Interpreter Events

Table of Contents

1. Concept

In order to support writing debuggers, profilers, analyzers and other useful tools that are needed for efficient programming, the interpreter is offering a generic event-oriented low-level API on top of which all these tools can be written. It is similar to the `parse/trace` and `lexer/trace` instrumentation approach.

2. Interpreter instrumentation

In order to access internal interpreter states, the interpreter is generating events at key points of his internal code evaluation process. Those events can be captured using a user-provided callback function. Events are emitted only if a *tracing* mode is enabled in the interpreter using `/trace` refinement on `do`.

```
do/trace <code> <callback>

<code>     : a block of code to evaluate in tracing mode.
<callback> : a callback function, triggered on each interpreter event.
```

Examples

3. Events

When the tracing mode is active, the interpreter will trigger events described below. Events can be grouped into the following categories:

- Global events: `INIT`, `END`

- Evaluating a block/paren of code: `ENTER`, `EXIT`
- Calling any type of function: `OPEN`, `CALL`, `RETURN`
- Evaluating a function body block: `PROLOG`, `EPILOG`
- Expression evaluation: `FETCH`, `PUSH` , `SET`, `ERROR`
- Exceptions handling: `THROW`, `CATCH`

Detailed description:

Event	Code	Offset	Value	Ref	Description
`INIT`	`none`	-1	n/a (`none`)	n/a (`none`)	when the tracing mode is initiated (`do/trace` call).
`END`	`none`	-1	n/a (`none`)	n/a (`none`)	when the tracing mode is ended (`do/trace` call exiting).
`ENTER`	`block!`, `paren!`	-1	n/a (`none`)	n/a (`none`)	when a block is about to be evaluated.
`EXIT`	`block!`, `paren!`	-1	n/a (`none`)	n/a (`none`)	when current evaluated block's tail has been reached.
`OPEN`	`block!`, `paren!`	`integer!`	`any-function!`	`word!`, `path!`	when a new function (any-function!) call is initiated and a new stack frame is opened.
`CALL`	`block!`, `paren!`	`integer!`	`any-function!` to call	`word!`, `path!`, `any-function!`	a function with all arguments fetched on the stack gets called.
`RETURN`	`block!`, `paren!`	`integer!`	returned `any-type!` value	`word!`, `path!`	when a function call has returned and its stack frame has been closed.
`PROLOG`	`block!`, `paren!`	-1	called `function!` value	`word!`, `path!`	when entering a function.
`EPILOG`	`block!`, `paren!`	-1	called `function!` value	`word!`, `path!`	when exiting a function.
`FETCH`	`block!`, `paren!`	`integer!`	fetched `any-type!` value	n/a (`none`)	a value is read from the input block to be evaluated.
`PUSH`	`block!`, `paren!`	`integer!`	pushed `any-type!` value	n/a (`none`)	a value has been pushed on the stack.
`SET`	`block!`, `paren!`	`integer!`	`any-type!`	`set-word!`, `set-path!`	a set-word or set-path is set to a value.

Event	Code	Offset	Value	Ref	Description
ERROR	none	-1	error! value	n/a (none)	when an error occurs and is about to be thrown up in the stack.
THROW	none	-1	thrown any-type! value	n/a (none)	when a value is thrown using throw native.
CATCH	none	-1	thrown any-type! value	n/a (none)	when a value is caught using catch native.

Events come with extra information:

- code: when available, it provides the input block! or paren! series currently interpreted.
- offset: when different from -1, indicates the input series offset at the event moment.
- value: when available, the currently processed value.
- ref: when available, references the word or path which evaluation produced the current event/value.

4. Event handler

```
func [
    event [word!]
    code  [block! paren! none!]
    value [any-type!]
    ref   [any-type!]
    frame [pair!]
][
    [events]                    ;-- optional restricted event names list
    ...body...
]
```

5. Predefined tools

5.1. debug

Debugger commands:

- next or n or just ENTER: evaluate next value.
- continue or c: exit debugging console and continue evaluation.
- stack or s: display the current calls and expression stack.
- parents or p: display the parents call stack.
- :word: outputs the value of word. If it is a function!, outputs the local context.

- `:a/b/c`: outputs the value of `a/b/c` path.
- `watch <word1> <word2>⋯`: watch one or more words. `w` can be used as shortcut for `watch`.
- `-watch <word1> <word2>⋯`: stop watching one or more words. `-w` can be used as shortcut for `-watch`.
- `+stack` or `+s`: outputs expression stack on each new event.
- `-stack` or `-s`: do not output expression stack on each new event.
- `+locals` or `+l`: output local context for each entry in the callstack.
- `-locals` or `-l`: do not output local context for each entry in the callstack.
- `+indent` or `+i`: indent the output of the expression stack.
- `-indent` or `-i`: do not indent the output of the expression stack.

5.2. `profile`

TBD

5.3. `trace`

TBD

5.4. Dumping raw events

TBD

6. Implementation notes