UCL insides

if you need to go deeper

UCL in C code

- LibUCL provides a rich API to deal with UCL objects
- Zero-copy mode
- References count (using atomic operations)
- Dictionaries (using very efficient **KHash** structure)
- Automatically growing arrays (using vector, not linked lists)

Example of using UCL in C

```c
obj = ucl_object_typed_new (UCL_OBJECT);

/* Keys replacing */
cur = ucl_object_fromstring_common ("value1", 0, UCL_STRING_TRIM);
ucl_object_insert_key (obj, cur, "key0", 0, false);
cur = ucl_object_fromdouble (0.1);
/* Create some strings */
cur = ucl_object_fromstring_common ("  test string    ", 0, UCL_STRING_TRIM);
ucl_object_insert_key (obj, cur, "key1", 0, false);
cur = ucl_object_fromstring_common ("  test \nstring\n    ", 0, UCL_STRING_TRIM);
ucl_object_insert_key (obj, cur, "key2", 0, false);
cur = ucl_object_fromstring_common ("  test string    \n", 0, 0);
ucl_object_insert_key (obj, cur, "key3", 0, false);
/* Array of numbers */
ar = ucl_object_typed_new (UCL_ARRAY);
cur = ucl_object_fromint (10);
ucl_array_append (ar, cur);
cur = ucl_object_fromdouble (10.1);
ucl_array_append (ar, cur);
cur = ucl_object_fromdouble (9.999);
ucl_array_prepend (ar, cur);
```

Extending UCL

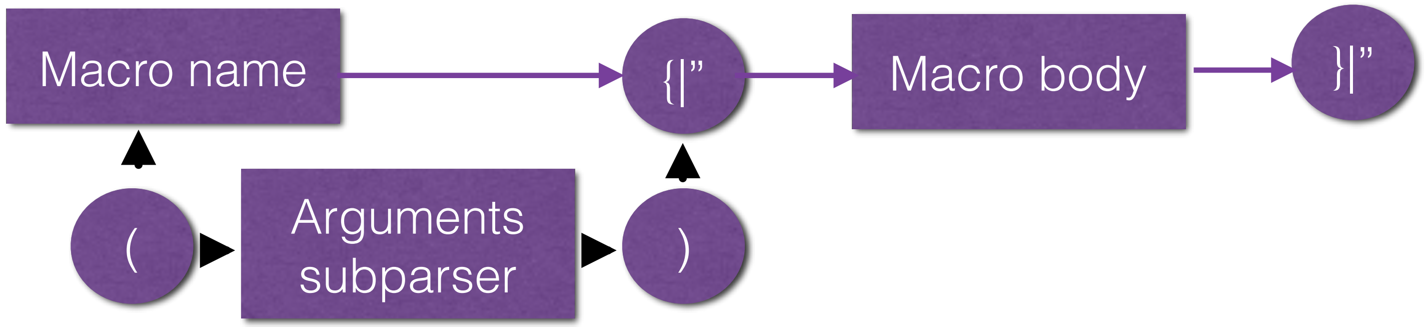
- Libucl is designed to be extendable language
- Extendability by macros
- Extendability by variables

UCL macros

- From side of input file
- From C code
- Invocation diagram

.macro(params) “value”

```
bool ucl_macro_handler (const unsigned char *data, size_t len,
        const ucl_object_t *arguments,
        void* ud)
```



Missing features

- Macro context:

```
.if (key == value) {
    other_key = "value";
}
```

  - Very hard to define all context (not merely previous one)
  - Breaks macro API
- Save macros’ positions when emitting data

UCL variables

- Variables in UCL: key = “${CURDIR}”
- Variables from C:

```
void ucl_parser_register_variable (struct ucl_parser *parser, const char *var,
        const char *value);
```

- What if we have no variable but:
    - ${VAR} or
    - $VAR

Issues with variables

- Too complex API for unknown variables:

```
bool ucl_variable_handler (const unsigned char *data, size_t len,
  unsigned char **replace, size_t *replace_len, bool *need_free, void* ud);
```

- Ambiguity in unknown variables handling

- Incompatible with zero-copy mode

Other languages bindings

- Internal bindings:
  - LUA
  - C++11
  - Python
- External bindings:
  - Go
  - Rust
  - ??? (your favourite language here)

Bindings example

```
local config = {
  options = {
    filters = {'spf', 'dkim', 'regexp'},
    url_tld = tld_file,
    dns = {
      nameserver = {'8.8.8.8'}
    },
  },
  logging = {
    type = 'console',
    level = 'debug'
  },
  metric = {
    name = 'default',
    actions = {
      reject = 100500,
    },
    unknown_weight = 1
  }
}

print(ucl.to_format(config, ‘ucl’))
```

Custom emitters

- Emitters can be highly customised:
  - Custom output functions (file, fd, string and so on)
  - Custom formats (JSON, pretty JSON, YAML, UCL, MessagePack)
  - Streamline emitters

Custom emitter example

- Glib GString emitter:

```
void
rspamd_ucl_emit_gstring (ucl_object_t *obj,
    enum ucl_emitter emit_type,
    GString *target)
{
    struct ucl_emitter_functions func = {
        .ucl_emitter_append_character = rspamd_gstring_append_character,
        .ucl_emitter_append_len = rspamd_gstring_append_len,
        .ucl_emitter_append_int = rspamd_gstring_append_int,
        .ucl_emitter_append_double = rspamd_gstring_append_double
    };

    func.ud = target;
    ucl_object_emit_full (obj, emit_type, &func);
}
```

Unsolved problems

- Preserving comments and other servicing stuff, 2 possible ways:
  - Save positions inside objects
  - Save shadow copy of the original configuration
- Ambiguity with implicit arrays

Saving context

Keep data in objects

- Relatively cheap in terms of memory and processing
- Cannot save the full document structure:
    - positions can be ambiguous;
    - absolutely nothing to do with macros;
    - variables saving is also tricky;
    - multiline and single line comments;
    - very complex and intrusive implementation

Saving context

Shadow context

- The idea is to copy the original document and use it during emitting
  - need to save somehow that the content of objects have been changed (tricky and unsafe as icl object structure is public for modifications);
  - cannot work with different emitter/input;
  - **BUT** can deal with macros and variables in a non-intrusive way

Implicit arrays

- Good points:
  - Simplifies configuration

```
section {
        name = "abc";
        param = "value";
}
section {
        name = "cba";
        param = “other_value";
}
```

- Bad points:
  - Complexity with iterations:

```
while ((val = ucl_iterate_object (obj, &it, true)) != NULL) {
    LL_FOREACH (val, cur) {
    …
    }
}
```

  - Complexity for other formats conversion (all but msgpack)

Thanks for attention

vsevolod@FreeBSD.org