :all );
```

```
my $machine = Pipeline(  
    "My::SAXFilter::One",  
    "My::SAXFilter::Two",  
    "My::SAXFilter::Three",  
    \*STDOUT  
);
```

```
$machine->parse_uri( $xml_file );
```



Links and Resources

- Kip's Perl/XML Column - <http://xml.com/pub/q/perlxml>
- Comments? - khampton@totalcinema.com