SLAX Functions

node-set **slax:break-lines***(node-set)*
Break a string (or a node set containing a string) into a set of elements, one per line of text.

boolean **slax:dampen***(name, max, time-period)*
Return **true** if **dampen()** has been called with a given name more than a given number of within a given number of minutes.

object **slax:first-of***(object+)*
Return the first argument in the argument list that is not empty or a zero length string.

string **slax:get-command***(prompt)*
Prompt the user for an input string, with the value recorded in **readline**'s history.

string **slax:get-input***(prompt)*
Prompt the user for an input string.

string **slax:get-secret***(prompt)*
Prompt the user for an input string but do not echo their response. Suitable for passwords.

boolean **slax:is-empty***(object)*
Return **true** if the argument is empty.

string **slax:printf***(format, string*)*
Use the format string to encode the arguments. Standard printf(3) options apply as to **%j** modifiers:

"**%jcs**"	Capitalize first letter
"**%jt{TAG}s**"	Prepend TAG if string is not empty
"**%j1s**"	Skip field if value has not changed

node-set **slax:regex***(pattern, string)*
Match a regular expression against a string. The first member of the returned node set is the full string match, and additional members of the node set are parentheses matches.

void **slax:sleep***(seconds, milliseconds)*
Sleep for a given time period.

node-set **slax:split***(pattern, string, limit)*
Break a string into a set of elements, up to the limit times, at the pattern.

string **slax:sysctl***(name, format)*
Retrieve a **sysctl** variable. Format is **"i"** or **"s"**.

void **slax:syslog***(priority, string+)*
Syslog the concatenation of set of arguments.

The slaxproc Command

slaxproc is a command line tool that allows access to the SLAX language. It can run scripts, control input and outfile files, convert between XSLT and SLAX formats, perform syntax checks, and even perform "partial" conversions.

slaxproc [options] [script] [input] [output]

Modes:

--run, **–r**
Runs a SLAX script (the default mode)

--slax-to-xslt, **–x**
Converts SLAX scripts into XSLT

--xslt-to-slax, **–s**
Converts XSLT scripts into SLAX

--check, **–c**
Checks syntax and content of script

File options:

--name *filename*, **-n** *filename*
Gives the script file name

--input *filename*, **-I** *filename*
Gives the input file name

--output *filename*, **-o** *filename*
Gives the output file name

Common options:

--debug, **-d**
Enables the SLAX/XSLT debugger

--exslt, **–e**
Enables the EXSLT library

--indent, **–g**
Indents script output

--param *name value*, **-a** *name value*
Passes parameters into the script

--partial, **–p**
Allows partial input (for **–x** or **–s**)

--trace *filename*, **-t** *filename*
Writes trace data to a file

--version, **–V**
Shows version information (and exits)

Examples:

```
% slaxproc -g example.slax in.xml out.xml
% slaxproc --trace /tmp/foo test1.slax in.xml
% slaxproc --debug -i in.xml -n my.slax
% slaxproc -c < in.xml > data.slax
```

SLAX 1.1 Quick Reference

Phil Shafer <phil@juniper.net>
http://code.google.com/p/libslax

SLAX is an alternative encoding for XSLT, the W3C standard XML transformation language. Since SLAX encodes the same informationas XSLT, scripts can be easily converted between the two formats. The SLAX syntax is similar to C and Perl. Scripts are simpler to write, easier to debug, more maintainable, and programmers are more productive. The SLAX distribution includes a debugger with a profiler, a call flow monitor, and much more.

```
# An Example SLAX Script
version 1.1;

param $name = "Poe";

var $favorites := {
   <name> "Doyle";
   <name hidden="yes"> "Spillane";
}

match / {     # Matches root of the input tree
   <top> {
      /* Parameters are passed by name */
      call test($elt = "author", $name);
   }
}

template test ($name, $elt = "default") {
   for $this ($favorites/name) {
      if ($name == $this && not(@hidden)) {
         element $elt {
            copy-of .//author[name/last == $this];
         }
      } else if ($name == $this) {
         message "Hidden: " _ $name;
      }
   }
}
```

SLAX Syntax

SLAX 1.0 Users: Features new to 1.1 are marked with ‡

List of Top-Level Statements:

attribute-set ‡	include	output-method ‡	template
decimal-format ‡	key ‡	param	var
function ‡	match	preserve-space ‡	version
import	mvar ‡	strip-space ‡	

List of Block-Level Statements:

apply-imports ‡	element ‡	number ‡	trace ‡
apply-templates	expr	param	uexpr ‡
attribute ‡	fallback ‡	processing-instruction	
call	for-each	result ‡	use-attribute-sets ‡
comment	for ‡	set ‡	var
copy-node ‡	if	sort ‡	while ‡
copy-of	message ‡	terminate ‡	

Elements and Attributes:

< *name attr1* **=** *val1 attr2* **=** *val2* **>;**
< *name* **>** *xpath-expression* **;**
< *name* **> {** *body* **}**

Creates an element with the given name and attribute values. The element and attribute names must be tokens, not XPath expressions. Attribute values are expressions.

element *name* **{** *body* **}**
attribute *name* **{** *body* **}**

Creates an element or attribute with the given name and a value given by a block of statements.

Variables and Parameters:

variable-value can be an *xpath-expression, an* **<***element***>**, a **{** *body* **}**, or a **call** statement

The initial value for a parameter or variable can be either an XPath expression, an element, or the result of a block of statements. The **":="** operator can be used instead of **"="** to avoid Resultant Tree Fragments (RTFs) by calling the **node-set()** extension function. A semi-colon after a braces-enclosed body is optional.

param $*pname [* **=** *variable-value]* **;**

Define a parameter, passed into the script, function, or template. If no value is passed in, the initial value given will be used.

var $*vname* = *variable-value* **;**

Define a variable, either local to a particular scope, or global to the script.

mvar $*vname* [= *variable-value]* **;**

Define a mutable variable, whose value can be changed using the **set** and **append** statements. The initial value for mvars is optional.

XPath Syntax

Axis Names:

ancestor::	**following-sibling::**
ancestor-or-self::	**namespace::**
attribute::	**parent::**
child::	**preceding::**
descendant::	**preceding-sibling::**
descendant-or-self::	
following::	**self::**

Concatenation:

SLAX uses underscore as the concatenation operator:

var $all = "these " _ are _ " concatenated";

Using Node Sets as Arrays:

A predicate containing a single number will match the nth member of a node-set, which can be used to treat node sets as arrays with origin 1. The **number()** function must be used to convert strings into numbers:

var $x = authors[4];
var $y = homes[number($x/index)];

Common EXSLT functions:

ns dyn = "http://exslt.org/dynamic";
object **dyn:evaluate**(*xpath-expression)*

ns esxl = "http://exslt.org/common";
string **exsl:object-type**(*object)*

ns math = "http://exslt.org/math";
number **math:abs**(*number)*
number **math:atan2**(*number, number)*
number **math:cos**(*number)*
(also **sin***(),* **tan***(),***acos***(),* **asin***(), and* **atan***())*
number **math:constant**(*name, precision)*
number **math:exp**(*number)*
number **math:log**(*number)*
number **math:power**(*base, power)*
number **math:random**()
number **math:sqrt**(*number)*

ns set = "http://exslt.org/sets";
node-set **set:difference**(*node-set, node-set)*
node-set **set:distinct**(*node-set)*
boolean **set:has-same-node**(*node-set, node-set)*
node-set **set:intersection**(*node-set, node-set)*
node-set **set:leading**(*node-set, up-to-node)*
node-set **set:trailing**(*node-set, after-node)*

ns str = "http://exslt.org/strings";
string **str:align**(*string, padding, alignment?)*
string **str:decode-uri**(*uri, encoding?)*
string **str:encode-uri**(*string, escape-all, encoding?)*
string **str:padding**(*number, string?)*
node-set **str:replace**(*string, search, replace)*
node-set **str:split**(*string, pattern?)*
node-set **str:tokenize**(*string, delimiters?)*

JUNOScript Functions

void **jcs:close** *(connection)*

Close a connection opened by **slax:open()**.

node-set **jcs:execute** *(connection, rpc)*

Execute an RPC over a connection, which can be either local or remote. The response is returned.

node-set **jcs:get-hello** *(connection)*

Return the contents of the **<hello>** message for a **NETCONF** connection.

string **jcs:get-protocol** *(connection)*

Return the protocol is use for a connection. Valid protocols include **"netconf"**, **"junos-netconf"**, and **"junoscript"**.

string **jcs:hostname** *(string)*

Return the DNS hostname for an IPv4 address, IPv6 address, or hostname.

node-set **jcs:invoke** *(object)*

Execute an RPC on the current host and return the results. The argument can be either a string containing a single RPC method name, or a node set contains the RPC.

object **jcs:open** *()*
object **jcs:open** *(target)*
object **jcs:open** *(target, username, password)*
object **jcs:open** *(target, object)*

Open a connection to the local box or to a remote one. The object can contain these elements:

<method> "junoscript" | **"netconf"**
<usernane> *string*
<passphrase> *passphrase or password*

void **jcs:output** *(string+)*

Emit a message to the user immediately, which consists of the concatenation of all arguments.

node-set **jcs:parse-ip** *(string)*

Return information about IPv4/IPv6 address/prefix.

$res[1] = Hostname or NULL on error
$res[2] = Address family, **"inet4"** or **"inet6"**
$res[3] = Prefix length
$res[4] = Network Address
$res[5] = Netmask if inet4 (empty for inet6)

void **jcs:progress** *(string+)*

If progress messages are enable, emit a message immediately to the user consisting of the concatenation of all the arguments.

Expressions:

expr *xpath-expression* **;**
uexpr *xpath-expression* **;**
Emits the string value of an expression. If **uexpr** is used, the value is emitted with the normal escaping mechanism disabled, which may allow invalid XML.

Changing Mutable Variables:

Be aware that mutable variables use non-standard SLAX-specific extension elements. Use of **mvar**s can affect the portability of your script.

set *$vname* **=** *variable-value* **;**
Set the value of a mutable variable. The variable must be defined using **mvar** and in scope.

append *$vname* **+=** *variable-value* **;**
Append a value to the node set contained in a mutable variable. The variable must be defined using **mvar** and in scope.

Output:

message *xpath-expression* **;**
message { *body* **}**
Display a message immediately to the user.

trace *xpath-expression* **;**
trace { *body* **}**
Write a message to the trace file, if tracing is enabled.

terminate *xpath-expression* **;**
terminate { *body* **}**
Display a message and exit the script immediately.

Namespaces:

ns *[prefix [ns-options]* **=** *] uri-string* **;**
Declares a namespace with an optional prefix. The ns-options are:
exclude exclude from output
extension defines extension elements

ns-alias *script-prefix result-prefix* **;**
Map a prefix used in the script to one that should be used in the emitted output.

ns-template *xpath-expression* **;**
Set the namespace for the node built by an **element** or **attribute** statement.

XPath Functions

String Functions:

string **concat***(string, string, string*)*
boolean **contains***(target-string, sub-string)*
string **normalize-space***(string?)*
boolean **starts-with***(target-string, sub-string)*
string **string***(object?)*
number **string-*length***(string?)*
string **substring***(string, offset, length?)*
string **substring-after***(string, sub-string)*
string **substring-before***(string, sub-string)*
string **translate***(base-string, if-str, then-str)*

Node Set Functions:

number **last***()*
number **position***()*
number **count***(node-set)*
node-set **id***(object)*
string **local-name***(node-set?)*
string **namespace-uri***(node-set?)*
string **name***(node-set?)*

Boolean Functions:

boolean **boolean***(object)*
boolean **not***(object)*
boolean **true***()*
boolean **false***()*
boolean **lang***(string)*

Number Functions:

number **number***(object?)*
number **sum***(node-set)*
number **floor***(number)*
number **ceiling***(number)*
number **round***(number)*

XSLT Functions:

node-set **current***()*
node-set **document***(object, node-set?)*
boolean **element-available***(element-name)*
string **format-number***(number, format-name)*
boolean **function-available***(function-name)*
string **generate-id***(node-set?)*
node-set **key***(key-name, object)*
object **system-property***(property-name)*

Templates

match *xpath-pattern* **{** *body* **}**
A match template matches on the given XPath pattern. When XSLT processing finds a node that matches the given pattern, the template's block of statements will be executed.

template *name [* **(** *parameters* **)** *]* **{** *body* **}**
A named template is explicitly called using the **call** statement. The body of the template contains a set of instructions that are executed.

call *name* **;**
call *name* **(** *parameters* **) ;**
call *name [* **(** *parameters* **)** *]* **{** *with-stmts* **}**
Named templates can be called

apply-templates *[xpath-expression]* **;**
apply-templates { *with-parameters* **} ;**
Recursively inspect child nodes, attempting to find matching templates to execute. If an XPath expression is given, recursion is done on the nodes selected by that expression.

mode *string* **;**
Set the mode for a template, or restrict the mode for apply-templates.

priority *number* **;**
Set the priority for a template.

Using parameters:

Template parameters are passed by name, not position:
call test($message = "EOF seen");

Defining: *$pname [* **=** *xpath-expression]*
In addition to the **param** statement, parameters can be defined inside a set of parentheses following the name of a named template. An optional XPath expression defines the default value of the parameter, which is used if the caller does not specific a value.

Passing: *$pname [* **=** *xpath-expression]*
Parameters can be passed using the name of the parameter. An XPath expression can be used to supply a value for the parameter, but if none is given, the current value of that variable is used.

with *$pname = [xpath-expression]* **;**
with *$pname* = **{** *body* **} ;**
Used to pass parameters to **match** templates via **apply-templates** and to pass block output to named templates.

Control Statements:

for-each (*xpath-expr* **) {** *body* **}**
Execute a block of statements using each member of a node set as the context node.

for *$vname* **(** *xpath-expr* **) {** *body* **}**
Execute a block of statements using each member of a node set as the value of the given variable.

while (*xpath-expr* **) {** *body* **}**
Execute a block of statements until an XPath expression evaluates to **true**. An mvar should be used in the expression to ensure an infinite loop is not created.

if (*condition* **) {** *body* **}**
[**else if (** *condition* **) {** *body* **}** *] [* **else {** *body* **}** *]*
Conditional execution of blocks of statements based on conditional XPath expressions.

sort *[xpath-expression]* **;**
sort *[xpath-expression]* **{**
  case-order "upper-first" | **"lower-first" ;**
  data-type "text" | **"number"** | *type-name* **;**
  order "ascending" | **"descending" ;**
}
Control the order in which **for-each** or **apply-templates** iterates through nodes.

Sequences:

for-each (*1* **...** *10* **) {** *body* **}**
for *$vname* **(** *$min* **...** *$max* **) {** *body* **}**
The **"..."** operator generates a sequence of nodes with a value of each integer between the left and right operands. If the left operand is less than the right one, the numbers are generated in decreasing order.

Functions:

function *qname* **(** *parameters* **) {** *body* **}**
Define an extension function that can be used in an XPath expression, using EXSLT's <func:function>. The body is a set of block statements and should include a **result** statement.

result *xpath-expression* ;
results { *body* **}**
Specifies the return value for a function, using EXSLT's <func:result>. The results can be either a simple scalar value or an XML hierarchy.

Top-Level Statements:

version *value* **;**
Must be first statement in a SLAX script. It is mandatory. Value must be “1.0” or “1.1”.

include *file-spec* **;**
import *file-spec* **;**
Include the contents of a SLAX or XSLT file.

attribute-set *name* **{** *body* **}**
Define a set of attributes used with the **use-attribute-sets** statement.

key *name* **{**
  match *xpath-pattern* ;
  value *xpath-expression* ;
}
Defines a key for use with the key() function.

decimal-format *name* {
  decimal-separator "." ;
  digit "#" ;
  grouping-separator "," ;
  infinity "Infinity" ;
  minus-sign "-" ;
  nan "NaN";
  pattern-separator ";" ;
  percent "%";
  per-mille "\x2030" ;
  zero-digit "0" ;
}
Define a format used with the **format-number()** function.

output-method [xml | text | html] {
  cdata-section-elements *name-list* ;
  doctype-public *string* ;
  doctype-system *string* ;
  encoding *string* ;
  indent "yes" | **"no"** ;
  media-type *string* ;
  omit-xml-declaration "yes" | **"no"** ;
  standalone "yes" | **"no"** ;
  version *version-string* ;
}
Controls how output data is emitted.

preserve-space *list-of-elements* **;**
strip-space *list-of-elements* **;**
Preserve or remove whitespace inside the given elements when emitting output.

Additional Statements:

apply-imports ;
Process the context node using only the match template imported by the current script.

comment *xpath-expresion* **;**
Creates an XML comment with given value.

copy-of *xpath-expresion* **;**
Copies a complete XML hierarchy (node set or fragment) specified by the XPath expression.

copy-node ;
copy-node { body **}**
Copies the current node and its namespaces, but not attributes or child nodes. The optional body is a block of statements that can emit additional nodes to be places inside that copy.

fallback { *body* **}**
Used when an extension function or element is not available in the current implementation. The body is executed to handle this error condition.

number *xpath-expr* **{**
  format *numbering-style* ;
  letter-value "alphabetic" | "traditional" ;
  grouping-size *number* ;
  grouping-separator *character*;
  language *language-name* ;
}
number {
  level "single" | "multiple" | "any";
  from *xpath-expr-when-to-start* ;
  count *xpath-expr-what-to-count* ;
}
Formats or generates a number for output.

processing-instruction *xpath-expr* **;**
processing-instruction *xpath-expr* **{** *body* **}**
Creates an XML processing instruction with a name given by an XPath expression and an optional body that provides a value.