

This module is an XIST namespace. It can be used for generating Oracle database functions that return XML strings. This is done by embedding processing instructions containing PL/SQL code into XML files and transforming those files with XIST.

An example that generates an HTML table containing the result of a search for names in a person table might look like this:

```
from ll.xist import xsc
from ll.xist.ns import html, htmlspecials
from ll import toxic

class search(xsc.Element):
    def convert(self, converter):
        e = xsc.Frag(
            toxic.args("search varchar2"),
            toxic.vars("i integer;"),
            toxic.type("varchar2(32000);"),
            htmlspecials.plaintable(
                toxic.code("""
                    i := 1;
                    for row in (select name from person where name like search) loop
                        """),
                html.tr(
                    html.th(toxic.expr("i"), align="right"),
                    html.td(toxic.expr("xmlescape(row.name)")),
                ),
                toxic.code("""
                    i := i+1;
                    end loop;
                """)
            )
        )
        return e.convert(converter)

print toxic.xml2ora(search().conv().asString(encoding="us-ascii")).encode("us-ascii")
```

Running this script will give the following output (the indentation will be different though):

```
(
  search varchar2
)
return varchar2
as
  c_out varchar2(32000);
  i integer;
begin
  c_out := c_out || '<table cellpadding="0" border="0" cellspacing="0">';
  i := 1;
  for row in (select name from person where name like search) loop
    c_out := c_out || '<tr><th align="right">';
    c_out := c_out || i;
    c_out := c_out || '</th><td>';
    c_out := c_out || xmlescape(row.name);
    c_out := c_out || '</td></tr>';
    i := i+1;
  end loop;
  c_out := c_out || '</table>';
```

```

    ◦ return c_out;
end;

```

Instead of generating the XML from a single XIST element, it's of course also possible to use an XML file. One that generates the same function as the one above looks like this:

```

<?args
  ◦ search varchar2
?>
<?vars
  ◦ i integer;
?>
<plaintable class="search">
  ◦ <?code
  ◦ ◦ i := 1;
  ◦ ◦ for row in (select name from person where name like search) loop
  ◦ ◦ ◦ ?>
  ◦ ◦ ◦ <tr>
  ◦ ◦ ◦ ◦ <th align="right"><?expr i?></th>
  ◦ ◦ ◦ ◦ <td><?expr xmlescape(row.name)?></td>
  ◦ ◦ ◦ </tr>
  ◦ ◦ ◦ <?code
  ◦ ◦ ◦ i := i + 1;
  ◦ ◦ end loop;
  ◦ ?>
</plaintable>

```

When we save the file above as `search.sqlxsc` then parsing the file, transforming it and printing the function body works like this:

```

from ll.xist import parsers
from ll.xist.ns import html, htmlspecials
from ll import toxic

node = parsers.parseFile("search.sqlxsc")
node = node.conv()
print toxic.xml2ora(node.asString(encoding="us-ascii")).encode("us-ascii")

```

## 1. def stringify(string, nchar=False):

Format string as multiple PL/SQL string constants or expressions. `nchar` specifies if a NVARCHAR constant should be generated or a VARCHAR. This is a generator.

## 2. def xml2ora(string):

The unicode object `string` must be an XML string. `xml2ora` extracts the relevant processing instructions and creates the body of an Oracle function from it.

## 3. def prettify(string):

Try to fix the indentation of the PL/SQL snippet passed in.

## 4. class args(11.xist.xsc.ProcInst):

Specifies the arguments to be used by the generated function. For example:

```
<?args
  ▫ key in integer,
  ▫ lang in varchar2
?>
```

If `args` is used multiple times, the contents will simply be concatenated.

## 5. class vars(11.xist.xsc.ProcInst):

Specifies the local variables to be used by the function. For example:

```
<?vars
  ▫ buffer varchar2(200) := 'foo';
  ▫ counter integer;
?>
```

If `vars` is used multiple times, the contents will simply be concatenated.

## 6. class code(11.xist.xsc.ProcInst):

A PL/SQL code fragment that will be embedded literally in the generated function. For example:

```
<?code select user into v_user from dual;?>
```

## 7. class expr(11.xist.xsc.ProcInst):

The data of an `expr` processing instruction must contain a PL/SQL expression whose value will be embedded in the string returned by the generated function. This value will not be escaped in any way, so you can generate XML tags with `expr` PIs but you must make sure to generate the value in the encoding that the caller of the generated function expects.

## 8. class proc(11.xist.xsc.ProcInst):

When this processing instruction is found in the source `xml2ora` will no longer generate a function as a result, but a procedure. This procedure must have `c_out` as an “out” variable (of the appropriate type (see `type`) where the output will be written to.

## 9. class type(11.xist.xsc.ProcInst):

Can be used to specify the return type of the generated function. The default is `clob`.