Implementing JoyLoL

Dr Stephen Gaito
PerceptiSys Ltd
December 4, 2018

Perish *then* Publish Press.

2

2

Contents

1 Introduction

1.1 Overview

1.1.1 Introduction

We will provide full formal descriptions of six distinctly different programming languages, JoyLoL, WhileLoL, WhileRecLoL, EagerLambdaLoL, LazyLambdaLoL and LogicLoL. The languages WhileLoL and WhileRecLoL will be in the same overall class of languages as most standard imperative programming languages such as, Lua, C, Pascal, Algol, Java, Python, PHP, Ruby, etc. Most programmers will recognize WhileRecLoL as a close, but simplified, cousin to the languages they are currently using. The languages EagerLambaLoL and LazyLambdaLoL represent the class of eager and lazy functional languages such as ML and Haskell respectively. The LogicLoL language represents the class of logic programming lanaguages such as Prolog.

The JoyLoL language, is in a new class of 'concatenative' languages originally explored by Manfred von Thun[1]. The primary importance of JoyLoL is that it *is* a fixed point of the formal semantics operator. As a fixed point of the semantics operator, JoyLoL provides a foundation for both computation and more importantly Mathematics[2].

Across all of these languages the constant similarity is the 'LoL' or List of Lists. In each language the *only* expressions are Lists of Lists. Depending upon the language, these lists of lists are potentially infinite expressions, which will, however, always have a finite description at any particular point in a computation.

As developed over the past 50 years, the formal semantics operator has three parts:

1. Denotational Semantics (roughly equivalent to Tarski's model semantics)
2. Operational Semantics (roughly equivalent to Gentzen's natural deduction)
3. Axiomatic Semantics (roughly equivalent to Type theory)

Good introductions to these three types of formal semantics can be found in [Win93] and [Gun92]. (TODO see [AV80] and [Bil90] for early reports of Prolog's semantics)

The collection of Lists of Lists *is* an "infinitely" "complex" "structure". In its complete incarnation, it is strictly *more* complex than the whole of any formal *set* theory such as ZFC[3]. This is, for finite beings, such as mere mortal mathematicians, a large and complex 'world' to explore. It is a world in which it is very easy to get lost. While we assert that JoyLoL provides a computational foundation for

[1] For example, see [Thu94b], [Thu94a] or [Thu94c] all of which can be found in [Thu11] or [Thu05].
[2] While we assert that JoyLoL provides a *computational* foundation for Mathematics, proving this assertion will be the work of many (future) papers. We will not even attempt a proof in this document.
[3] In this paper we will only consider the component which corresponds to classical, ω-computation.

Mathematics, to help us 'mere mortal mathematicians' orient ourselves, we will often make reference to classical mathematical concepts. It is important to realize that these classical mathematical concepts are simply aids to our mathematical intuition, and not formal statements.

The most important classical intuition is that of Algebra and CoAlgebra, or equivalently, for a Computer Scientist, that of Data and Process. Both classical Mathematics and Computational theory have, by and large, explicitly limited themselves to the well-founded and terminating, largely to avoid Poincaré's 'Vicious Circles'. We will see that the non-well-founded, non-terminating processes, Cantor's 'Absolute Infinite', have a surprisingly easily understood structure, essentially dual to classical set theory. However, the *computational* theory of these non-terminating processes, has a profound impact on Mathematics. By ignoring this computational theory, we, as mathematicians, make simple problems, hard.

Intuitively, a List of Lists, is a potentially infinite structure which records a potentially infinite *structured* collection of observations of a potentially non-terminating process of processes. As *finite* mathematicians, we can only ever manipulate finite structures, *finite records of observations* of a potentially infinte process. One such record of observations might be denoted by, for example:

```
(() ( ( )))
```

This is of course the denotation of Lists in John McCarthy's Lisp, see [MAE+65].

Where classically, formal semantics concentrated on *one* denotation, to provide a *formal* description of these potentially non-terminating process, it is critical that we carefully distinguish *two* distinct denotations: the classical *algebraic* denotation (corresponding to a least fixed point of the semantics operator) and the non-classical *coAlgebraic* denotation (corresponding to a greatest fixed point of the semantics operator). While for data, the data itself is its own denotation, for a potentially non-terminating process, *an answer* (a data object) is insufficient to *denote* that process. Instead the appropriate denotation of a non-terminating process is its trace of observations (or any finite record of this trace which is 'sufficient' for current purposes). Similarly, while the 'big-step' operational semantics might suffice for a data object or a terminating process, the 'one-step' operational semantics is the only definition of operational semantics appropriate for a potentially non-terminating process. Finally, to provide an Axiomatic semantics, with out recourse to classical first order set theory, we will make essential use of finite descriptions of the traces of potentially non-terminating processes.

Philosophically, it is important to know when two 'things' are the 'same'. For an algebraic list of lists, two lists are equal if there is a *finite*, structurally inductive, comparison of the two objects. This is the familiar concept of recursive equality. For sets this is *extensive* equality. For a coAlgebraic list of lists, two potentially non-terminating processes are equal if they respond in the same way to any collection of 'observations'. This is the concept, from Theoretical Computer Science and CoAlgebraic Category theory, of 'bisimulation'.

1.1.2 How certain can certainty be?

We intend to show that JoyLoL provides a foundation for Mathematics. Any foundation for Mathematics, must be 'certain', but what exactly does 'certainty' mean and how 'certain' can a finite computational artefact be?

(TODO: there are two aspects here: (1) implementation vs idea (logical-formalism vs intuitionism) and (2) current mathematical certainty expressed *using* logical-formalism vs other possible approaches)

sCurrently, certainty in mathematics, is generally identified with the *computation* of the 'Logical' 'Truth' of a 'formal' assertion, which represents *a logical implementation*, of an intuitive understanding, of an idea which we want to show is ***'certain'***.

Notice that, as with any *human* language, there are many different possible ways to express the 'same' intuitive thought.

Notice as well that there are multiple different levels of granularity with which to express and 'prove' a given statement to a 'sufficient' 'level' of detail. While most current mathematical 'proofs' are conducted in an informal but rigorous style, the generally acknowledged highest standard of proof is a natural deduction proof expressed using first, second or higher order logical notation. (TODO wrong words!) At the moment, the translation of a high-level mathematical argument into a 'completely rigorous' but impossibly detailed logical argument is very tedious and difficult, largely because the highly detailed, n-th order logical notation, is very far from the original intuitive idea to be proved.

There are many discussions of the Logical-Formalist 'schools' of the foundations of Mathematics, [Gia02], [Sha00] and [Hat82] each provide interesting accounts of the strengths and weaknesses of these approaches. Equally important in any of these expositions, are the accounts of the Intuitionist critique of the logical-formalist approach.

For a (software) Business Analyst or Systems Architect, the key words from the above discussion are, 'computation', 'implementation', 'specification', and 'algorithm'. For the Business Analyst, every specification is always just one of many possible *implementations* of the business problem to be solved. Each different possible phrasing of the specification carries different 'non-functional' (or extra-specificational) implications for the way a given business problem is 'solved'. Equally, for the Systems Architect, each proposed software implementation satisfying a given specification, has different non-functional implications in terms of, for example, speed, memory usage, and programmer or maintenance effort. The critical point here is "*an* implementation is just that *an* implementation", one of many possible solutions to a problem. Each of which provides different capabilities or penalties. As any Business Analyst or Systems Architect knows, business problems can only ever be solved by *an* implementation, but to keep business flexibility in a highly dynamical environment, it should be as easy as possible to *change* implementations

as and when those implementations begin to limit the business growth. Implementations are critical to a business, *but* implementations come and go, the goal is always to solve the business problem.

For our purposes, the classical Logical-Formalist approach using, for example, set theory expressed in a first order logic formalism, or type theory[4], ...

Instead of computing logical truth that a given structure exists, using Gentzen's natural deduction (algorithm), compute the structure itself. However, if we *compute* a structure, how do we know that we have computed the structure we specified?

TODO: We want to discuss deduction vs induction in the generation of knowledge. Mathematics is (almost) pure deduction. However any human subject 'to be done' must include aspects of scientific/engineering induction. That is the verification of any *deductive* mathematical proof, requires some computational system to behave 'correctly'. How do we know that any particular verificational computation of a given proof is correct?

Deductively we can be certain a given computation is, *in theory*, correct. However conducting the *actual* computation entails dependence on *inductively* determined *models* of 'reality' which may or may not apply in a given time or place.

Arthur C. Clarke's 'Nine Billion Names of God', or Anthony S. Haines' 'And on gloomy Sunday...', in two different ways, suggest how potentially extremely rare events, the coincidence of naming all the names of God, or a 'research agency' outside our existence, could both have profound *global* impacts on any given 'computation'. The point here is not that either of these stories are 'true' (though they *could be*), but rather that potentially rare events might break any given model of physically realized computation. There will never be any way a *finite* being can mitigate against these rare events. Given the rarity of these example events in that they will only occur *once* in a given existence, it is not, on the whole, rational, to worry about rare events such as these. So, a *finite* being, can *never* be 100% *certain* of any computation, but we can be fairly certain, or rather, for all rational purposes, a computation can be considered to be *certain enough*.

We can 'draw the line' between deductive certainty and inductively good enough models at various levels in the 'computational hierarchy'. We could take the quite considerable effort to deductively verify the whole computational infrastructure (compilers, operating system, peripherals, CPU, memory, transistor states, etc.) down to the Quantum-mechanical level. Or, we could simply assume a good-enough model of computation of a computational language and then deductively verify any given program in that language. However, no mater where we draw this line between deductive certainty and inductively good enough models, a line must be chosen. The result of any particular computation will only ever be 'good enough'.

As with any security issue, we have a risk / benefit analysis to conduct. We then have to make difficult choices as to how to minimize the costs of the risks, maximize

[4] See for example, [ACV13].

the value of the benefits, while simultaneously minimizing the costs of the effort all required to get the chosen low-level of risks and high-level of benefits.

For current Mathematical practice, an individual mathematician 'verifying' a proof, is likely to be *highly* error prone. However it is assumed that over all interested mathematicians, any mistakes in a given proof *will be found*. To help reduce the difficulty of understanding (i.e. verifying) any given proof statement, mathematicians spend a lot of effort identifying independently useful lemmas from which to build simpler proof statements to a wide range of similar mathematical problems.

Similarly our collective confidence in a given proof of correctness of a given program, will come from running the verification on multiple *independently different* platforms (the equivalent of multiple mathematicians). Our collective confidence will also be increased if the program is structured out of a 'library' of simpler and independently useful 'parts', each of which are verified and more importantly regularly used on a wide range of range of platforms and in a wide range of problems.

See [Mac01] and [Lak76] TODO: provide references to relevant royal society conference

Bootstrap and circularity

Provide box diagram of working parts

white/black box testing.... formal model testing (finite automata corresponding to any finite operational structure) alasthe use of pre/post conditions is insufficient to provide any meaningful input to a model tester as the 'real' complexity of a given operational transition is in the parts not checked in the pre/post-condition HOWEVER, the parts not checked SHOULD NOT effect the transition. So I guess this might be tested. NO unfortunately any computation by case analysis will break this ability. Since the cases will be hidden to the external specifications.... white vs black box...

1.1.3 Judgements

TODO: cover judgements as base case coAlgebraic 'sets'.

1.2 The syntax and semantics of JoyLoL

1.2.1

The formal semantics of the *other* programming languages are certainly simpler. This is because, the semantics of the other languages, are defined in terms of the semantics of JoyLoL. This means that any deep subtleties are simply encapsulated in the semantics of JoyLoL. Any reader who is willing to take the existence of the formal semantics of JoyLoL as given, are welcome to skip to the other chapters until they are ready or willing to understand the full import of the semantics of JoyLoL.

The essential subtleties of our formal semantics for JoyLoL comes from three areas:

1. **Foundations**
   We are explicitly working *without* classical first order set theory. Yet formal semantics *is* defined as *mappings* between *collections* of 'things'.
   How do we define mappings and collections while we are defining semantics with which to define mappings and collections with which... ?[5]
2. **Metamathematics**
   A partial answer to our foundational question above
   Any formal semantics is a programming language *about* another 'object-level' programming language (which is itself a language *about* objects as 'values').
3. **Computation of *properties* of potentially non-terminating processes**

We begin by listing the syntax of JoyLol together with its associated semantics. We loosely follow the presentation in [Win93].

[5] I personally know of no completely rigorous exposition of the foundations of classical mathematics. I suspect the closest, mathematically, we might come close to this is in Gödel's proofs first and second incompleteness results. However, even here the results hinge upon an informal interpretation of 'truth' at the meta-level.

1.3 The syntax of WhileRecLoL

1.4 CPU Model

In [<gaito2018hilbertsProgramRevisitedBaseCase>] we modelled the *simple* model of computation which we call `JoyLoL`. However there are no implementations of this `JoyLoL` model of computation in a commercially available CPU. In this chapter we define a model of an idealized *commercially* available CPU upon which we can rigorously implement pure `JoyLoL`.

In both our models of computation, pure `JoyLoL` and an idealized commercial CPU, the important distinction is the underlying memory models. The pure `JoyLoL` model of computation is essentially a *restricted access* Harvard model of memory. The pure `JoyLoL` model has a pair of memories one each for the data and the process. Each memory is structured as a stack for which only the 'top' of the stack can be directly accessed.

We tend to assume that our commercial CPUs have a von Neumann model of memory consisting of *one, large, randomly accessible* memory at the Instruction Set Architectural (ISA) interface[6]. As we will discuss below, for a mathematician, this assumption, of a *single, large, randomly accessible* memory, is misguided.

For a 64 bit CPU, the *maximal* addressable memory is 16 Exabytes. For a mathematician, this is *tiny* when compared to ω. We could model a 128 bit CPU, or, an x bit CPU for any particular value of x we might choose, however for any *finite* value of x, we will still have a *maximal* addressable memory which is *tiny* compared to ω. While we could consider an idealized ω bit CPU, no such *implementable* CPU exists. Our goal here is to model a *realizable* CPU upon which `JoyLoL` might be able to run efficiently.

1.4.1 Model details

1. Storage
    1. uint64, uint32, uint16, uint8
    2. int64, int32, int16, int8
    3. byte, utf8Char, char

[6] Note that most modern CPUs actually have a modified Harvard architecture at level of the the underlying micro-architecture, since they implement a pair of data and instruction caches between the Random Access Memory (RAM) and the CPU itself. We will, for our purposes, ignore this underlying micro-architecture since we are only interested in modelling the ISA interface. At the level of the ISA, both data and instructions are 'loaded' from 'one' 'large' (randomly accessible) Memory (RAM).

4. at the moment we do *not* model floats or doubles
5. we also do *not* explicitly model pointers. (Should we?)
6. arrays
7. structures

2. Actions
   1. array indexing
   2. addition, subtraction, mutiplication, division
   3. and, or, xor, ...
   4. assignments
   5. functions

2 Examples

In this part we collect a number of important worked examples.

2.1 Lists

We explore the use of `JoyLoL` to provided packed lists, aka arrays.
How do we show two coalgebras to be bisimilar?
We need to define:

- **joylolNatural** need to define natural numbers object. Then provide recursive definitions of addition and multiplication. Then translations from strings in various bases.
- **bytesNatural** need to define natural numbers object. Then provid recursive definitions of addition and multiplication. Then translation from strings in various bases.
- **gmpNatural** need to define natural numbers object. Then provide recursive definitions of addition and multiplication. Then translation from strings in various bases.
- **byte** (or **uint8**)
- **uint64**
- **int64**
- **lists**
- **array**

```
typdef array
```

2.2 Sets

34 34

2.3 Hash Tables

2.4 AVL Trees

We explore the use of `JoyLoL` by building an implementation of AVL trees. We need to define:

- **byte** (or **uint8**)
- **uint64**
- **int64**
- **lists**
- **array**
- **uft8Char**
- **Symbols**
- **JoyLoLObjs**
- **DictNodeObjs**

```
typedef struct dictNode_object_struct {
  JObj         super;
  Symbol      *symbol;
  JObj        *preObs;
  JObj        *value;
  JObj        *postObs;
  DictNodeObj *left;
  DictNodeObj *right;
  DictNodeObj *previous;
  DictNodeObj *next;
  size_t       height;
  long         balance;
} DictNodeObj;
```

3 Base CoAlgebras

39 39

3.1 Assertions

3.1.1 Goals

An Assertion has exactly two possible values, ‘true’ or ‘false’.

3.1.2 Code

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```c
typedef struct assertion_object_struct {
  JObj     super;
  JObj    *assertion;
} AssertionObj;

#define ASSERTION_FLAG_MASK  0x8L
#define asAssertion(aLoL)             \
  (((AssertionObj*)(aLoL))->assertion)

#define assertionValue(aLoL)              \
  (((aLoL)->flags) & ASSERTION_FLAG_MASK)
```

3.1.2.1 Test Suite: newAssertion

CHeader : public

```c
typedef JObj* (NewAssertion)(
  JoyLoLInterp *jInterp,
  JObj         *anAssertion
);

```

Code 3.1.2

```
#define newAssertion(jInterp, anAssertion)       \
  (                                              \
    assert(getAssertionsClass(jInterp)           \
      ->newAssertionFunc),                       \
    (getAssertionsClass(jInterp)                 \
      ->newAssertionFunc(jInterp, anAssertion)) \
  )
```

CHeader : private

```
extern JObj* newAssertionImpl(
  JoyLoLInterp *jInterp,
  JObj         *anAssertion
);
```

CCode : default

```
JObj* newAssertionImpl(
  JoyLoLInterp *jInterp,
  JObj         *anAssertion
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj* result = newObject(jInterp, AssertionsTag);
  assert(result);

  result->type  = jInterp->coAlgs[AssertionsTag];
  result->flags |= ASSERTION_FLAG_MASK; // default is TRUE
  asAssertion(result) = anAssertion;
  return result;
}
```

Test case

should create a new assertion

```
  AssertPtrNotNull(jInterp);

  JObj* aNewBool      = newBoolean(jInterp, TRUE);
  JObj* aNewAssertion = newAssertion(jInterp, aNewBool);
  AssertPtrNotNull(aNewAssertion);
  AssertPtrNotNull(asType(aNewAssertion));
  AssertIntEquals(asTag(aNewAssertion), AssertionsTag);
  AssertPtrNotNull(asAssertion(aNewAssertion));
```

```
  AssertPtrEquals(asAssertion(aNewAssertion), aNewBool);
  AssertIntTrue(asFlags(aNewAssertion));
  AssertIntTrue(isAtom(aNewAssertion));
  AssertIntTrue(isAssertion(aNewAssertion));
  AssertIntFalse(isPair(aNewAssertion));
```

Test case

print Assertion

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* trueBool = newBoolean(jInterp,   TRUE);
  JObj* aLoL     = newAssertion(jInterp, trueBool);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "{ true } ");
  strBufClose(aStrBuf);

  JObj *falseBool = newBoolean(jInterp,   FALSE);
  aLoL            = newAssertion(jInterp, falseBool);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "{ false } ");
  strBufClose(aStrBuf);
```

3.1.2.2 Test Suite: parseAssertion

CHeader : public

```
1  typedef AssertionObj *(ParseAssertion)(
2    JoyLoLInterp *jInterp,
3    Symbol       *assertionStr
4  );
5
6  #define parseAssertion(jInterp, assertionStr)       \
7    (                                                 \
8      assert(getAssertionsClass(jInterp)              \
9        ->parseAssertionFunc),                        \
```

Code 3.1.2

```
    (getAssertionsClass(jInterp)                    \
      ->parseAssertionFunc(jInterp, assertionStr))  \
  )
```

CHeader : private

```
extern AssertionObj* parseAssertionImpl(
  JoyLoLInterp *jInterp,
  Symbol       *assertionStr
);
```

CCode : default

```
AssertionObj* parseAssertionImpl(
  JoyLoLInterp *jInterp,
  Symbol       *assertionStr
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj *assertionList = NULL;
  if (assertionStr && (assertionStr[0] != 0)) {
    TextObj *assertionText = createTextFromString(jInterp, assertionStr);
    assertionList = parseAllSymbols(assertionText);
    freeText(assertionText);
  }
  if (!assertionList) assertionList = newPair(jInterp, NULL, NULL);

  return (AssertionObj*)newAssertionImpl(jInterp, assertionList);
}
```

Test case

should parse an assertion

```
  AssertPtrNotNull(jInterp);
  AssertionObj *aNewAssertion = parseAssertion(jInterp, " true ");
  AssertPtrNotNull(aNewAssertion);
  AssertPtrNotNull(asType(aNewAssertion));
  AssertIntEquals(asTag(aNewAssertion), AssertionsTag);
  AssertPtrNotNull(asAssertion(aNewAssertion));
  AssertIntTrue(asFlags(aNewAssertion));
  AssertIntTrue(isAtom(aNewAssertion));
  AssertIntTrue(isAssertion(aNewAssertion));
```

```
AssertIntFalse(isPair(aNewAssertion));
AssertIntTrue(isPair(asAssertion(aNewAssertion)));
AssertIntTrue(isSymbol(asCar(asAssertion(aNewAssertion))));
AssertStrEquals(asSymbol(asCar(asAssertion(aNewAssertion))), "true");
```

3.1.2.3 Test Suite: isAssertion

CHeader : public

```
#define isAssertion(aLoL)                \
  (                                      \
    (                                    \
      (aLoL) &&                          \
      asType(aLoL) &&                    \
      (asTag(aLoL) == AssertionsTag)     \
    ) ?                                  \
      TRUE :                             \
      FALSE                              \
  )
```

3.1.2.4 Test Suite: isTrue and isFalse

CHeader : public

```
#define assertionTrue(aLoL)                \
  (                                        \
    (                                      \
      (aLoL) &&                            \
      asType(aLoL) &&                      \
      (asTag(aLoL) == AssertionsTag) &&    \
      assertionValue(aLoL)                 \
    ) ?                                    \
      TRUE :                               \
      FALSE                                \
  )
#define assertionFalse(aLoL)               \
  (                                        \
    (                                      \
      (!aLoL) ||                           \
      (!asType(aLoL)) ||                   \
      (asTag(aLoL) != AssertionsTag) ||    \
      !assertionValue(aLoL)                \
```

```
    ) ?                                 \
      TRUE :                            \
      FALSE                             \
  )
```

Test case

should return appropriate assertion values

```
JObj *trueBool    = newBoolean(jInterp,   TRUE);
JObj *anAssertion = newAssertion(jInterp, trueBool);
AssertPtrNotNull(anAssertion);
AssertPtrNotNull(asType(anAssertion));
AssertIntEquals(asTag(anAssertion), AssertionsTag);
AssertIntTrue(assertionValue(anAssertion));
AssertIntTrue(assertionTrue(anAssertion));
AssertIntFalse(assertionFalse(anAssertion));
```

CHeader : private

```
extern Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != AssertionsTag) return FALSE;
  if (asAssertion(lolA) != asAssertion(lolB)) return FALSE;
  return TRUE;
```

```
16  }
```

3.1.2.5 Test Suite: printing assertions

CHeader : private

```
1  extern Boolean printAssertionCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  );
```

CCode : default

```
1  Boolean printAssertionCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *anAssertion,
4    size_t           timeToLive
5  ) {
6    assert(anAssertion);
7    assert(asType(anAssertion));
8    assert(asTag(anAssertion) == AssertionsTag);
9    JObj *theAssertionBody = asAssertion(anAssertion);

10   DEBUG(aStrBuf->jInterp, "printAssertionsCoAlg %p %p %zu %p\n",
11     aStrBuf, anAssertion, timeToLive, theAssertionBody);

12   if (timeToLive < 1) {
13     strBufPrintf(aStrBuf, "... ");
14     return TRUE;
15   }

16   size_t printedOk = TRUE;
17   if (theAssertionBody) {
18     strBufPrintf(aStrBuf, "{ ");
19     printedOk =
20       (asType(theAssertionBody)->printFunc)
21         (aStrBuf, theAssertionBody, timeToLive-1);
22     strBufPrintf(aStrBuf, "} ");
23   } else {
24     strBufPrintf(aStrBuf, "{ } ");
25   }
26
```

```
27    return printedOk;
28  }
```

Test case

should print assertions

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[AssertionsTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* trueBoolean   = newBoolean(jInterp,   TRUE);
  JObj* aNewAssertion = newAssertion(jInterp, trueBoolean);
  AssertPtrNotNull(aNewAssertion);
  printLoL(aStrBuf, aNewAssertion);
  AssertStrEquals(getCString(aStrBuf), "{ true } ");
  strBufClose(aStrBuf);

  JObj* falseBoolean = newBoolean(jInterp,   FALSE);
  aNewAssertion      = newAssertion(jInterp, falseBoolean);
  AssertPtrNotNull(aNewAssertion);
  printLoL(aStrBuf, aNewAssertion);
  AssertStrEquals(getCString(aStrBuf), "{ false } ");
  strBufClose(aStrBuf);
```

3.1.2.6 Test Suite: registerAssertions

CHeader : public

```
1   typedef struct assertions_class_struct {
2     JClass       super;
3     NewAssertion      *newAssertionFunc;
4     ParseAssertion    *parseAssertionFunc;
5   } AssertionsClass;
```

CCode : default

```
1   static Boolean initializeAssertions(
2     JoyLoLInterp *jInterp,
3     JClass   *aJClass
4   ) {
```

```
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerAssertions(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerAssertions(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  AssertionsClass* theCoAlg
    = joyLoLCalloc(1, AssertionsClass);
  assert(theCoAlg);

  theCoAlg->super.name           = AssertionsName;
  theCoAlg->super.objectSize     = sizeof(AssertionObj);
  theCoAlg->super.initializeFunc = initializeAssertions;
  theCoAlg->super.registerFunc   = registerAssertionWords;
  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
  theCoAlg->super.printFunc      = printAssertionCoAlg;
  theCoAlg->newAssertionFunc     = newAssertionImpl;
  theCoAlg->parseAssertionFunc   = parseAssertionImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == AssertionsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Assertions coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerAssertions
  AssertPtrNotNull(jInterp);
```

```
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getAssertionsClass(jInterp));
  AssertionsClass *coAlg = getAssertionsClass(jInterp);
  registerAssertions(jInterp);
  AssertPtrNotNull(getAssertionsClass(jInterp));
  AssertPtrEquals(getAssertionsClass(jInterp), coAlg);
  AssertIntEquals(
    getAssertionsClass(jInterp)->super.objectSize,
    sizeof(AssertionObj)
  )
```

3.1.3 Words

```
(
  (pushData true)
)

(
  (pushData false)
)

(
  (popData aBool)
  (doIfte aBool
    (pushData true)
    (pushData false)
  )
)

(
  (popData aBool)
  (doIfte aBool
    (pushData false)
    (pushData true)
  )
)

(
  (popData aBool)
  (doIfte aBool
    (pushData false)
    (pushData true)
  )
```

```
)

(
  (popData aBool1)
  (popData aBool2)
  (doIfte aBool1
    (pushData true)
    (doIfte aBool2
      (pushData true)
      (pushData false)
    )
  )
)

(
  (popData aBool1)
  (popData aBool2)
  (doIfte aBool1
    (doIfte aBool2
      (pushData true)
      (pushData false)
    )
    (pushData false)
  )
)
```

CCode : default

```
1  static void isAssertionAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isAssertion(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, result);
12 }
```

CCode : default

```
1  static void assertionIsTrueAP(ContextObj* aCtx) {
2    assert(aCtx);
```

```
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, TRUE);
  } else {
    result = newBoolean(jInterp, FALSE);
  }
  pushCtxData(aCtx, result);
}
```

CCode : default

```
static void assertionIsFalseAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, FALSE);
  } else {
    result = newBoolean(jInterp, TRUE);
  }
  pushCtxData(aCtx, result);
}
```

CHeader : private

```
extern Boolean registerAssertionWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerAssertionWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);

  extendJoyLoLInRoot(jInterp, "isAssertion",      "", isAssertionAP,      "");
```

```
  extendJoyLoLInRoot(jInterp, "assertionIsTrue",  "", assertionIsTrueAP,
"");
  extendJoyLoLInRoot(jInterp, "assertionIsFalse", "", assertionIsFalseAP,
"");

  return TRUE;
}
```

3.1.4 Lua functions

CCode : default

```
static int lua_assertions_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_assertions [] = {
  {"gitVersion", lua_assertions_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_assertions (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerAssertions(jInterp);
  luaL_newlib(lstate, lua_assertions);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedAssertions` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedAssertions(
  lua_State *lstate
```

```
);
```

CCode : default

```
Boolean requireStaticallyLinkedAssertions(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_assertions(lstate);
  lua_setfield(lstate, -2, "joylol.assertions");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.1.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_assertions(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
//#include <joylol/dictNodes.h>
//#include <joylol/dictionaries.h>
#include <joylol/texts.h>
#include <joylol/parsers.h>
#include <joylol/pairs.h>
#include <joylol/cFunctions.h>
#include <joylol/booleans.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/assertions-private.h>
// dictionary
```

17
```
// printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.booleans");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.parsers");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.contexts");
requireStaticallyLinkedAssertions(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.2 Booleans

3.2.1 Goals

A Boolean has exactly two possible values, ‘true’ or ‘false’.

3.2.2 Code

3.2.2.1 Test Suite: newBoolean

CHeader : public

```
typedef JObj* (NewBoolean)(
  JoyLoLInterp*,
  Boolean
);

#define newBoolean(jInterp, aBool)      \
  (                                     \
    assert(getBooleansClass(jInterp)    \
      ->newBooleanFunc),                \
    (getBooleansClass(jInterp)          \
      ->newBooleanFunc(jInterp, aBool)) \
  )
#define BOOLEAN_FLAG_MASK 0x8L
#define asBoolean(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
);
```

CCode : default

```
JObj* newBooleanImpl(
  JoyLoLInterp* jInterp,
  Boolean aBoolean
) {
  assert(jInterp);
  assert(jInterp->coAlgs);
```

```
7   JObj* result = newObject(jInterp, BooleansTag);
8   assert(result);

9   result->type  = jInterp->coAlgs[BooleansTag];
10  if (aBoolean) {
11    result->flags |= BOOLEAN_FLAG_MASK;
12  } else {
13    result->flags &= ~BOOLEAN_FLAG_MASK;
14  }
15  return result;
16 }
```

Test case

should create a new boolean

```
AssertPtrNotNull(jInterp);

JObj* aNewBoolean = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aNewBoolean);
AssertPtrNotNull(asType(aNewBoolean));
AssertIntEquals(asTag(aNewBoolean), BooleansTag);
AssertIntTrue(asFlags(aNewBoolean));
AssertIntTrue(isAtom(aNewBoolean));
AssertIntTrue(isBoolean(aNewBoolean));
AssertIntFalse(isPair(aNewBoolean));
```

Test case

print Boolean

```
AssertPtrNotNull(jInterp);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

JObj* aLoL = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "true ");
strBufClose(aStrBuf);

aLoL = newBoolean(jInterp, FALSE);
```

```
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

3.2.2.2 Test Suite: isBoolean

CHeader : public

```
#define isBoolean(aLoL)                \
  (                                    \
    (                                  \
      (aLoL) &&                        \
      asType(aLoL) &&                  \
      (asTag(aLoL) == BooleansTag)     \
    ) ?                                \
      TRUE :                           \
      FALSE                            \
  )
```

3.2.2.3 Test Suite: isTrue and isFalse

CHeader : public

```
#define isTrue(aLoL)                     \
  (                                      \
    (                                    \
      (aLoL) &&                          \
      asType(aLoL) &&                    \
      (asTag(aLoL) == BooleansTag) &&    \
      asBoolean(aLoL)                    \
    ) ?                                  \
      TRUE :                             \
      FALSE                              \
  )
#define isFalse(aLoL)                    \
  (                                      \
    (                                    \
      (!aLoL) ||                         \
      (!asType(aLoL)) ||                 \
      (asTag(aLoL) != BooleansTag) ||    \
      !asBoolean(aLoL)                   \
```

```
    ) ?                                 \
      TRUE :                            \
      FALSE                             \
  )
```

Test case

should return appropriate boolean values

```
JObj *aBool = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aBool);
AssertPtrNotNull(asType(aBool));
AssertIntEquals(asTag(aBool), BooleansTag);
AssertIntTrue(asBoolean(aBool));
AssertIntTrue(isTrue(aBool));
AssertIntFalse(isFalse(aBool));
aBool = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aBool);
AssertPtrNotNull(asType(aBool));
AssertIntEquals(asTag(aBool), BooleansTag);
AssertIntFalse(asBoolean(aBool));
AssertIntFalse(asBoolean(aBool));
AssertIntFalse(isTrue(aBool));
AssertIntTrue(isFalse(aBool));
```

CHeader : private

```
extern Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
```

```
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != BooleansTag) return FALSE;
  if (asBoolean(lolA) != asBoolean(lolB)) return FALSE;
  return TRUE;
}
```

3.2.2.4 Test Suite: printing booleans

CHeader : private

```
extern Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == BooleansTag);

  if (asBoolean(aLoL)) strBufPrintf(aStrBuf, "true ");
  else strBufPrintf(aStrBuf, "false ");
  return TRUE;
}
```

Test case

should print booleans

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[BooleansTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);
```

```
JObj* aNewBoolean = newBoolean(jInterp, TRUE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "true ");
strBufClose(aStrBuf);

aNewBoolean = newBoolean(jInterp, FALSE);
AssertPtrNotNull(aNewBoolean);
printLoL(aStrBuf, aNewBoolean);
AssertStrEquals(getCString(aStrBuf), "false ");
strBufClose(aStrBuf);
```

3.2.2.5 Test Suite: registerBooleans

CHeader : public

```
1  typedef struct booleans_class_struct {
2    JClass       super;
3    NewBoolean      *newBooleanFunc;
4  } BooleansClass;
```

CCode : default

```
1  static Boolean initializeBooleans(
2    JoyLoLInterp *jInterp,
3    JClass   *aJClass
4  ) {
5    assert(jInterp);
6    assert(aJClass);
7    return TRUE;
8  }
```

CHeader : private

```
1  extern Boolean registerBooleans(JoyLoLInterp *jInterp);
```

CCode : default

```
1  Boolean registerBooleans(JoyLoLInterp *jInterp) {
2    assert(jInterp);
3    assert(jInterp->coAlgs);

4    BooleansClass* theCoAlg
```

```
    = joyLoLCalloc(1, BooleansClass);
  assert(theCoAlg);

  theCoAlg->super.name           = BooleansName;
  theCoAlg->super.objectSize     = sizeof(JObj);
  theCoAlg->super.initializeFunc = initializeBooleans;
  theCoAlg->super.registerFunc   = registerBooleanWords;
  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
  theCoAlg->super.printFunc      = printBoolCoAlg;
  theCoAlg->newBooleanFunc       = newBooleanImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == BooleansTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Booleans coAlg

```
// CTestsSetup has already created a jInterp
// and run registerBooleans
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getBooleansClass(jInterp));
BooleansClass *coAlg = getBooleansClass(jInterp);
registerBooleans(jInterp);
AssertPtrNotNull(getBooleansClass(jInterp));
AssertPtrEquals(getBooleansClass(jInterp), coAlg);
AssertIntEquals(
  getBooleansClass(jInterp)->super.objectSize,
  sizeof(JObj)
)
```

3.2.3 Words

(

```
  (pushData true)
)

(
  (pushData false)
)

(
  (popData aBool)
  (doIfte aBool
    (pushData true)
    (pushData false)
  )
)

(
  (popData aBool)
  (doIfte aBool
    (pushData false)
    (pushData true)
  )
)

(
  (popData aBool)
  (doIfte aBool
    (pushData false)
    (pushData true)
  )
)

(
  (popData aBool1)
  (popData aBool2)
  (doIfte aBool1
    (pushData true)
    (doIfte aBool2
      (pushData true)
      (pushData false)
    )
  )
)

(
  (popData aBool1)
```

```
  (popData aBool2)
  (doIfte aBool1
    (doIfte aBool2
      (pushData true)
      (pushData false)
    )
    (pushData false)
  )
)
```

CCode : default

```
1  static void isBooleanAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isBoolean(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, result);
12 }
```

CCode : default

```
1  static void isTrueAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isTrue(top)) {
8      result = newBoolean(jInterp, TRUE);
9    } else {
10     result = newBoolean(jInterp, FALSE);
11   }
12   pushCtxData(aCtx, result);
13 }
```

CCode : default

```
1  static void isFalseAP(ContextObj* aCtx) {
2    assert(aCtx);
```

```
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isTrue(top)) {
    result = newBoolean(jInterp, FALSE);
  } else {
    result = newBoolean(jInterp, TRUE);
  }
  pushCtxData(aCtx, result);
}
```

CHeader : private

```
extern Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerBooleanWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  ContextObj *rootCtx = jInterp->rootCtx;
  assert(rootCtx);
  DictObj *dict = rootCtx->dict;
  assert(dict);

  DictNodeObj* true  = getSymbolEntry(dict, "true");
  true->value = newBoolean(jInterp, TRUE);

  DictNodeObj* false = getSymbolEntry(dict, "false");
  false->value = newBoolean(jInterp, FALSE);

  extendJoyLoLInRoot(jInterp, "isBoolean", "", isBooleanAP, "");
  extendJoyLoLInRoot(jInterp, "isTrue",    "", isTrueAP,    "");
  extendJoyLoLInRoot(jInterp, "isFalse",   "", isFalseAP,   "");

  return TRUE;
}
```

3.2.4 Lua functions

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```c
static int lua_booleans_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_booleans [] = {
  {"gitVersion", lua_booleans_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_booleans (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerBooleans(jInterp);
  luaL_newlib(lstate, lua_booleans);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedBooleans` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedBooleans(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_booleans(lstate);
  lua_setfield(lstate, -2, "joylol.booleans");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.2.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_booleans(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
#include <joylol/dictionaries.h>
#include <joylol/texts.h>
#include <joylol/cFunctions.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/booleans.h>
#include <joylol/booleans-private.h>
// dictionary
```

```
15 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.contexts");
requireStaticallyLinkedBooleans(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.3 C-Functions

3.3.1 Goals

A C-Function represents a JoyLoL word implemented as an ANSI-C function which directly manipulates a Context.

3.3.2 Code

CHeader : public

```
1  typedef struct context_object_struct ContextObj;
2  typedef void (CFunction)(ContextObj*);
3  typedef ContextObj* (CtxCFunction)(ContextObj*);
4
5  #define CTX_CFUNCTION_FLAG 0x8L
6  typedef struct cFunction_object_struct {
7    JObj   super;
8    CFunction *func;
9  } CFunctionObj;
10
11 #define asCFunc(aLoL)              \
12   (((CFunctionObj*)(aLoL))->func)
13 #define asCtxCFunc(aLoL)                          \
14   ((CtxCFunction*)(((CFunctionObj*)(aLoL))->func))
```

3.3.2.1 Test Suite: newCFunction

CHeader : public

```
1  typedef CFunctionObj *(NewCFunction)(
2    JoyLoLInterp *jInterp,
3    CFunction    *aFunc
4  );
5
6  #define newCFunction(jInterp, aFunc)      \
7    (                                       \
8      assert(getCFunctionsClass(jInterp)    \
9        ->newCFunctionFunc),                \
10     (getCFunctionsClass(jInterp)          \
11       ->newCFunctionFunc(jInterp, aFunc)) \
```

Code 3.3.2

```
  )
typedef CFunctionObj *(NewCtxCFunction)(
  JoyLoLInterp *jInterp,
  CtxCFunction *aFunc
);

#define newCtxCFunction(jInterp, aFunc)       \
  (                                           \
    assert(getCFunctionsClass(jInterp)        \
      ->newCtxCFunctionFunc),                 \
    (getCFunctionsClass(jInterp)              \
      ->newCtxCFunctionFunc(jInterp, aFunc))  \
  )
```

CHeader : private

```
extern CFunctionObj *newCFunctionImpl(
  JoyLoLInterp *jInterp,
  CFunction *aFunc
);

extern CFunctionObj *newCtxCFunctionImpl(
  JoyLoLInterp *jInterp,
  CtxCFunction *aFunc
);
```

CCode : default

```
CFunctionObj *newCFunctionImpl(
  JoyLoLInterp *jInterp,
  CFunction    *aFunc
) {
  assert(jInterp);
  CFunctionObj* aNewFunc =
    (CFunctionObj*)newObject(jInterp, CFunctionsTag);
  assert(aNewFunc);
  aNewFunc->func  = aFunc;
  return aNewFunc;
}

CFunctionObj *newCtxCFunctionImpl(
  JoyLoLInterp *jInterp,
  CtxCFunction *aFunc
) {
```

```
17    assert(jInterp);
18    CFunctionObj* aNewFunc =
19      (CFunctionObj*)newObject(jInterp, CFunctionsTag);
20    assert(aNewFunc);
21    aNewFunc->func  = (CFunction*)aFunc;
22    aNewFunc->super.flags |= CTX_CFUNCTION_FLAG;
23    return aNewFunc;
24  }
```

```
  void testCFunction(ContextObj* aCtx) { }
  ContextObj* testCtxCFunction(ContextObj* aCtx) {
    return aCtx;
  }
```

Test case

should create a new CFunction

```
  AssertPtrNotNull(jInterp);

  CFunctionObj *aNewFunc =
    newCFunction(jInterp, testCFunction);
  AssertPtrNotNull(aNewFunc);
  AssertPtrNotEquals((void*)aNewFunc, (void*)testCFunction);
  AssertPtrNotNull(aNewFunc->super.type);
  AssertIntEquals(aNewFunc->super.tag, CFunctionsTag);
  AssertPtrEquals(aNewFunc->func, testCFunction);
  AssertIntTrue(isCFunction((JObj*)aNewFunc));
  AssertIntFalse(isCtxCFunction((JObj*)aNewFunc));
  AssertIntTrue(isAtom((JObj*)aNewFunc));
  AssertIntFalse(isPair((JObj*)aNewFunc));

  CFunctionObj *aNewCtxFunc =
    newCtxCFunction(jInterp, testCtxCFunction);
  AssertPtrNotNull(aNewCtxFunc);
  AssertPtrNotEquals((void*)aNewCtxFunc, (void*)testCtxCFunction);
  AssertPtrNotNull(aNewCtxFunc->super.type);
  AssertIntEquals(aNewCtxFunc->super.tag, CFunctionsTag);
  AssertPtrEquals((void*)aNewCtxFunc->func, (void*)testCtxCFunction);
  AssertIntTrue(isCFunction((JObj*)aNewCtxFunc));
  AssertIntTrue(isCtxCFunction((JObj*)aNewCtxFunc));
  AssertIntTrue(isAtom((JObj*)aNewCtxFunc));
```

```
  AssertIntFalse(isPair((JObj*)aNewCtxFunc));
```

CHeader : public

```
#define isCFunction(aLoL)                \
  (                                      \
    (                                    \
      (aLoL) &&                          \
      asType(aLoL) &&                    \
      (asTag(aLoL) == CFunctionsTag) && \
       asCFunc(aLoL)                     \
    ) ?                                  \
      TRUE :                             \
      FALSE                              \
  )
#define isCtxCFunction(aLoL)                    \
  (                                             \
    (                                           \
      (aLoL) &&                                 \
      asType(aLoL) &&                           \
      (asTag(aLoL) == CFunctionsTag) &&         \
      isFlagSet(aLoL, CTX_CFUNCTION_FLAG) &&  \
       asCFunc(aLoL)                            \
    ) ?                                         \
      TRUE :                                    \
      FALSE                                     \
  )
```

CHeader : private

```
Boolean equalityFuncCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityFuncCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
```

```
) {
  DEBUG(jInterp, "funcCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA) != CFunctionsTag) return FALSE;
  if (asCFunc(lolA) != asCFunc(lolB)) return FALSE;
  return TRUE;
}
```

3.3.2.2 Test Suite: print CFunction

CHeader : private

```
extern Boolean printFuncCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printFuncCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == CFunctionsTag);

  strBufPrintf(
    aStrBuf,
    "<%s%p> ",
    (isFlagSet(aLoL, CTX_CFUNCTION_FLAG) ? "ctx:" : ""),
    asCFunc(aLoL)
  );
  return TRUE;
}
```

Test case

should print CFuncion

```
  AssertPtrNotNull(jInterp);

  char buffer[100];
  buffer[0] = 0;
  sprintf((char*)&buffer, "<%p> ", testCFunction);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aNewFunc =
    (JObj*)newCFunction(jInterp, testCFunction);
  AssertPtrNotNull(aNewFunc);
  printLoL(aStrBuf, aNewFunc);
  AssertStrEquals(getCString(aStrBuf), buffer);
  strBufClose(aStrBuf);

  char ctxBuffer[100];
  ctxBuffer[0] = 0;
  sprintf((char*)&ctxBuffer, "<ctx:%p> ", testCtxCFunction);

  JObj* aNewCtxFunc =
    (JObj*)newCtxCFunction(jInterp, testCtxCFunction);
  AssertPtrNotNull(aNewCtxFunc);
  printLoL(aStrBuf, aNewCtxFunc);
  AssertStrEquals(getCString(aStrBuf), ctxBuffer);
  strBufClose(aStrBuf);
```

3.3.2.3 Test Suite: registerCFunctions

CHeader : public

```
1  typedef struct cFunctions_class_struct {
2    JClass           super;
3    NewCFunction    *newCFunctionFunc;
4    NewCtxCFunction *newCtxCFunctionFunc;
5  } CFunctionsClass;
```

CCode : default

```
1  static Boolean initializeCFunctions(
```

```
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerCFunctions(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerCFunctions(JoyLoLInterp *jInterp) {
  assert(jInterp);

  CFunctionsClass* theCoAlg    =
    joyLoLCalloc(1, CFunctionsClass);
  theCoAlg->super.name           = CFunctionsName;
  theCoAlg->super.objectSize     = sizeof(CFunctionObj);
  theCoAlg->super.initializeFunc = initializeCFunctions;
  theCoAlg->super.registerFunc   = registerCFunctionWords;
  theCoAlg->super.equalityFunc   = equalityFuncCoAlg;
  theCoAlg->super.printFunc      = printFuncCoAlg;
  theCoAlg->newCFunctionFunc     = newCFunctionImpl;
  theCoAlg->newCtxCFunctionFunc  = newCtxCFunctionImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == CFunctionsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register cFunctions

```
  // CTestsSetup has already created a jInterp
  // and run registerCFunctions
```

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getCFunctionsClass(jInterp));
  CFunctionsClass *coAlg =
    getCFunctionsClass(jInterp);
  AssertIntTrue(registerCFunctions(jInterp));
  AssertPtrNotNull(getCFunctionsClass(jInterp));
  AssertPtrEquals(getCFunctionsClass(jInterp), coAlg);
  AssertIntEquals(
    getCFunctionsClass(jInterp)->super.objectSize,
    sizeof(CFunctionObj)
  )
```

3.3.3 Lua interface

CCode : default

```
1  static const KeyValues gitVersionKeyValues[] = {
2    { "authorName",      "Stephen Gaito"},
3    { "commitDate",      "2018-12-03"},
4    { "commitShortHash", "38e0564"},
5    { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
6    { "subject",         "updated textadept lexer for JoyLoL"},
7    { "notes",           ""},
8    { NULL,                NULL}
9  };
```

CCode : default

```
1  static int lua_cFunctions_getGitVersion (lua_State *lstate) {
2    const char* aKey   = lua_tostring(lstate, 1);
3    if (aKey) {
4      getGitVersionInto(gitVersionKeyValues, aKey, aValue);
5      lua_pushstring(lstate, aValue);
6    } else {
7      lua_pushstring(lstate, "no valid key provided");
8    }
9    return 1;
10 }
11
12 static const struct luaL_Reg lua_cFunctions [] = {
13   {"gitVersion", lua_cFunctions_getGitVersion},
```

```
  {NULL, NULL}
};

int luaopen_joylol_cFunctions (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerCFunctions(jInterp);
  luaL_newlib(lstate, lua_cFunctions);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedCFunctions` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedCFunctions(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedCFunctions(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_cFunctions(lstate);
  lua_setfield(lstate, -2, "joylol.cFunctions");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.3.4 JoyLoL words

CCode : default

```
static void isCFunctionAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
```

```
7     if (isCFunction(top))
8       result = newBoolean(jInterp, TRUE);
9     else
10      result = newBoolean(jInterp, FALSE);
11    pushCtxData(aCtx, result);
12  }
```

CHeader : private

```
1   extern Boolean registerCFunctionWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoAlg
4   );
```

CCode : default

```
1   Boolean registerCFunctionWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoALg
4   ) {
5     assert(jInterp);
6     extendJoyLoLInRoot(jInterp, "isCFunction", "", isCFunctionAP, "");

7     return TRUE;
8   }
```

3.3.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
1   #include <stdlib.h>
2   #include <string.h>
3   #include <assert.h>
4   #include <joylol/jInterps.h>
5   #include <joylol/booleans.h>
6   #include <joylol/stringBuffers.h>
7   #include <joylol/cFunctions.h>
8   #include <joylol/texts.h>
9   #include <joylol/assertions.h>
10  #include <joylol/contexts.h>
```

```
11  #include <joylol/cFunctions-private.h>


    addJoyLoLLuaPath(lstate);
    requireStaticallyLinkedJInterps(lstate);
    requireLuaModule(lstate, "joylol.assertions");
    requireLuaModule(lstate, "joylol.pairs");
    requireLuaModule(lstate, "joylol.stringBuffers");
    requireLuaModule(lstate, "joylol.texts");
    requireLuaModule(lstate, "joylol.dictionaries");
    requireLuaModule(lstate, "joylol.dictNodes");
    requireLuaModule(lstate, "joylol.contexts");
    requireStaticallyLinkedCFunctions(lstate);
    getJoyLoLInterpInto(lstate, jInterp);
    initializeAllLoaded(lstate, jInterp);
    registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.4 CoAlgebras

3.4.1 Goals

An instance of the CoAlgebras coalgebra represents a given CoAlgebra.

3.4.2 Code

CHeader : public

```
1  typedef struct coAlgebra_object_struct CoAlgebraObj;
2
3  typedef struct coAlgebra_object_struct {
4    JObj   super;
5    // ??
6  } CoAlgebraObj;
7
8  //#define asCFunc(aLoL) (((CFunctionObj*)(aLoL))->func)
```

3.4.2.1 Test Suite: newCoAlg

CHeader : public

```
1  typedef CoAlgebraObj *(NewCoAlgebra)(
2    JoyLoLInterp *jInterp//,
3    //CFunction    *aFunc
4  );
5
6  #define newCoAlgebra(jInterp/*, aFunc*/)      \
7    (                                           \
8      assert(getCoAlgebrasClass(jInterp)    \
9        ->newCoAlgebraFunc),                \
10     (getCoAlgebrasClass(jInterp)          \
11       ->newCoAlgebraFunc(jInterp/*, aFunc*/)) \
12   )
```

CHeader : private

```
1  extern CoAlgebraObj *newCoAlgebraImpl(
2    JoyLoLInterp *jInterp//,
3    //CFunction *aFunc
4  );
```

CCode : default

```
CoAlgebraObj *newCoAlgebraImpl(
  JoyLoLInterp *jInterp//,
  //CFunction *aFunc
) {
  assert(jInterp);
  CoAlgebraObj* aNewCoAlgebra =
    (CoAlgebraObj*)newObject(jInterp, CoAlgebrasTag);
  assert(aNewCoAlgebra);
  //aNewFunc->func  = aFunc;
  return aNewCoAlgebra;
}
```

Test case

should create a new CoAlgebra

```
  AssertPtrNotNull(jInterp);

  CoAlgebraObj *aNewCoAlgebra = newCoAlgebra(jInterp);
  AssertPtrNotNull(aNewCoAlgebra);
  AssertPtrNotNull(aNewCoAlgebra->super.type);
  AssertIntEquals(aNewCoAlgebra->super.tag, CoAlgebrasTag);
  AssertIntTrue(isCoAlgebra((JObj*)aNewCoAlgebra));
  AssertIntTrue(isAtom((JObj*)aNewCoAlgebra));
  AssertIntFalse(isPair((JObj*)aNewCoAlgebra));
```

CHeader : public

```
#define isCoAlgebra(aLoL)                \
  (                                      \
    (                                    \
      (aLoL) &&                          \
      asType(aLoL) &&                    \
      (asTag(aLoL) == CoAlgebrasTag)     \
    ) ?                                  \
      TRUE :                             \
      FALSE                              \
  )
```

CHeader : private

```
extern Boolean equalityCoAlgebraCoAlg(
  JoyLoLInterp *jInterp,
```

```
  JObj          *lolA,
  JObj          *lolB,
  size_t         timeToLive
);
```

CCode : default

```
Boolean equalityCoAlgebraCoAlg(
  JoyLoLInterp *jInterp,
  JObj          *lolA,
  JObj          *lolB,
  size_t         timeToLive
) {
  DEBUG(jInterp, "coAlgebraCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA) != CoAlgebrasTag) return FALSE;
  return TRUE;
}
```

3.4.2.2 Test Suite: print CoAlgebra

CHeader : private

```
extern Boolean printCoAlgebarCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printCoAlgebraCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == CoAlgebrasTag);

  char ptoa[100];
```

```
10    sprintf(ptoa, "<CoAlgebra:%p> ", aLoL);
11    strBufPrintf(aStrBuf, ptoa);
12    return TRUE;
13  }
```

Test case

should print CoAlgebra

```
  AssertPtrNotNull(jInterp);

  char buffer[100];
  buffer[0] = 0;

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aNewCoAlgebra = (JObj*)newCoAlgebra(jInterp);
  AssertPtrNotNull(aNewCoAlgebra);

  sprintf((char*)&buffer, "<CoAlgebra:%p> ", aNewCoAlgebra);

  printLoL(aStrBuf, aNewCoAlgebra);
  AssertStrEquals(getCString(aStrBuf), buffer);
  strBufClose(aStrBuf);
```

3.4.2.3 Test Suite: registerCoAlgebras

CHeader : public

```
1  typedef struct coAlgebras_class_struct {
2    JClass    super;
3    NewCoAlgebra *newCoAlgebraFunc;
4  } CoAlgebrasClass;
```

CCode : default

```
1  static Boolean initializeCoAlgebras(
2    JoyLoLInterp *jInterp,
3    JClass   *aJClass
4  ) {
5    assert(jInterp);
6    assert(aJClass);
```

```
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerCoAlgebras(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerCoAlgebras(JoyLoLInterp *jInterp) {
  assert(jInterp);

  CoAlgebrasClass* theCoAlg    =
    joyLoLCalloc(1, CoAlgebrasClass);
  theCoAlg->super.name           = CoAlgebrasName;
  theCoAlg->super.objectSize     = sizeof(CoAlgebraObj);
  theCoAlg->super.initializeFunc = initializeCoAlgebras;
  theCoAlg->super.registerFunc   = registerCoAlgebraWords;
  theCoAlg->super.equalityFunc   = equalityCoAlgebraCoAlg;
  theCoAlg->super.printFunc      = printCoAlgebraCoAlg;
  theCoAlg->newCoAlgebraFunc     = newCoAlgebraImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == CoAlgebrasTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register coAlgebras

```
  // CTestsSetup has already created a jInterp
  // and run registerCoAlgebras

  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getCoAlgebrasClass(jInterp));
  CoAlgebrasClass *coAlg =
    getCoAlgebrasClass(jInterp);
```

```
  AssertIntTrue(registerCoAlgebras(jInterp));
  AssertPtrNotNull(getCoAlgebrasClass(jInterp));
  AssertPtrEquals(getCoAlgebrasClass(jInterp), coAlg);
  AssertIntEquals(
    getCoAlgebrasClass(jInterp)->super.objectSize,
    sizeof(CoAlgebraObj)
  )
```

3.4.3 Lua interface

CCode : default

```
1  static const KeyValues gitVersionKeyValues[] = {
2    { "authorName",      "Stephen Gaito"},
3    { "commitDate",      "2018-12-03"},
4    { "commitShortHash", "38e0564"},
5    { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
6    { "subject",         "updated textadept lexer for JoyLoL"},
7    { "notes",           ""},
8    { NULL,                NULL}
9  };
```

CCode : default

```
1  static int lua_coAlgebras_getGitVersion (lua_State *lstate) {
2    const char* aKey   = lua_tostring(lstate, 1);
3    if (aKey) {
4      getGitVersionInto(gitVersionKeyValues, aKey, aValue);
5      lua_pushstring(lstate, aValue);
6    } else {
7      lua_pushstring(lstate, "no valid key provided");
8    }
9    return 1;
10 }
11
12 static const struct luaL_Reg lua_coAlgebras [] = {
13   {"gitVersion", lua_coAlgebras_getGitVersion},
14   {NULL, NULL}
15 };
16
17 int luaopen_joylol_coAlgebras (lua_State *lstate) {
18   getJoyLoLInterpInto(lstate, jInterp);
19   registerCoAlgebras(jInterp);
```

```
  luaL_newlib(lstate, lua_coAlgebras);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedCoAlgebras` does just this.

CHeader : public

```
extern Boolean requireStaticallyLinkedCoAlgebras(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedCoAlgebras(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_coAlgebras(lstate);
  lua_setfield(lstate, -2, "joylol.coAlgebras");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.4.4 JoyLoL words

CCode : default

```
static void isCoAlgebraAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isCoAlgebra(top))
    result = newBoolean(jInterp, TRUE);
  else
    result = newBoolean(jInterp, FALSE);
  pushCtxData(aCtx, result);
```

```
12  }
```

CHeader : private

```
1   extern Boolean registerCoAlgebraWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoAlg
4   );
```

CCode : default

```
1   Boolean registerCoAlgebraWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoAlg
4   ) {
5     assert(jInterp);
6     extendJoyLoLInRoot(jInterp, "isCoAlgebra", "", isCoAlgebraAP, "");
7     return TRUE;
8   }
```

3.4.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
1   #include <stdlib.h>
2   #include <string.h>
3   #include <assert.h>
4   #include <joylol/jInterps.h>
5   #include <joylol/stringBuffers.h>
6   #include <joylol/booleans.h>
7   #include <joylol/cFunctions.h>
8   #include <joylol/texts.h>
9   #include <joylol/assertions.h>
10  #include <joylol/contexts.h>
11  #include <joylol/coAlgebras.h>
12  #include <joylol/coAlgebras-private.h>
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
```

```
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireStaticallyLinkedCoAlgebras(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.5 Contexts

3.5.1 Goals

JoyLoL Contexts form the kernel of the trampolined interpreter for JoyLoL.

A Context consists of the data and process stacks organized as two lists of lists. The evaluation or interpretation of JoyLoL in the context of a Context consists of taking the top item off the process stack and 'performing' it. 'Performing' a given JoyLoL word, will result in changes being made to both the data and process stacks. A Context is interpreted in an eval loop, until there are no more JoyLoL words on the process stack.

Question: are threads (POSIX-threads/native-threads) identified with contexts? Or are contexts the equivalent of 'green threads' (i.e. 'threads' emulated by JoyLoL rather than identified with the underlying Operating System's threads)? If contexts are not identified with POSIX-threads then we will need to mutex contexts (green threads) as 'data' structures, since it will be possible for multiple running threads to alter the same context (as a data structure). If contexts are identified with POSIX-threads, then there is no sense to our previous (joyLoL20170316) commands to 'switch contexts'.

CHeader : public

```
1 typedef struct context_object_struct ContextObj;
```

3.5.1.1 Thoughts

The current implementation pops the top of the data of the old context and pushes it onto the *data* of the new context.

However once something gets onto the data of a context it is "inert"... and won't be executed... any commands that we want to run in a different context need to be pushed on the process NOT the data

What should happen with a given context's process stack is emptied? Should all processing (of everything) cease.. or should a context switch to its parent context? At the moment.... processing stops, which is I guess the correct continuation.

The best way to think about context switching is as a potentially very large number of cooperating entities working on separate parts of a very large list of lists structure. The fact that they are cooperating means that they choose to hand-off to another context and hope to be resumed at some point in the "future" and restart computation where the original context handed off. This means that the process stack MUST NOT be altered. However the data stack might be and this represents the only way of communication between the contexts. Note that any given context can ignore the top item on its data stack....

Computations involving processes, will by necessity require cooperating contexts to perform useful but potentially infinite computations.

What does switching contexts *mean* categorically?

3.5.2 Context core definition code

CHeader : public

```
1  typedef Boolean (ExtendJoyLoL)(
2    JoyLoLInterp *jInterp,
3    DictObj      *aDict,
4    Symbol       *definedName,
5    AssertionObj *preCondition,
6    CFunction    *aFunc,
7    AssertionObj *postCondition,
8    Boolean       redefine
9  );
10
11 typedef Boolean (ExtendCtxJoyLoL)(
12   JoyLoLInterp *jInterp,
13   DictObj      *aDict,
14   Symbol       *definedName,
15   AssertionObj *preCondition,
16   CtxCFunction *aFunc,
17   AssertionObj *postCondition,
18   Boolean       redefine
19 );
20
21 #define extendJoyLoL(jInterp, aDict,      \
22   definedName, preStr, aFunc, postStr,    \
23   redefine)                               \
24   (                                       \
25     assert(getContextsClass(jInterp)      \
26       ->extendJoyLoLFunc),                \
27     (getContextsClass(jInterp)            \
28       ->extendJoyLoLFunc(jInterp, aDict,  \
29         definedName,                      \
30         parseAssertion(jInterp, preStr),  \
31         aFunc,                            \
32         parseAssertion(jInterp, postStr), \
33         redefine                          \
34       )                                   \
35     )                                     \
36   )
```

```
37  #define extendJoyLoLObj(jInterp, aDict,                    \
38    definedName, preCond, aFunc, postCond, redefine)         \
39    (                                                        \
40      assert(getContextsClass(jInterp)                       \
41        ->extendJoyLoLFunc),                                 \
42      (getContextsClass(jInterp)                             \
43        ->extendJoyLoLFunc(jInterp, aDict,                   \
44          definedName, preCond, aFunc, postCond, redefine)) \
45    )
46  #define extendJoyLoLIn(jInterp, aDict, definedName, \
47    preStr, aFunc, postStr)                            \
48    extendJoyLoL(jInterp, aDict, definedName,          \
49      preStr, aFunc, postStr, FALSE                    \
50    )

51  #define reExtendJoyLoLIn(jInterp, aDict, definedName, \
52    preStr, aFunc, postStr)                             \
53    extendJoyLoL(jInterp, aDict, definedName,           \
54      preStr, aFunc, postStr, TRUE                      \
55    )

56  #define extendJoyLoLObjIn(jInterp, aDict, definedName,  \
57    preCond, aFunc, postCond)                              \
58    extendJoyLoLObj(jInterp, aDict, definedName,          \
59      preCond, aFunc, postCond, FALSE                     \
60    )

61  #define reExtendJoyLoLObjIn(jInterp, aDict, definedName,  \
62    preCond, aFunc, postCond)                                \
63    extendJoyLoLObj(jInterp, aDict, definedName,            \
64      preCond, aFunc, postCond, TRUE                        \
65    )

66  #define extendJoyLoLInRoot(jInterp, definedName,  \
67    preStr, aFunc, postStr)                         \
68    (                                               \
69      assert(jInterp),                              \
70      assert(jInterp->rootCtx),                     \
71      extendJoyLoL(jInterp, jInterp->rootCtx->dict, \
72        definedName, preStr, aFunc, postStr, FALSE) \
73    )
```

```
74  #define reExtendJoyLoLInRoot(jInterp, definedName,  \
75    preStr, aFunc, postStr)                           \
76    (                                                 \
77      assert(jInterp),                                \
78      assert(jInterp->rootCtx),                       \
79      extendJoyLoL(jInterp, jInterp->rootCtx->dict,   \
80        definedName, preStr, aFunc, postStr, TRUE)    \
81    )
82  #define extendJoyLoLObjInRoot(jInterp, definedName,   \
83    preCond, aFunc, postCond)                           \
84    (                                                   \
85      assert(jInterp),                                  \
86      assert(jInterp->rootCtx),                         \
87      extendJoyLoLObj(jInterp, jInterp->rootCtx->dict,  \
88        definedName, preCond, aFunc, postCond, FALSE)   \
89    )

90  #define reExtendJoyLoLObjInRoot(jInterp, definedName, \
91    preCond, aFunc, postCond)                           \
92    (                                                   \
93      assert(jInterp),                                  \
94      assert(jInterp->rootCtx),                         \
95      extendJoyLoLObj(jInterp, jInterp->rootCtx->dict,  \
96        definedName, preCond, aFunc, postCond, TRUE)    \
97    )
98  #define extendCtxJoyLoL(jInterp, aDict,     \
99    definedName, preStr, aFunc, postStr,      \
100   redefine)                                 \
101   (                                         \
102     assert(getContextsClass(jInterp)        \
103       ->extendCtxJoyLoLFunc),               \
104     (getContextsClass(jInterp)              \
105       ->extendCtxJoyLoLFunc(jInterp, aDict, \
106         definedName,                        \
107         parseAssertion(jInterp, preStr),    \
108         aFunc,                              \
109         parseAssertion(jInterp, preStr),    \
110         redefine                            \
111       )                                     \
112     )                                       \
113   )

114 #define extendCtxJoyLoLObj(jInterp, aDict,          \
```

```
115   definedName, preCond, aFunc, postCond, redefine)  \
116   (                                                  \
117     assert(getContextsClass(jInterp)                 \
118       ->extendCtxJoyLoLFunc),                        \
119     (getContextsClass(jInterp)                       \
120       ->extendCtxJoyLoLFunc(jInterp, aDict,          \
121         definedName, preCond, aFunc, postCond,       \
122         redefine                                     \
123       )                                              \
124     )                                                \
125   )

126 #define extendCtxJoyLoLIn(jInterp, aDict, definedName,  \
127   preStr, aFunc, postStr)                               \
128   extendCtxJoyLoL(jInterp, aDict, definedName,          \
129     preStr, aFunc, postStr, FALSE                       \
130   )

131 #define reExtendCtxJoyLoLIn(jInterp, aDict, definedName,  \
132   preStr, aFunc, postStr)                                 \
133   extendCtxJoyLoL(jInterp, aDict, definedName,            \
134     preStr, aFunc, postStr, TRUE                          \
135   )

136 #define extendCtxJoyLoLObjIn(jInterp, aDict, definedName, \
137   preCond, aFunc, postCond)                               \
138   extendCtxJoyLoLObj(jInterp, aDict, definedName,         \
139     preCond, aFunc, postCond, FALSE                       \
140   )

141 #define reExtendCtxJoyLoLObjIn(jInterp, aDict, definedName, \
142   preCond, aFunc, postCond)                                   \
143   extendCtxJoyLoL(jInterp, aDict, definedName,                \
144     preCond, aFunc, postCond, TRUE                            \
145   )

146 #define extendCtxJoyLoLInRoot(jInterp, definedName,   \
147   preStr, aFunc, postStr)                             \
148   (                                                   \
149     assert(jInterp),                                  \
150     assert(jInterp->rootCtx),                         \
151     extendCtxJoyLoL(jInterp, jInterp->rootCtx->dict,  \
152       definedName, preStr, aFunc, postStr, FALSE)     \
```

```
153   )

154 #define reExtendCtxJoyLoLInRoot(jInterp, definedName, \
155   preStr, aFunc, postStr)                              \
156   (                                                    \
157     assert(jInterp),                                   \
158     assert(jInterp->rootCtx),                          \
159     extendCtxJoyLoL(jInterp, jInterp->rootCtx->dict,   \
160       definedName, preStr, aFunc, postStr, TRUE        \
161     )                                                  \
162   )
163 #define extendCtxJoyLoLObjInRoot(jInterp, definedName,  \
164   preCond, aFunc, postCond)                              \
165   (                                                      \
166     assert(jInterp),                                     \
167     assert(jInterp->rootCtx),                            \
168     extendCtxJoyLoLObj(jInterp, jInterp->rootCtx->dict, \
169       definedName, preCond, aFunc, postCond, FALSE       \
170     )                                                    \
171   )

172 #define reExtendCtxJoyLoLObjInRoot(jInterp, definedName,  \
173   preCond, aFunc, postCond)                                \
174   (                                                        \
175     assert(jInterp),                                       \
176     assert(jInterp->rootCtx),                              \
177     extendCtxJoyLoL(jInterp, jInterp->rootCtx->dict,       \
178       definedName, preCond, aFunc, postCond, TRUE          \
179     )                                                      \
180   )
```

CHeader : private

```
1  extern Boolean extendJoyLoLImpl(
2    JoyLoLInterp *jInterp,
3    DictObj      *aDict,
4    Symbol       *definedName,
5    AssertionObj *preCondition,
6    CFunction    *aFunc,
7    AssertionObj *postCondition,
8    Boolean       redefine
9  );
10
11 extern Boolean extendCtxJoyLoLImpl(
```

```
12    JoyLoLInterp *jInterp,
13    DictObj      *aDict,
14    Symbol       *definedName,
15    AssertionObj *preCondition,
16    CtxCFunction *aFunc,
17    AssertionObj *postCondition,
18    Boolean       redefine
19  );
```

CCode : default

```
1   Boolean extendJoyLoLImpl(
2     JoyLoLInterp *jInterp,
3     DictObj      *aDict,
4     Symbol       *definedName,
5     AssertionObj *preCondition,
6     CFunction    *aFunc,
7     AssertionObj *postCondition,
8     Boolean       redefine
9   ) {
10    assert(aDict);
11    assert(preCondition);
12    assert(postCondition);
13
14    DictNodeObj* aSym = getSymbolEntryInChild(aDict, definedName);
15    assert(aSym);
16    if (!aSym->value || redefine) {
17      aSym->value =
18        (JObj*)newCFunction(jInterp, aFunc);
19    }
20
21    return TRUE;
22  }
23
24  Boolean extendCtxJoyLoLImpl(
25    JoyLoLInterp *jInterp,
26    DictObj      *aDict,
27    Symbol       *definedName,
28    AssertionObj *preCondition,
29    CtxCFunction *aFunc,
30    AssertionObj *postCondition,
31    Boolean       redefine
32  ) {
33    assert(aDict);
```

```
34    assert(preCondition);
35    assert(postCondition);
36
37    DictNodeObj* aSym = getSymbolEntryInChild(aDict, definedName);
38    assert(aSym);
39    if (!aSym->value || redefine) {
40      aSym->value =
41        (JObj*)newCtxCFunction(jInterp, aFunc);
42    }
43
44    return TRUE;
45  }
```

CHeader : public

```
1   typedef Boolean (DefineJoyLoL)(
2     JoyLoLInterp *jInterp,
3     DictObj      *aDict,
4     Symbol       *definedName,
5     AssertionObj *preCondition,
6     JObj         *aLoL,
7     AssertionObj *postCondition,
8     Boolean       redefine
9   );
10
11  #define defineJoyLoL(jInterp, aDict, definedName,     \
12    preCondition, aLoL, postCondition, redefine)        \
13    (                                                   \
14      assert(getContextsClass(jInterp)                  \
15        ->defineJoyLoLFunc),                            \
16      (getContextsClass(jInterp)                        \
17        ->defineJoyLoLFunc(jInterp, aDict, definedName, \
18          preCondition, aLoL, postCondition, redefine   \
19        )                                               \
20      )                                                 \
21    )
22  #define defineJoyLoLIn(jInterp, aDict, definedName, \
23    preCondition, aLoL, postCondition)                \
24    defineJoyLoL(jInterp, aDict, definedName,         \
25      preCondition, aLoL, postCondition, FALSE        \
26    )
27  #define reDefineJoyLoLIn(jInterp, aDict, definedName, \
28    preCondition, aLoL, postCondition)                  \
29    defineJoyLoL(jInterp, aDict, definedName,           \
```

```
    preCondition, aLoL, postCondition, TRUE           \
  )
#define defineJoyLoLInRoot(jInterp, definedName,               \
  preCondition, aLoL, postCondition)                           \
  (                                                            \
    assert(jInterp),                                           \
    assert(jInterp->rootCtx),                                  \
    defineJoyLoL(jInterp, jInterp->rootCtx->dict, definedName, \
      preCondition, aLoL, postCondition, FALSE                 \
    )                                                          \
  )
#define reDefineJoyLoLInRoot(jInterp, definedName,             \
  preCondition, aLoL, postCondition)                           \
  (                                                            \
    assert(jInterp),                                           \
    assert(jInterp->rootCtx),                                  \
    defineJoyLoL(jInterp, jInterp->rootCtx->dict, definedName, \
      preCondition, aLoL, postCondition, TRUE                  \
    )                                                          \
  )
```

CHeader : private

```
extern Boolean defineJoyLoLImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  AssertionObj *preCondition,
  JObj         *aLoL,
  AssertionObj *postCondition,
  Boolean       redefine
);
```

CCode : default

```
Boolean defineJoyLoLImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  AssertionObj *preCondition,
  JObj         *aLoL,
  AssertionObj *postCondition,
  Boolean       redefine
) {
```

Context core definition code 3.5.2

```
  assert(aDict);
  assert(preCondition);
  assert(postCondition);

  DictNodeObj* aSym = getSymbolEntryInChild(aDict, definedName);
  assert(aSym);
  if (!aSym->value || redefine) {
    aSym->value = copyLoL(jInterp, aLoL);
  }

  return TRUE;
}
```

CHeader : public

```
typedef Boolean (DefineContext)(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  ContextObj   *newCtx,
  Boolean       redefine
);

#define defineContext(jInterp, aDict,     \
  definedName, newCtx, redefine)          \
  (                                       \
    assert(getContextsClass(jInterp)      \
      ->defineContextFunc),               \
    (getContextsClass(jInterp)            \
      ->defineContextFunc(jInterp, aDict, \
        definedName, newCtx, redefine))   \
  )
#define defineContextIn(jInterp, aDict, definedName, newCtx)  \
  defineContext(jInterp, aDict, definedName, newCtx, FALSE)
#define reDefineContextIn(jInterp, aDict, definedName, newCtx)  \
  defineContext(jInterp, aDict, definedName, newCtx, TRUE)
#define defineContextInRoot(jInterp, definedName, newCtx) \
  (                                                       \
    assert(jInterp),                                      \
    assert(jInterp->rootCtx),                             \
    defineContext(jInterp, jInterp->rootCtx->dict,        \
      definedName, newCtx, FALSE)                         \
  )
#define reDefineContextInRoot(jInterp, definedName, newCtx) \
```

```
  (                                                                   \
    assert(jInterp),                                                  \
    assert(jInterp->rootCtx),                                         \
    defineContext(jInterp, jInterp->rootCtx->dict,                    \
      definedName, newCtx, TRUE)                                      \
  )
```

CHeader : private

```
extern Boolean defineContextImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  ContextObj   *newCtx,
  Boolean       redefine
);
```

CCode : default

```
Boolean defineContextImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  ContextObj   *newCtx,
  Boolean       redefine
) {
  DEBUG(jInterp, "defineContext %p %p [%s] %p %zu\n",
        jInterp, aDict, definedName, newCtx, redefine);

  assert(aDict);

  DictNodeObj* aSym = getSymbolEntryInChild(aDict, definedName);
  assert(aSym);
  if (!aSym->value || redefine) {
    aSym->value   = (JObj*)newCtx;
  }

  return TRUE;
}
```

CHeader : public

```
typedef Boolean (DefineNaming)(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
```

```
  Symbol       *definedName,
  DictObj      *newDict,
  Boolean       redefine
);

#define defineNaming(jInterp, aDict,      \
  definedName, newCtx, redefine)          \
  (                                       \
    assert(getContextsClass(jInterp)      \
      ->defineNamingFunc),                \
    (getContextsClass(jInterp)            \
      ->defineNamingFunc(jInterp, aDict,  \
        definedName, newDict, redefine))  \
  )
#define defineNamingIn(jInterp, aDict, definedName, newDict)  \
  defineNaming(jInterp, aDict, definedName, newDict, FALSE)
#define reDefineNamingIn(jInterp, aDict, definedName, newDict)  \
  defineNaming(jInterp, aDict, definedName, newDict, TRUE)
#define defineNamingInRoot(jInterp, definedName, newDict) \
  (                                                       \
    assert(jInterp),                                      \
    assert(jInterp->rootCtx),                             \
    defineNaming(jInterp, jInterp->rootCtx->dict,         \
      definedName, newDict, FALSE)                        \
  )
#define reDefineNamingInRoot(jInterp, definedName, newDict) \
  (                                                         \
    assert(jInterp),                                        \
    assert(jInterp->rootCtx),                               \
    defineNaming(jInterp, jInterp->rootCtx->dict,           \
      definedName, newDict, TRUE)                           \
  )
```

CHeader : private

```
extern Boolean defineNamingImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  DictObj      *newDict,
  Boolean       redefine
);
```

CCode : default

```
Boolean defineNamingImpl(
  JoyLoLInterp *jInterp,
  DictObj      *aDict,
  Symbol       *definedName,
  DictObj      *newDict,
  Boolean       redefine
) {
  DEBUG(jInterp, "defineNaming %p %p [%s] %p %zu\n",
        jInterp, aDict, definedName, newDict, redefine);

  assert(aDict);

  DictNodeObj* aSym = getSymbolEntryInChild(aDict, definedName);
  assert(aSym);
  if (!aSym->value || redefine) {
    aSym->value   = (JObj*)newDict;
  }

  return TRUE;
}
```

3.5.3 Context core data code

```
{{aCtx}}->data =
  newPair({{aCtx}}->jInterp, {{lolToPush}}, {{aCtx}}->data);
```

CHeader : public

```
typedef void (ClearCtx)(
  ContextObj *aCtx
);

#define clearCtxData(aCtx)                  \
  (                                         \
    assert(aCtx),                           \
    assert(getContextsClass(aCtx->jInterp)  \
      ->clearCtxDataFunc),                  \
    (getContextsClass(aCtx->jInterp)        \
      ->clearCtxDataFunc(aCtx))             \
  )
```

CHeader : private

Context core data code 3.5.3

```
extern void clearCtxDataImpl(
  ContextObj *aCtx
);
```

CCode : default

```
void clearCtxDataImpl(
  ContextObj *aCtx
) {
  assert(aCtx);
  aCtx->data = NULL;
}
```

CHeader : public

```
typedef void (PushCtx)(
  ContextObj *aCtx,
  JObj       *lolToPush
);

#define pushCtxData(aCtx, lolToPush)         \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->pushCtxDataFunc),                    \
    (getContextsClass(aCtx->jInterp)         \
      ->pushCtxDataFunc(aCtx, lolToPush))    \
  )
#define pushOnTopCtxData(aCtx, lolToPush)        \
  (                                              \
    assert(aCtx),                                \
    assert(getContextsClass(aCtx->jInterp)       \
      ->pushOnTopCtxDataFunc),                   \
    (getContextsClass(aCtx->jInterp)             \
      ->pushOnTopCtxDataFunc(aCtx, lolToPush))   \
  )
```

CHeader : private

```
extern void pushCtxDataImpl(
  ContextObj *aCtx,
  JObj       *lolToPush
);

extern void pushOnTopCtxDataImpl(
```

```
  ContextObj *aCtx,
  JObj       *lolToPush
);
```

CCode : default

```
void pushCtxDataImpl(
  ContextObj *aCtx,
  JObj       *lolToPush
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->data = newPair(aCtx->jInterp, lolToPush, aCtx->data);
}

void pushOnTopCtxDataImpl(
  ContextObj *aCtx,
  JObj       *lolToPush
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  asCar(aCtx->data) =
    newPair(aCtx->jInterp, lolToPush, asCar(aCtx->data));
}
```

CHeader : public

```
typedef void (PushNullCtx)(
  ContextObj *aCtx
);

#define pushNullCtxData(aCtx)                \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->pushNullCtxDataFunc),                \
    (getContextsClass(aCtx->jInterp)         \
      ->pushNullCtxDataFunc(aCtx))           \
  )
```

CHeader : private

```
extern void pushNullCtxDataImpl(
  ContextObj* aCtx
```

```
3  );
```

CCode : default

```
1  void pushNullCtxDataImpl(
2    ContextObj* aCtx
3  ) {
4    assert(aCtx);
5    assert(aCtx->jInterp);
6    aCtx->data = newPair(aCtx->jInterp, NULL, aCtx->data);
7  }
```

CHeader : public

```
1  typedef void (PushBooleanCtx)(
2    ContextObj *aCtx,
3    Boolean     aBool
4  );
5
6  #define pushBooleanCtxData(aCtx, aBool)      \
7    (                                          \
8      assert(aCtx),                            \
9      assert(getContextsClass(aCtx->jInterp)   \
10       ->pushBooleanCtxDataFunc),             \
11     (getContextsClass(aCtx->jInterp)         \
12       ->pushBooleanCtxDataFunc(aCtx, aBool)) \
13   )
```

CHeader : private

```
1  extern void pushBooleanCtxDataImpl(
2    ContextObj *aCtx,
3    Boolean     aBool
4  );
```

CCode : default

```
1  void pushBooleanCtxDataImpl(
2    ContextObj *aCtx,
3    Boolean     aBool
4  ) {
5    assert(aCtx);
6    assert(aCtx->jInterp);
7
8    JObj* aBoolPA = newBoolean(aCtx->jInterp, aBool);
9    if (!aBoolPA) return;
```

```
  aCtx->data = newPair(aCtx->jInterp, aBoolPA, aCtx->data);
}
```

CHeader : public

```
typedef void (PushNaturalCtx)(
  ContextObj *aCtx,
  size_t      anNatural
);

#define pushNaturalCtxData(aCtx, aNatural)      \
  (                                             \
    assert(aCtx),                               \
    assert(getContextsClass(aCtx->jInterp)      \
      ->pushNaturalCtxDataFunc),                \
    (getContextsClass(aCtx->jInterp)            \
      ->pushNaturalCtxDataFunc(aCtx, aNatural)) \
  )
```

CHeader : private

```
extern void pushNaturalCtxDataImpl(
  ContextObj *aCtx,
  size_t      aNatural
);
```

CCode : default

```
void pushNaturalCtxDataImpl(
  ContextObj *aCtx,
  size_t      aNatural
) {
  assert(aCtx);
  assert(aCtx->jInterp);

  char aNaturalStr[100];
  aNaturalStr[0] = 0;
  snprintf(aNaturalStr, 90, "%zu", aNatural);

  JObj* aNaturalPA = newNatural(aCtx->jInterp, aNaturalStr);
  if (!aNaturalPA) return;
  aCtx->data = newPair(aCtx->jInterp, aNaturalPA, aCtx->data);
}
```

CHeader : public

Context core data code 3.5.3

```
typedef void (PushSymbolCtx)(
  ContextObj *aCtx,
  Symbol     *aSymbol
);

#define pushSymbolCtxData(aCtx, aSymbol)      \
  (                                           \
    assert(aCtx),                             \
    assert(getContextsClass(aCtx->jInterp)    \
      ->pushSymbolCtxDataFunc),               \
    (getContextsClass(aCtx->jInterp)          \
      ->pushSymbolCtxDataFunc(aCtx, aSymbol)) \
  )
```

CHeader : private

```
extern void pushSymbolCtxDataImpl(
  ContextObj* aCtx,
  Symbol* aSymbol
);
```

CCode : default

```
void pushSymbolCtxDataImpl(
  ContextObj* aCtx,
  Symbol* aSymbol
) {
  assert(aCtx);
  if (!aSymbol) return;
  assert(aCtx->jInterp);

  JObj* aSymbolPA = NULL;
  if (strchr(aSymbol, '.')) {
    aSymbolPA =
      newSymbol(aCtx->jInterp, aSymbol, "pushSymbol", 0);
  } else {
    aSymbolPA = getAsSymbol(aCtx->dict, aSymbol, "pushSymbol", 0);
  }
  if (!aSymbolPA) return;
  aCtx->data = newPair(aCtx->jInterp, aSymbolPA, aCtx->data);
}
```

CHeader : public

```
typedef void (PushParsedArrayOfStringsCtx)(
```

```
  ContextObj *aCtx,
  Symbol     *someStrings[]
);

#define pushParsedArrayOfStringsCtxData(      \
  aCtx, someStrings)                          \
  (                                           \
    assert(aCtx),                             \
    assert(getContextsClass(aCtx->jInterp)    \
      ->pushParsedArrayOfStringsCtxDataFunc), \
    (getContextsClass(aCtx->jInterp)          \
      ->pushParsedArrayOfStringsCtxDataFunc(  \
        aCtx, someStrings))                   \
  )
```

CHeader : private

```
extern void pushParsedArrayOfStringsCtxDataImpl(
  ContextObj* aCtx,
  Symbol* someStrings[]
);
```

CCode : default

```
void pushParsedArrayOfStringsCtxDataImpl(
  ContextObj* aCtx,
  Symbol* someStrings[]
) {
  assert(aCtx);
  if (!someStrings) return;
  TextObj* aText =
    createTextFromArrayOfStrings(aCtx->jInterp, someStrings);
  pushParsedTextCtxData(aCtx, aText);
  freeText(aText);
}
```

CHeader : public

```
typedef void (PushParsedStringCtx)(
  ContextObj *aCtx,
  Symbol     *aString
);

#define pushParsedStringCtxData(aCtx, aString)      \
  (                                                 \
```

```
8     assert(aCtx),                                        \
9     assert(getContextsClass(aCtx->jInterp)               \
10      ->pushParsedStringCtxDataFunc),                    \
11    (getContextsClass(aCtx->jInterp)                     \
12      ->pushParsedStringCtxDataFunc(aCtx, aString)) \
13  )
```

CHeader : private

```
1  extern void pushParsedStringCtxDataImpl(
2    ContextObj* aCtx,
3    Symbol* aString
4  );
```

CCode : default

```
1  void pushParsedStringCtxDataImpl(
2    ContextObj* aCtx,
3    Symbol* aString
4  ) {
5    assert(aCtx);
6    if (!aString) return;
7    TextObj* aText =
8      createTextFromString(aCtx->jInterp, aString);
9    pushParsedTextCtxData(aCtx, aText);
10   freeText(aText);
11 }
```

CHeader : public

```
1  typedef void (PushParsedTextCtx)(
2    ContextObj *aCtx,
3    TextObj    *aTex
4  );
5
6  #define pushParsedTextCtxData(aCtx, aText)       \
7    (                                              \
8      assert(aCtx),                                \
9      assert(getContextsClass(aCtx->jInterp)       \
10       ->pushParsedTextCtxDataFunc),              \
11     (getContextsClass(aCtx->jInterp)             \
12       ->pushParsedTextCtxDataFunc(aCtx, aText)) \
13   )
```

CHeader : private

```
extern void pushParsedTextCtxDataImpl(
  ContextObj* aCtx,
  TextObj* aText
);
```

CCode : default

```
void pushParsedTextCtxDataImpl(
  ContextObj* aCtx,
  TextObj* aText
) {
  assert(aCtx);
  if (!aText) return;
  assert(aCtx->jInterp);
  JObj* aLoL = parseAllSymbols(aText);
  if (!aLoL) return;
  aCtx->data = newPair(aCtx->jInterp, aLoL, aCtx->data);
}
```

CHeader : public

```
typedef void (PrependListCtx)(
  ContextObj *aCtx,
  JObj   *lolToPrepend
);

#define prependListCtxData(aCtx, lolToPrepend)      \
  (                                                 \
    assert(aCtx),                                   \
    assert(getContextsClass(aCtx->jInterp)          \
      ->prependListCtxDataFunc),                    \
    (getContextsClass(aCtx->jInterp)                \
      ->prependListCtxDataFunc(aCtx, lolToPrepend)) \
  )
```

CHeader : private

```
extern void prependListCtxDataImpl(
  ContextObj* aCtx,
  JObj* lolToPrepend
);
```

CCode : default

```
void prependListCtxDataImpl(
  ContextObj* aCtx,
```

Context core data code 3.5.3

```
  JObj* lolToPrepend
) {
  assert(aCtx);
  if (!lolToPrepend) return;

  aCtx->data =
    concatLists(aCtx->jInterp, lolToPrepend, aCtx->data);
}
```

CHeader : public

```
typedef JObj *(PeekCtx)(
  ContextObj *aCtx
);

#define peekCtxData(aCtx)                    \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->peekCtxDataFunc),                    \
    (getContextsClass(aCtx->jInterp)         \
      ->peekCtxDataFunc(aCtx))               \
  )
#define peekCtxDataInto(aCtx, aVar)                    \
assert(aCtx);                                          \
JObj* aVar = peekCtxData(aCtx);                        \
if (aCtx->tracingOn) {                                 \
  JoyLoLInterp *jInterp = aCtx->jInterp;               \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);    \
  strBufPrintf(aStrBuf, "%s = ", #aVar);               \
  printLoL(aStrBuf, aVar);                             \
  strBufPrintf(aStrBuf, " (peek)\n");                  \
  jInterp->writeStdOut(jInterp, getCString(aStrBuf)); \
  strBufClose(aStrBuf);                                \
}
#define peekCtxDataIntoImpl(aCtx, aVar)                \
assert(aCtx);                                          \
JObj* aVar = peekCtxDataImpl(aCtx);                    \
if (aCtx->tracingOn) {                                 \
  JoyLoLInterp *jInterp = aCtx->jInterp;               \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);    \
  strBufPrintf(aStrBuf, "%s = ", #aVar);               \
  printLoL(aStrBuf, aVar);                             \
  strBufPrintf(aStrBuf, " (peek)\n");                  \
```

```
  jInterp->writeStdOut(jInterp, getCString(aStrBuf)); \
  strBufClose(aStrBuf);                               \
}
```

CHeader : private

```
extern JObj* peekCtxDataImpl(
  ContextObj* aCtx
);
```

CCode : default

```
JObj* peekCtxDataImpl(
  ContextObj* aCtx
) {
  assert(aCtx);
  if (!aCtx->data) return NULL;

  DEBUG(aCtx->jInterp, "peekCtxData: %p %p %zu\n",
    aCtx->data, asType(aCtx->data), (size_t)asTag(aCtx->data)
  );
  assert(isPair(aCtx->data));

  JObj* peekedLoL = asCar(aCtx->data);
  return peekedLoL;
}
```

CHeader : public

```
typedef JObj *(PopCtx)(
  ContextObj *aCtx
);

#define popCtxData(aCtx)                    \
  (                                         \
    assert(aCtx),                           \
    assert(getContextsClass(aCtx->jInterp)  \
      ->popCtxDataFunc),                    \
    (getContextsClass(aCtx->jInterp)        \
      ->popCtxDataFunc(aCtx))               \
  )
#define popCtxDataInto(aCtx, aVar)                  \
assert(aCtx);                                       \
JObj* aVar = popCtxData(aCtx);                      \
if (aCtx->tracingOn) {                              \
```

```
  JoyLoLInterp *jInterp = aCtx->jInterp;                \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);     \
  strBufPrintf(aStrBuf, "%s = ", #aVar);                \
  printLoL(aStrBuf, aVar);                              \
  strBufPrintf(aStrBuf, "\n");                          \
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));   \
  strBufClose(aStrBuf);                                 \
}
#define popCtxDataIntoImpl(aCtx, aVar)                  \
assert(aCtx);                                           \
JObj* aVar = popCtxDataImpl(aCtx);                      \
if (aCtx->tracingOn) {                                  \
  JoyLoLInterp *jInterp = aCtx->jInterp;                \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);     \
  strBufPrintf(aStrBuf, "%s = ", #aVar);                \
  printLoL(aStrBuf, aVar);                              \
  strBufPrintf(aStrBuf, "\n");                          \
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));   \
  strBufClose(aStrBuf);                                 \
}
```

CHeader : private

```
extern JObj* popCtxDataImpl(
  ContextObj* aCtx
);
```

CCode : default

```
JObj* popCtxDataImpl(
  ContextObj* aCtx
) {
  assert(aCtx);
  if (!aCtx->data) return NULL;

  DEBUG(aCtx->jInterp, "popCtxData: %p %p %zu\n",
    aCtx->data, asType(aCtx->data), (size_t)asTag(aCtx->data)
  );
  assert(isPair(aCtx->data));

  JObj* poppedLoL = asCar(aCtx->data);
  aCtx->data      = asCdr(aCtx->data);
  return poppedLoL;
}
```

CHeader : public

```
#define showCtxStack(aCtx, aStack, stackLabel, aStrBuf)       \
  {                                                           \
    size_t index = 1;                                         \
    JObj* curStack = aStack;                                  \
    strBufPrintf(aStrBuf, "----\n");                          \
    while(isPair(curStack)) {                                 \
      if (aCtx->showDepth < index) break;                     \
      strBufPrintf(aStrBuf, "%s[%zu]: ", stackLabel, index);  \
      printLoL(aStrBuf, asCar(curStack));                     \
      strBufPrintf(aStrBuf, "\n");                            \
      curStack = asCdr(curStack);                             \
      index++;                                                \
    }                                                         \
    strBufPrintf(aStrBuf, "%s[*]: ", stackLabel);             \
    printLoL(aStrBuf, curStack);                              \
    strBufPrintf(aStrBuf, "\n");                              \
  }
#define showCtxData(aCtx, aStrBuf)            \
  assert(aCtx);                               \
  showCtxStack(aCtx, aCtx->data, "d", aStrBuf)
```

3.5.4 Context core process code

CHeader : public

```
#define clearCtxProcess(aCtx)                \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->clearCtxProcessFunc),                \
    (getContextsClass(aCtx->jInterp)         \
      ->clearCtxProcessFunc(aCtx))           \
  )
```

CHeader : private

```
extern void clearCtxProcessImpl(
  ContextObj *aCtx
);
```

CCode : default

```
void clearCtxProcessImpl(
```

```
  ContextObj *aCtx
) {
  assert(aCtx);
  aCtx->process = NULL;
}
```

CHeader : public

```
#define pushCtxProcess(aCtx, lolToPush)        \
  (                                            \
    assert(aCtx),                              \
    assert(getContextsClass(aCtx->jInterp)     \
      ->pushCtxProcessFunc),                   \
    (getContextsClass(aCtx->jInterp)           \
      ->pushCtxProcessFunc(aCtx, lolToPush))   \
  )
```

CHeader : private

```
extern void pushCtxProcessImpl(
  ContextObj* aCtx,
  JObj* lolToPush
);
```

CCode : default

```
void pushCtxProcessImpl(
  ContextObj* aCtx,
  JObj* lolToPush
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->process = newPair(aCtx->jInterp, lolToPush, aCtx->process);
}
```

CHeader : public

```
#define pushNullCtxProcess(aCtx)             \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->pushNullCtxProcessFunc),             \
    (getContextsClass(aCtx->jInterp)         \
      ->pushNullCtxProcessFunc(aCtx))        \
  )
```

CHeader : private

```
extern void pushNullCtxProcessImpl(
  ContextObj* aCtx
);
```

CCode : default

```
void pushNullCtxProcessImpl(
  ContextObj* aCtx
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->process = newPair(aCtx->jInterp, NULL, aCtx->process);
}
```

CHeader : public

```
#define pushBooleanCtxProcess(aCtx, aBool)       \
  (                                              \
    assert(aCtx),                                \
    assert(getContextsClass(aCtx->jInterp)       \
      ->pushBooleanCtxProcessFunc),              \
    (getContextsClass(aCtx->jInterp)             \
      ->pushBooleanCtxProcessFunc(aCtx, aBool)) \
  )
```

CHeader : private

```
extern void pushBooleanCtxProcessImpl(
  ContextObj *aCtx,
  Boolean     aBool
);
```

CCode : default

```
void pushBooleanCtxProcessImpl(
  ContextObj *aCtx,
  Boolean     aBool
) {
  assert(aCtx);
  assert(aCtx->jInterp);

  JObj* aBoolPA = newBoolean(aCtx->jInterp, aBool);
  if (!aBoolPA) return;
  aCtx->process = newPair(aCtx->jInterp, aBoolPA, aCtx->process);
```

```
}
```

CHeader : public

```
#define pushNaturalCtxProcess(aCtx, aNatural)       \
  (                                                 \
    assert(aCtx),                                   \
    assert(getContextsClass(aCtx->jInterp)          \
      ->pushNaturalCtxProcessFunc),                 \
    (getContextsClass(aCtx->jInterp)                \
      ->pushNaturalCtxProcessFunc(aCtx, aNatural)) \
  )
```

CHeader : private

```
extern void pushNaturalCtxProcessImpl(
  ContextObj *aCtx,
  size_t      aNatural
);
```

CCode : default

```
void pushNaturalCtxProcessImpl(
  ContextObj *aCtx,
  size_t      aNatural
) {
  assert(aCtx);
  assert(aCtx->jInterp);

  char aNaturalStr[100];
  aNaturalStr[0] = 0;
  snprintf(aNaturalStr, 90, "%zu", aNatural);

  JObj* aNaturalPA = newNatural(aCtx->jInterp, aNaturalStr);
  if (!aNaturalPA) return;
  aCtx->process = newPair(aCtx->jInterp, aNaturalPA, aCtx->process);
}
```

CHeader : public

```
#define pushSymbolCtxProcess(aCtx, aSymbol)       \
  (                                               \
    assert(aCtx),                                 \
    assert(getContextsClass(aCtx->jInterp)        \
      ->pushSymbolCtxProcessFunc),                \
    (getContextsClass(aCtx->jInterp)              \
```

```
      ->pushSymbolCtxProcessFunc(aCtx, aSymbol)) \
  )
```

CHeader : private

```
extern void pushSymbolCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* aSymbol
);
```

CCode : default

```
void pushSymbolCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* aSymbol
) {
  assert(aCtx);
  if (!aSymbol) return;
  assert(aCtx->jInterp);

  JObj* aSymbolPA = getAsSymbol(aCtx->dict, aSymbol, "pushSymbol", 0);
  if (!aSymbolPA) return;
  aCtx->process = newPair(aCtx->jInterp, aSymbolPA, aCtx->process);
}
```

CHeader : public

```
#define pushParsedArrayOfStringsCtxProcess(      \
  aCtx, someStrings)                             \
  (                                              \
    assert(aCtx),                                \
    assert(getContextsClass(aCtx->jInterp)       \
      ->pushParsedArrayOfStringsCtxProcessFunc), \
    (getContextsClass(aCtx->jInterp)             \
      ->pushParsedArrayOfStringsCtxProcessFunc(  \
        aCtx, someStrings))                      \
  )
```

CHeader : private

```
extern void pushParsedArrayOfStringsCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* someStrings[]
);
```

CCode : default

Context core process code 3.5.4

```c
void pushParsedArrayOfStringsCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* someStrings[]
) {
  assert(aCtx);
  if (!someStrings) return;
  TextObj* aText =
    createTextFromArrayOfStrings(aCtx->jInterp, someStrings);
  pushParsedTextCtxProcess(aCtx, aText);
  freeText(aText);
}
```

CHeader : public

```c
#define pushParsedStringCtxProcess(aCtx, aString)       \
  (                                                     \
    assert(aCtx),                                       \
    assert(getContextsClass(aCtx->jInterp)              \
      ->pushParsedStringCtxProcessFunc),                \
    (getContextsClass(aCtx->jInterp)                    \
      ->pushParsedStringCtxProcessFunc(aCtx, aString)) \
  )
```

CHeader : private

```c
extern void pushParsedStringCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* aString
);
```

CCode : default

```c
void pushParsedStringCtxProcessImpl(
  ContextObj* aCtx,
  Symbol* aString
) {
  assert(aCtx);
  if (!aString) return;
  TextObj* aText =
    createTextFromString(aCtx->jInterp, aString);
  pushParsedTextCtxProcess(aCtx, aText);
  freeText(aText);
}
```

CHeader : public

```
#define pushParsedTextCtxProcess(aCtx, aText)       \
  (                                                 \
    assert(aCtx),                                   \
    assert(getContextsClass(aCtx->jInterp)          \
      ->pushParsedTextCtxProcessFunc),              \
    (getContextsClass(aCtx->jInterp)                \
      ->pushParsedTextCtxProcessFunc(aCtx, aText)) \
  )
```

CHeader : private

```
extern void pushParsedTextCtxProcessImpl(
  ContextObj* aCtx,
  TextObj* aText
);
```

CCode : default

```
void pushParsedTextCtxProcessImpl(
  ContextObj* aCtx,
  TextObj* aText
) {
  assert(aCtx);
  if (!aText) return;
  assert(aCtx->jInterp);
  JObj* aLoL = parseAllSymbols(aText);
  if (!aLoL) return;
  aCtx->process = newPair(aCtx->jInterp, aLoL, aCtx->process);
}
```

CHeader : public

```
#define prependListCtxProcess(aCtx, lolToPrepend)       \
  (                                                     \
    assert(aCtx),                                       \
    assert(getContextsClass(aCtx->jInterp)              \
      ->prependListCtxProcessFunc),                     \
    (getContextsClass(aCtx->jInterp)                    \
      ->prependListCtxProcessFunc(aCtx, lolToPrepend)) \
  )
```

CHeader : private

```
extern void prependListCtxProcessImpl(
  ContextObj* aCtx,
  JObj* lolToPrepend
```

```
);
```

CCode : default

```
void prependListCtxProcessImpl(
  ContextObj* aCtx,
  JObj* lolToPrepend
) {
  assert(aCtx);
  if (!lolToPrepend) return;

  aCtx->process =
    concatLists(aCtx->jInterp, lolToPrepend, aCtx->process);
}
```

CHeader : public

```
#define peekCtxProcess(aCtx)                    \
  (                                             \
    assert(aCtx),                               \
    assert(getContextsClass(aCtx->jInterp)      \
      ->peekCtxProcessFunc),                    \
    (getContextsClass(aCtx->jInterp)            \
      ->peekCtxProcessFunc(aCtx))               \
  )
#define peekCtxProcessInto(aCtx, aVar)                  \
assert(aCtx);                                           \
JObj* aVar = peekCtxProcess(aCtx);                      \
if (aCtx->tracingOn) {                                  \
  JoyLoLInterp *jInterp = aCtx->jInterp;                \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);     \
  strBufPrintf(aStrBuf, "%s = ", #aVar);                \
  printLoL(aStrBuf, aVar);                              \
  strBufPrintf(aStrBuf, " (peek)\n");                   \
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));   \
  strBufClose(aStrBuf);                                 \
}
#define peekCtxProcessIntoImpl(aCtx, aVar)              \
assert(aCtx);                                           \
JObj* aVar = peekCtxProcessImpl(aCtx);                  \
if (aCtx->tracingOn) {                                  \
  JoyLoLInterp *jInterp = aCtx->jInterp;                \
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);     \
  strBufPrintf(aStrBuf, "%s = ", #aVar);                \
```

```
28    printLoL(aStrBuf, aVar);                                  \
29    strBufPrintf(aStrBuf, " (peek)\n");                       \
30    jInterp->writeStdOut(jInterp, getCString(aStrBuf));       \
31    strBufClose(aStrBuf);                                     \
32  }
```

CHeader : private

```
1   extern JObj* peekCtxProcessImpl(
2     ContextObj* aCtx
3   );
```

CCode : default

```
1   JObj* peekCtxProcessImpl(
2     ContextObj* aCtx
3   ) {
4     assert(aCtx);
5     if (!aCtx->process) return NULL;
6
7     DEBUG(aCtx->jInterp, "peekCtxProcess: %p %p %zu\n",
8       aCtx->process, asType(aCtx->process), (size_t)asTag(aCtx->process)
9     );
10    assert(isPair(aCtx->process));

11    JObj* peekedLoL = asCar(aCtx->process);
12    return peekedLoL;
13  }
```

CHeader : public

```
1   #define popCtxProcess(aCtx)                       \
2     (                                               \
3       assert(aCtx),                                 \
4       assert(getContextsClass(aCtx->jInterp)        \
5         ->popCtxProcessFunc),                       \
6       (getContextsClass(aCtx->jInterp)              \
7         ->popCtxProcessFunc(aCtx))                  \
8     )
9   #define popCtxProcessInto(aCtx, aVar)             \
10  assert(aCtx);                                     \
11  JObj* aVar = popCtxProcess(aCtx);                 \
12  if (aCtx->tracingOn) {                            \
13    JoyLoLInterp *jInterp = aCtx->jInterp;          \
14    StringBufferObj *aStrBuf = newStringBuffer(aCtx);  \
```

```
15    strBufPrintf(aStrBuf, "%s = ", #aVar);                \
16    printLoL(aStrBuf, aVar);                              \
17    strBufPrintf(aStrBuf, "\n");                          \
18    jInterp->writeStdOut(jInterp, getCString(aStrBuf));   \
19    strBufClose(aStrBuf);                                 \
20  }
21  #define popCtxProcessIntoImpl(aCtx, aVar)               \
22  assert(aCtx);                                           \
23  JObj* aVar = popCtxProcessImpl(aCtx);                   \
24  if (aCtx->tracingOn) {                                  \
25    JoyLoLInterp *jInterp = aCtx->jInterp;                \
26    StringBufferObj *aStrBuf = newStringBuffer(aCtx);     \
27    strBufPrintf(aStrBuf, "%s = ", #aVar);                \
28    printLoL(aStrBuf, aVar);                              \
29    strBufPrintf(aStrBuf, "\n");                          \
30    jInterp->writeStdOut(jInterp, getCString(aStrBuf));   \
31    strBufClose(aStrBuf);                                 \
32  }
```

CHeader : private

```
1   extern JObj* popCtxProcessImpl(
2     ContextObj* aCtx
3   );
```

CCode : default

```
1   JObj* popCtxProcessImpl(
2     ContextObj* aCtx
3   ) {
4     assert(aCtx);
5     if (!aCtx->process) return NULL;
6
7     DEBUG(aCtx->jInterp, "popCtxProcess: %p %p %zu\n",
8       aCtx->process, asType(aCtx->process), (size_t)asTag(aCtx->process)
9     );
10    assert(isPair(aCtx->process));

11    JObj* poppedLoL = asCar(aCtx->process);
12    aCtx->process   = asCdr(aCtx->process);
13    return poppedLoL;
14  }
```

CHeader : public

```
#define showCtxProcess(aCtx, aStrBuf)                \
  assert(aCtx);                                      \
  showCtxStack(aCtx, aCtx->process, "p", aStrBuf)
```

3.5.5 Context control code

3.5.5.1 Operators

3.5.5.2 Combinators

CCode : default

```
static void pushListAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, aList);
  top = newPair(aCtx->jInterp, top, NULL);
  aList = concatLists(aCtx->jInterp, top, aList);
  pushCtxData(aCtx, aList);
}

static void popListAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  popCtxDataIntoImpl(aCtx, aList);
  popListInto(aCtx, aList, top);
  pushCtxData(aCtx, aList);
  pushCtxData(aCtx, top);
}

static void wrapAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  popCtxDataIntoImpl(aCtx, top);
  top = newPair(aCtx->jInterp, top, NULL);
  pushCtxData(aCtx, top);
}

static void prependAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
```

```
30    popCtxDataIntoImpl(aCtx, second);
31    JObj* result = concatLists(aCtx->jInterp, top, second);
32    pushCtxDataImpl(aCtx, result);
33  }
```

CCode : default

```
1   static void appendAP(ContextObj* aCtx) {
2     popCtxDataIntoImpl(aCtx, top);
3     popCtxDataIntoImpl(aCtx, second);
4     JObj* result = concatLists(aCtx->jInterp, second, top);
5     pushCtxDataImpl(aCtx, result);
6   }
```

CCode : default

```
1   static void extractAP(ContextObj* aCtx) {
2     popCtxDataIntoImpl(aCtx, top);
3     prependListCtxData(aCtx, top);
4   }
```

```
\startWord[extract]

\preDataStack
  (
    top : list
    dataStack
  )
\preProcessStack
  (
    processStack
  )
\preConditions
  (top isFinite)
\stopPreStack

\postDataStack
  (
    top (prepended)
    dataStack
  )
\postProcessStack
  (
    processStack
```

```
  )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void interpretAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  prependListCtxProcess(aCtx, top);
}
```

```
\startWord[interpret]

\preDataStack
  (
    top : list
    dataStack
  )
\preProcessStack
  (
    processStack
  )
\preConditions
  (top isFinite)
\stopPreStack

\postDataStack
  (
    dataStack
  )
\postProcessStack
  (
    top (prepended)
    processStack
  )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void ifteAP(ContextObj* aCtx) {
```

Context control code 3.5.5

```
  popCtxDataIntoImpl(aCtx, topElse);
  popCtxDataIntoImpl(aCtx, topThen);
  popCtxDataIntoImpl(aCtx, topIf);
  pushCtxProcessImpl(aCtx, topElse); // save for later
  pushCtxProcessImpl(aCtx, topThen); // save for later
  pushSymbolCtxProcess(aCtx, "ifteCont");
  prependListCtxProcess(aCtx, topIf); // execute the if condition
}
```

```
(
  (popData topElse)
  (popData topThen)
  (popData topIf)
  (pushProcess topElse)
  (pushProcess topThen)
  (pushSymbolProcess ifteCont)
  (prependProcess topIf)
)
```

CCode : default

```
static void ifteContAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataIntoImpl(aCtx, topIf);
  popCtxProcessIntoImpl(aCtx, topThen);
  popCtxProcessIntoImpl(aCtx, topElse);
  if (isTrue(topIf)) {
    prependListCtxProcess(aCtx, topThen);
  } else {
    prependListCtxProcess(aCtx, topElse);
  }
}
```

```
(
  (popData topIf)
  (popProcess topThen)
  (popProcess topElse)
  (doIfte topIf
    (prependProcess topThen)
    (prependProcess topElse)
  )
```

)

CCode : default

```c
static void forAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxProcessIntoImpl(aCtx, nextCommand);
  if (symbolIs(jInterp, nextCommand, "forDone")) {
    // this for loop is done
    DEBUG(jInterp, "forAP DONE%s\n", "");
  } else {
    DEBUG(jInterp, "forAP continue%s\n", "");
    pushSymbolCtxProcess(aCtx, "for");
    prependListCtxProcess(aCtx, nextCommand);
  }
}
```

```
\startWord[for]
\preDataStack
  (
    dataStack
  )
\preProcessStack
  (
    nextCommand : aType
    processStack
  )
\preConditions
  (nextCommand isFinite)  >> if nextCommand == 'forDone' ok
                             if nextCommand isList then nextCommand isFinite
                             else ok <<
\stopPreStack

\postDataStack
  (
    dataStack
  )
\postProcessStack
  (nextCommand 'forDone' =symbol) -> (
    processStack
  )
  OR
```

```
  (else) -> (
    nextCommand (prepended)
    'for' : Symbol
    processStack
  )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void forDoneAP(ContextObj* aCtx) {
  // ignore
}
```

```
\startWord[forDone]

\preDataStack
  ( dataStack )
\preProcessStack
  ( processStack )
\preConditions
\stopPreStack

\postDataStack
  ( dataStack )
\postProcessStack
  ( processStack )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void lispInterpretAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  if (!isPair(top)) {
    raiseExceptionMsg(aCtx,
      "listInterpret expected a pair as top");
    return;
  }
  JObj* operationName = asCar(top);
  JObj* operationBody = asCdr(top);
```

```
  pushCtxProcessImpl(aCtx, operationName);
  pushCtxProcessImpl(aCtx, operationBody);
}
```

```
\startWord[lispInterpret]

\preDataStack
  (
    (
      operationName : list
      operationBody
    )
    dataStack
  )
\preProcessStack
  (
    processStack
  )
\preConditions
\stopPreStack

\postDataStack
  ( dataStack )
\postProcessStack
  (
    operationBody : list <<< PREPEND?
    operationName : aType
    processStack
  )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void lispForAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxProcessIntoImpl(aCtx, nextCommand);
  if (symbolIs(jInterp, nextCommand, "forDone")) {
    // this lisp for loop is done
    DEBUG(jInterp, "lispForAP DONE%s\n", "");
```

```
  } else {
    DEBUG(jInterp, "lispForAP continue%s\n", "");
    if (!isPair(nextCommand)) {
      raiseExceptionMsg(aCtx,
        "listFor (continue) expected a pair as nextCommand");
      return;
    }
    JObj* operationName = asCar(nextCommand);
    JObj* operationBody = asCdr(nextCommand);
    pushSymbolCtxProcess(aCtx, "lispFor");
    pushCtxProcessImpl(aCtx, operationName);
    pushCtxProcessImpl(aCtx, operationBody);
  }
}
```

CCode : default

```
static void doneAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->process = NULL;
}
```

CCode : default

```
static void clearContextAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->data    = NULL;
  aCtx->process = NULL;
}
```

CCode : default

```
static void tryAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, handlerExp);
  popCtxDataIntoImpl(aCtx, tryExp);
  pushCtxProcessImpl(aCtx, handlerExp);
  pushSymbolCtxProcess(aCtx, "tryHandler");
  prependListCtxProcess(aCtx, tryExp);
}
```

CCode : default

```
static void raiseAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->exceptionRaised = TRUE;
```

```
  popCtxDataIntoImpl(aCtx, raiseExp);
  pushSymbolCtxProcess(aCtx, "findFirstTryHandler");
  prependListCtxProcess(aCtx, raiseExp);
}
```

CCode : default

```
static void raiseIfFalseAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataIntoImpl(aCtx, condition);
  popCtxProcessIntoImpl(aCtx, raiseExp);
  if (isFalse(condition)) {
    pushSymbolCtxProcess(aCtx, "findFirstTryHandler");
    prependListCtxProcess(aCtx, raiseExp);
  }
}
```

CCode : default

```
static ContextObj *findFirstTryHandlerAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  if (aCtx->process) {
    popCtxProcessIntoImpl(aCtx, aCommand);
    if (symbolIs(jInterp, aCommand, "tryHandler")) {
      aCtx->exceptionRaised = FALSE;
      popCtxProcessIntoImpl(aCtx, handlerExp);
      prependListCtxProcess(aCtx, handlerExp);
      return aCtx;
    }
    pushSymbolCtxProcess(aCtx, "findFirstTryHandler");
    return aCtx;
  }

  if (aCtx == jInterp->rootCtx) {
    aCtx->exceptionRaised = TRUE;
    return aCtx;
  }

  ContextObj* parentCtx = aCtx->parent;
```

```
23    if (!parentCtx ) parentCtx = jInterp->rootCtx;
24    assert(parentCtx);

25    popCtxDataIntoImpl(   aCtx,      oldContextTop );
26    pushCtxDataImpl(      parentCtx, oldContextTop );

27    wrapAP(parentCtx);

28    pushSymbolCtxProcess( parentCtx, "raise"       );
29
30    if (aCtx->tracingOn) {
31      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
32      strBufPrintf(aStrBuf,
33        "findFirstTryHandler(switchCtx)\n oldCtx: %s\n newCtx: %s\n",
34        aCtx->name, parentCtx->name
35      );
36      strBufPrintf(aStrBuf, "top = ");
37      printLoL(aStrBuf, oldContextTop);
38      strBufPrintf(aStrBuf, "\n");
39      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
40      strBufClose(aStrBuf);
41    }
42
43    return parentCtx;
44  }
```

CCode : default

```
1   static void tryHandlerAP(ContextObj* aCtx) {
2     popCtxProcessIntoImpl(aCtx, handlerExp);
3   }
```

CHeader : public

```
1   typedef void (RaiseException)(
2     ContextObj *aCtx,
3     Symbol     *message
4   );
5
6   #define raiseException(aCtx, message)       \
7     (                                         \
8       assert(aCtx),                           \
9       assert(getContextsClass(aCtx->jInterp)  \
10        ->raiseExceptionFunc),                \
```

```
    (getContextsClass(aCtx->jInterp)        \
      ->raiseExceptionFunc(aCtx, message))  \
  )
#define raiseExceptionMsg(aCtx, message)    \
  (                                         \
    assert(aCtx),                           \
    pushNullCtxData(aCtx),                  \
    assert(getContextsClass(aCtx->jInterp)  \
      ->raiseExceptionFunc),                \
    (getContextsClass(aCtx->jInterp)        \
      ->raiseExceptionFunc(aCtx, message))  \
  )
```

CHeader : private

```
extern void raiseExceptionImpl(
  ContextObj *aCtx,
  Symbol     *message
);
```

CCode : default

```
void raiseExceptionImpl(
  ContextObj *aCtx,
  Symbol     *message
) {
  assert(aCtx);
  JoyLoLInterp* jInterp = aCtx->jInterp;
  assert(jInterp);
  if (!message) message = "unknown";

  Symbol *file = "unknown(command)";
  size_t  line = 0;
  if (isSymbol(aCtx->command)) {
    file = asFile(aCtx->command);
    line = asLine(aCtx->command);
  }

  DEBUG(jInterp, "raiseException %p [%s] (%s) <%s> %zu\n",
    aCtx, aCtx->name, message, file, line
  );

  wrapAP(aCtx);
  pushSymbolCtxData(aCtx, file);
```

```
23    prependAP(aCtx);
24    pushNaturalCtxData(aCtx, line);
25    prependAP(aCtx);
26    pushSymbolCtxData(aCtx, message);
27    prependAP(aCtx);
28    pushCtxData(aCtx, (JObj*)aCtx);
29    prependAP(aCtx);
30    wrapAP(aCtx);
31    pushSymbolCtxProcess(aCtx, "raise");

32    if (aCtx->tracingOn) {
33      DEBUG(jInterp, "raiseException -> tracing%s\n", "");
34      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
35      strBufPrintf(aStrBuf, "d>>");
36      printLoL(aStrBuf, aCtx->data);
37      strBufPrintf(aStrBuf, "\n");
38      strBufPrintf(aStrBuf, "p>>");
39      printLoL(aStrBuf, aCtx->process);
40      strBufPrintf(aStrBuf, "\n");
41      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
42      strBufClose(aStrBuf);
43      DEBUG(jInterp, "raiseException <- tracing%s\n", "");
44    }
45  }
```

CHeader : public

```
1   typedef Boolean (ReportException)(
2     ContextObj* aCtx
3   );
4
5   #define reportException(aCtx)                \
6     (                                          \
7       assert(aCtx),                            \
8       assert(getContextsClass(aCtx->jInterp)   \
9         ->reportExceptionFunc),                \
10      (getContextsClass(aCtx->jInterp)         \
11        ->reportExceptionFunc(aCtx))           \
12    )
```

CHeader : private

```
1   extern Boolean reportExceptionImpl(
2     ContextObj* aCtx
```

```
);
```

CCode : default

```
Boolean reportExceptionImpl(
  ContextObj* aCtx
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  DEBUG(jInterp, "reportException %p [%s] %zu\n",
    aCtx, aCtx->name, aCtx->exceptionRaised
  );
  if (!aCtx->exceptionRaised) return FALSE;

  extractAP(aCtx);

  popCtxDataIntoImpl(aCtx, expCtxObj);
  ContextObj *expCtx = aCtx;
  if (isContext(expCtxObj)) {
    expCtx = (ContextObj*)expCtxObj;
  }
  assert(expCtx->dict);

  popCtxDataIntoImpl(aCtx, expMsg);
  popCtxDataIntoImpl(aCtx, expLine);
  popCtxDataIntoImpl(aCtx, expFile);
  popCtxDataIntoImpl(aCtx, expStack);
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  strBufPrintf(aStrBuf, "\nUNHANDLED EXCEPTION:\n");
  printLoL(aStrBuf, expMsg);
  strBufPrintf(aStrBuf, "\n\n     in file: ");
  if (isSymbol(expFile)) {
    strBufPrintf(aStrBuf, "%s", asSymbol(expFile));
  } else {
    printLoL(aStrBuf, expFile);
  }
  strBufPrintf(aStrBuf,   "\n     on line: ");
  printLoL(aStrBuf, expLine);
  strBufPrintf(aStrBuf, "\n\n   wordStack: ");
  printLoL(aStrBuf, expStack);
  strBufPrintf(aStrBuf,   "\n     context: %s(%s)",
    expCtx->name, expCtx->dict->name);
  strBufPrintf(aStrBuf,   "\n   dataStack: ");
```

```
  printLoL(aStrBuf, aCtx->data);
  strBufPrintf(aStrBuf,   "\nprocessStack: ");
  printLoL(aStrBuf, aCtx->process);
  strBufPrintf(aStrBuf, "\n");
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
  aCtx->exceptionRaised = FALSE;
  return TRUE;
}
```

3.5.5.3 Support

CCode : default

```
static void defineAP(ContextObj* aCtx) {
  assert(aCtx);
  popCtxDataIntoImpl(aCtx,    naming);
  popCtxDataIntoImpl(aCtx,    wordDefinition);

  popListInto(aCtx, wordDefinition, name);
  popListInto(aCtx, wordDefinition, preCondition);
  popListInto(aCtx, wordDefinition, definition);
  popListInto(aCtx, wordDefinition, postCondition);

  if (!isDictionary(naming)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, name);
    pushOnTopCtxDataImpl(aCtx, preCondition);
    pushOnTopCtxDataImpl(aCtx, definition);
    pushOnTopCtxDataImpl(aCtx, postCondition);
    pushOnTopCtxDataImpl(aCtx, naming);
    raiseException(aCtx,
      "define requires a dictionary as top");
    return;
  }
  if (!isSymbol(name)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, name);
    pushOnTopCtxDataImpl(aCtx, preCondition);
    pushOnTopCtxDataImpl(aCtx, definition);
    pushOnTopCtxDataImpl(aCtx, postCondition);
    pushOnTopCtxDataImpl(aCtx, naming);
    raiseException(aCtx,
```

```
      "define requires a symbol as wordDefinition top"
    );
    return;
  }
  if (!isAssertion(preCondition)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, name);
    pushOnTopCtxDataImpl(aCtx, preCondition);
    pushOnTopCtxDataImpl(aCtx, definition);
    pushOnTopCtxDataImpl(aCtx, postCondition);
    pushOnTopCtxDataImpl(aCtx, naming);
    raiseException(aCtx,
      "define requires an assertion as wordDefinition second");
    return;
  }
  if (!isAssertion(postCondition)) {
    printf("postCondition: %p %zu\n",
      postCondition, (size_t)asTag(postCondition));
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, name);
    pushOnTopCtxDataImpl(aCtx, preCondition);
    pushOnTopCtxDataImpl(aCtx, definition);
    pushOnTopCtxDataImpl(aCtx, postCondition);
    pushOnTopCtxDataImpl(aCtx, naming);
    raiseException(aCtx,
      "define requires an assertion as wordDefinitoin fourth");
    return;
  }
  defineJoyLoLIn(
    aCtx->jInterp,
    (DictObj*)naming,
    asSymbol(name),
    (AssertionObj*)preCondition,
    definition,
    (AssertionObj*)postCondition
  );
}
```

CCode : default

```
static void defineContextAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
```

```
popCtxDataIntoImpl(aCtx, contextName);
popCtxDataIntoImpl(aCtx, contextNamedIn);
popCtxDataIntoImpl(aCtx, contextNamingScope);
popCtxDataIntoImpl(aCtx, contextData);
popCtxDataIntoImpl(aCtx, contextProcess);
if (!isSymbol(contextName)) {
  pushNullCtxDataImpl(aCtx);
  pushOnTopCtxDataImpl(aCtx, contextProcess);
  pushOnTopCtxDataImpl(aCtx, contextData);
  pushOnTopCtxDataImpl(aCtx, contextNamingScope);
  pushOnTopCtxDataImpl(aCtx, contextNamedIn);
  pushOnTopCtxDataImpl(aCtx, contextName);
  raiseException(aCtx,
    "defineContext required a symbol as top");
  return;
}
if (!isDictionary(contextNamedIn)) {
  pushNullCtxDataImpl(aCtx);
  pushOnTopCtxDataImpl(aCtx, contextProcess);
  pushOnTopCtxDataImpl(aCtx, contextData);
  pushOnTopCtxDataImpl(aCtx, contextNamingScope);
  pushOnTopCtxDataImpl(aCtx, contextNamedIn);
  pushOnTopCtxDataImpl(aCtx, contextName);
  raiseException(aCtx,
    "defineContext required a dictionary as second");
  return;
}
if (!isDictionary(contextNamingScope)) {
  pushNullCtxDataImpl(aCtx);
  pushOnTopCtxDataImpl(aCtx, contextProcess);
  pushOnTopCtxDataImpl(aCtx, contextData);
  pushOnTopCtxDataImpl(aCtx, contextNamingScope);
  pushOnTopCtxDataImpl(aCtx, contextNamedIn);
  pushOnTopCtxDataImpl(aCtx, contextName);
  raiseException(aCtx,
    "defineContext required a dictionary as third");
  return;
}
ContextObj* newCtx = newContext(
  jInterp,
  asSymbol(contextName),
  aCtx,
```

```
    (DictObj*)contextNamingScope,
    contextData,
    contextProcess
  );
  assert(newCtx);
  defineContextIn(
    jInterp,
    (DictObj*)contextNamedIn,
    asSymbol(contextName),
    newCtx
  );
}
```

CCode : default

```
static void newContextAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  popCtxDataIntoImpl(aCtx, contextName);
  popCtxDataIntoImpl(aCtx, contextNamingScope);
  popCtxDataIntoImpl(aCtx, contextData);
  popCtxDataIntoImpl(aCtx, contextProcess);
  if (!isSymbol(contextName)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, contextProcess);
    pushOnTopCtxDataImpl(aCtx, contextData);
    pushOnTopCtxDataImpl(aCtx, contextNamingScope);
    pushOnTopCtxDataImpl(aCtx, contextName);
    raiseException(aCtx,
      "newContext requires a symbol as top");
    return;
  }
  if (!isDictionary(contextNamingScope)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, contextProcess);
    pushOnTopCtxDataImpl(aCtx, contextData);
    pushOnTopCtxDataImpl(aCtx, contextNamingScope);
    pushOnTopCtxDataImpl(aCtx, contextName);
    raiseException(aCtx,
      "newContext requires a dictionary as second");
    return;
  }
```

```
30    ContextObj* newCtx = newContext(
31      jInterp,
32      asSymbol(contextName),
33      aCtx,
34      (DictObj*)contextNamingScope,
35      contextData,
36      contextProcess
37    );
38    assert(newCtx);
39    pushCtxDataImpl(aCtx, (JObj*)newCtx);
40  }
```

CCode : default

```
1   static void thisContextAP(ContextObj* aCtx) {
2     assert(aCtx);
3     DEBUG(aCtx->jInterp, "thisContextAP > %p\n", aCtx);
4     pushCtxDataImpl(aCtx, (JObj*)aCtx);
5     DEBUG(aCtx->jInterp, "thisContextAP < %p\n", aCtx);
6   }
```

CCode : default

```
1   static ContextObj* switchCtxAP(ContextObj* aCtx) {
2     assert(aCtx);
3
4     popCtxDataIntoImpl(aCtx, newContext);
5     popCtxDataIntoImpl(aCtx, oldContextTop);
6     if (!isContext(newContext)) {
7       raiseExceptionMsg(aCtx,
8         "switchCtx required a context as top");
9       return aCtx;
10    }

11    //
12    // switch to newContext
13    //
14    DEBUG(aCtx->jInterp, "switchCtxAP -> switching origCtx: %s\n",
15      aCtx->name);
16    ContextObj* newCtx = (ContextObj*)newContext;
17    assert(newCtx);
18
19    pushCtxDataImpl(newCtx, oldContextTop);
20    newCtx->tracingOn = aCtx->tracingOn;
```

```
  DEBUG(aCtx->jInterp, "switchCtxAP <- switching newCtx: %s\n",
            newCtx->name);
  //
  return newCtx;
}
```

CCode : default

```
static void defineNamingAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  popCtxDataIntoImpl(aCtx, dictName);
  popCtxDataIntoImpl(aCtx, dictNamedIn);
  popCtxDataIntoImpl(aCtx, dictNamingScope);
  if (!isSymbol(dictName)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, dictNamingScope);
    pushOnTopCtxDataImpl(aCtx, dictNamedIn);
    pushOnTopCtxDataImpl(aCtx, dictName);
    raiseException(aCtx,
      "defineNaming required a symbol as top");
    return;
  }
  if (!isDictionary(dictNamedIn)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, dictNamingScope);
    pushOnTopCtxDataImpl(aCtx, dictNamedIn);
    pushOnTopCtxDataImpl(aCtx, dictName);
    raiseException(aCtx,
      "defineNaming requires a dictionary as second");
    return;
  }
  if (!isDictionary(dictNamingScope)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, dictNamingScope);
    pushOnTopCtxDataImpl(aCtx, dictNamedIn);
    pushOnTopCtxDataImpl(aCtx, dictName);
    raiseException(aCtx,
      "defineNaming requires a dictionary as third");
    return;
  }
  DictObj* newDict =
```

```
    newDictionary(
      jInterp,
      asSymbol(dictName),
      (DictObj*)dictNamingScope
    );
  assert(newDict);
  defineNamingIn(
    jInterp,
    (DictObj*)dictNamedIn,
    asSymbol(dictName),
    newDict
  );
}
```

CCode : default

```
static void newNamingAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  popCtxDataIntoImpl(aCtx, dictName);
  popCtxDataIntoImpl(aCtx, dictNamingScope);
  if (!isSymbol(dictName)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, dictNamingScope);
    pushOnTopCtxDataImpl(aCtx, dictName);
    raiseException(aCtx,
      "defineNaming requires a symbol as top");
    return;
  }
  if (!isDictionary(dictNamingScope)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, dictNamingScope);
    pushOnTopCtxDataImpl(aCtx, dictName);
    raiseException(aCtx,
      "defineNaming requires a dictionary as second");
    return;
  }

  DictObj* newDict =
    newDictionary(
      jInterp,
      asSymbol(dictName),
```

```
      (DictObj*)dictNamingScope
    );
  assert(newDict);
  pushCtxDataImpl(aCtx, (JObj*)newDict);
}
```

CCode : default

```
static void localizeNamingAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  popCtxDataIntoImpl(aCtx, localNaming);
  if (!isSymbol(localNaming)){
    raiseExceptionMsg(aCtx,
      "localizeNaming requires a symbol as top");
    return;
  }

  DictObj* newDict =
    newDictionary(
      jInterp,
      asSymbol(localNaming),
      aCtx->dict
    );
  assert(newDict);
  aCtx->dict = newDict;
}
```

CCode : default

```
static void thisNamingAP(ContextObj* aCtx) {
  assert(aCtx);
  pushCtxDataImpl(aCtx, (JObj*)(aCtx->dict));
}
```

CCode : default

```
static void undefineAP(ContextObj* aCtx) {
  assert(aCtx);

  popCtxDataIntoImpl(aCtx, namingScope);
  popCtxDataIntoImpl(aCtx, nameList);
```

```
  popListInto(aCtx, nameList, name);

  if (!isDictionary(namingScope)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, namingScope);
    pushOnTopCtxDataImpl(aCtx, name);
    raiseException(aCtx,
      "undelete requires a dictionary as top");
    return;
  }

  if (!isSymbol(name)) {
    pushNullCtxDataImpl(aCtx);
    pushOnTopCtxDataImpl(aCtx, namingScope);
    pushOnTopCtxDataImpl(aCtx, name);
    raiseException(aCtx,
      "undefine requires a list quoted symbol as second");
    return;
  }

  deleteSymbol(((DictObj*)namingScope), asSymbol(name));
  }
```

CCode : default

```
void initContextsAPControl(JoyLoLInterp *jInterp) {
  extendJoyLoLInRoot(jInterp, "pushList",       "", pushListAP,       "");
  extendJoyLoLInRoot(jInterp, "popList",        "", popListAP,        "");
  extendJoyLoLInRoot(jInterp, "wrap",           "", wrapAP,           "");
  extendJoyLoLInRoot(jInterp, "prepend",        "", prependAP,        "");
  extendJoyLoLInRoot(jInterp, "append",         "", appendAP,         "");
  extendJoyLoLInRoot(jInterp, "extract",        "", extractAP,        "");
  extendJoyLoLInRoot(jInterp, "i",              "", interpretAP,      "");
  extendJoyLoLInRoot(jInterp, "interpret",      "", interpretAP,      "");
  extendJoyLoLInRoot(jInterp, "ifte",           "", ifteAP,           "");
  extendJoyLoLInRoot(jInterp, "ifteCont",       "", ifteContAP,       "");
  extendJoyLoLInRoot(jInterp, "for",            "", forAP,            "");
  extendJoyLoLInRoot(jInterp, "forDone",        "", forDoneAP,        "");
  extendJoyLoLInRoot(jInterp, "lispInterpret",  "", lispInterpretAP,  "");
  extendJoyLoLInRoot(jInterp, "lispFor",        "", lispForAP,        "");
  extendJoyLoLInRoot(jInterp, "define",         "", defineAP,         "");
  extendJoyLoLInRoot(jInterp, "defineContext",  "", defineContextAP,  "");
  extendJoyLoLInRoot(jInterp, "newContext",     "", newContextAP,     "");
  extendJoyLoLInRoot(jInterp, "thisContext",    "", thisContextAP,    "");
```

```
  extendJoyLoLInRoot(jInterp, "defineNaming",    "", defineNamingAP,    "");
  extendJoyLoLInRoot(jInterp, "newNaming",       "", newNamingAP,       "");
  extendJoyLoLInRoot(jInterp, "localizeNaming", "", localizeNamingAP,
"");
  extendJoyLoLInRoot(jInterp, "thisNaming",      "", thisNamingAP,      "");
  extendJoyLoLInRoot(jInterp, "undefine",        "", undefineAP,        "");
  extendJoyLoLInRoot(jInterp, "done",            "", doneAP,            "");
  extendJoyLoLInRoot(jInterp, "clear",           "", clearContextAP,    "");
  extendJoyLoLInRoot(jInterp, "try",             "", tryAP,             "");
  extendJoyLoLInRoot(jInterp, "raise",           "", raiseAP,           "");
  extendJoyLoLInRoot(jInterp, "raiseIfFalse",    "", raiseIfFalseAP,    "");
  extendJoyLoLInRoot(jInterp, "tryHandler",      "", tryHandlerAP,      "");

  extendCtxJoyLoLInRoot(jInterp, "findFirstTryHandler", "", findFirstTryHandlerAP,
"");
  extendCtxJoyLoLInRoot(jInterp, "switchCtx",           "", switchCtxAP,
"");
}
```

3.5.6 Supporting JoyLoL words

CHeader : public

```
#define isContext(aLoL)           \
  (                               \
    (                             \
      (aLoL) &&                   \
      (aLoL->tag == ContextsTag)  \
    ) ?                           \
      TRUE :                      \
      FALSE                       \
  )
```

CCode : default

```
static void isContextAP(ContextObj* aCtx) {
  assert(aCtx);
  popCtxDataIntoImpl(aCtx, top);
  JObj* result = NULL;
  if (isContext(top)) result = newBoolean(aCtx->jInterp, TRUE);
  else                result = newBoolean(aCtx->jInterp, FALSE);
  pushCtxDataImpl(aCtx, result);
}
```

CCode : default

```
static void showStackOnAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->showStack = TRUE;
}
```

CCode : default

```
static void showStackOffAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->showStack = FALSE;
}
```

CCode : default

```
static void showStackAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  strBufPrintf(aStrBuf, "d>>");
  printLoL(aStrBuf, aCtx->data);
  strBufPrintf(aStrBuf, "\np>>");
  printLoL(aStrBuf, aCtx->process);
  strBufPrintf(aStrBuf, "\n");
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CCode : default

```
static void tracingOnAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->tracingOn = TRUE;
}
```

CCode : default

```
static void tracingOffAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->tracingOn = FALSE;
}
```

CCode : default

```
static void debugOnAP(ContextObj* aCtx) {
```

```
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->jInterp->debug = TRUE;
}
```

CCode : default

```
static void debugOffAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->jInterp->debug = FALSE;
}
```

CCode : default

```
static void verboseOnAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->verbose = TRUE;
  aCtx->jInterp->verbose = TRUE;
}
```

CCode : default

```
static void verboseOffAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  aCtx->verbose = FALSE;
  aCtx->jInterp->verbose = FALSE;
}
```

CCode : default

```
static void checkingOnAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->checkingOn = TRUE;
}
```

CCode : default

```
static void checkingOffAP(ContextObj* aCtx) {
  assert(aCtx);
  aCtx->checkingOn = FALSE;
}
```

CCode : default

```
static void definitionsAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  listDefinitions(aCtx->dict, aStrBuf);
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CCode : default

```
static void definitionsInAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  popCtxDataIntoImpl(aCtx, namingScope);
  if (!isDictionary(namingScope)) {
    raiseExceptionMsg(aCtx,
      "definitionsIn requires a dictionary as top");
    return;
  }
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  listDefinitions(((DictObj*)namingScope), aStrBuf);
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CCode : default

```
static void showLoadExtensionsAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  LoaderObj* loader = jInterp->loader;
  assert(loader);
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  listLoadExtensions(loader, aStrBuf);
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CCode : default

```
static void showLoadPathsAP(ContextObj* aCtx) {
```

```
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  LoaderObj* loader = jInterp->loader;
  assert(loader);
  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  listLoadPaths(loader, aStrBuf);
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CCode : default

```
static void loadExtensionAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  LoaderObj *loader = jInterp->loader;
  assert(loader);
  popCtxDataIntoImpl(aCtx, top);
  if (!isSymbol(top)) {
    raiseExceptionMsg(aCtx,
      "loadExtension requires a symbol as top");
    return;
  }
  pushLoadExtension(loader, asSymbol(top));
}
```

CCode : default

```
static void loadPathAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  LoaderObj *loader = jInterp->loader;
  assert(loader);
  popCtxDataIntoImpl(aCtx, top);
  if (!isSymbol(top)) {
    raiseExceptionMsg(aCtx,
      "loadPath requires a symbol as top");
    return;
  }
  pushLoadPath(loader, asSymbol(top));
```

```
}
```

CCode : default

```
static void loadFileAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  DEBUG(aCtx->jInterp, "loadFileAP > %p\n", aCtx);
  assert(aCtx);
  popCtxDataIntoImpl(aCtx, top);
  if (!isSymbol(top)) {
    raiseExceptionMsg(aCtx,
      "loadFile requires a symbol as top");
    return;
  }

  int oldVerboseFlag = aCtx->verbose;
  aCtx->verbose = aCtx->showStack;
  loadAFile(aCtx, asSymbol(top));
  aCtx->verbose = oldVerboseFlag;
  DEBUG(aCtx->jInterp, "loadFileAP < %p\n", aCtx);
}
```

CCode : default

```
static void lispLoadFileAP(ContextObj* aCtx) {
  assert(aCtx);
  assert(aCtx->jInterp);
  DEBUG(aCtx->jInterp, "listLoadFileAP > %p\n", aCtx);
  pushSymbolCtxProcess(aCtx, "lispInterpret");
  loadFileAP(aCtx);
  DEBUG(aCtx->jInterp, "listLoadFileAP < %p\n", aCtx);
}
```

CCode : default

```
static void whatIsThis(
  ContextObj *aCtx,
  JObj       *top,
  Symbol     *stackName
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
```

```
9   StringBufferObj *aStrBuf = newStringBuffer(aCtx);
10  strBufPrintf(aStrBuf, "%s top ", stackName);
11
12  if (!top) {
13    strBufPrintf(aStrBuf, "is NIL\n");
14  } else {
15    if (!top->type) {
16      strBufPrintf(aStrBuf, "has no type\n");
17    } else {
18      strBufPrintf(aStrBuf,
19        "data top is a %s [",
20        top->type->name
21      );
22      printLoL(aStrBuf, top);
23      strBufPrintf(aStrBuf, "]\n");
24    }
25  }
26  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
27  strBufClose(aStrBuf);
28 }
29
30 static void whatIsThisDAP(ContextObj* aCtx) {
31   assert(aCtx);
32   popCtxDataInto(aCtx, top);
33   pushCtxData(aCtx, top);
34   whatIsThis(aCtx, top, "data");
35 }
36
37 static void whatIsThisPAP(ContextObj* aCtx) {
38   assert(aCtx);
39   popCtxProcessInto(aCtx, top);
40   pushCtxProcess(aCtx, top);
41   whatIsThis(aCtx, top, "process");
42 }
```

CCode : default

```
1 static void exitJoylolAP(ContextObj *aCtx) {
2   assert(aCtx);
3   JoyLoLInterp *jInterp = aCtx->jInterp;
```

```
4     assert(jInterp);

5     popCtxDataInto(aCtx, top);
6     double resultDbl  = asNaturalDbl(jInterp, top);
7     int    resultCode = (int)resultDbl;

8     if (jInterp->verbose) {
9       StringBufferObj *aStrBuf = newStringBuffer(aCtx);
10      strBufPrintf(
11        aStrBuf,
12        "exiting joylol with code: %d\n",
13        resultCode
14      );
15      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
16      strBufClose(aStrBuf);
17    }
18    exit(resultCode);
19  }
```

CCode : default

```
1   static void tracingPointAP(ContextObj *aCtx) {
2     assert(aCtx);
3     JoyLoLInterp *jInterp = aCtx->jInterp;
4     assert(jInterp);
5     DictObj *theDict = aCtx->dict;
6     assert(theDict);

7     popCtxDataInto(aCtx, tracingLabel);

8     if (!isSymbol(tracingLabel)) {
9       pushNullCtxDataImpl(aCtx);
10      pushOnTopCtxDataImpl(aCtx, tracingLabel);
11      raiseException(aCtx,
12        "tracingPoint requires a symbol as top"
13      );
14      return;
15    }

16    Symbol *theLabel = asSymbol(tracingLabel);
17
18    CtxTracingLabel *curLabel  = aCtx->tracingLabels;
19    CtxTracingLabel *lastLabel = NULL;
20
```

```
21    while (curLabel) {
22      if (strcmp(curLabel->name, theLabel) == 0 ) break;
23      lastLabel = curLabel;
24      curLabel  = curLabel->next;
25    }
26
27    if (!curLabel) {
28      curLabel = calloc(1, sizeof(CtxTracingLabel));
29      assert(curLabel);
30      curLabel->name  = strdup(theLabel);
31      curLabel->count = 0;
32      curLabel->next  = NULL;
33      if (lastLabel) {
34        lastLabel->next = curLabel;
35      } else {
36        aCtx->tracingLabels = curLabel;
37      }
38    }
39
40    StringBufferObj *aStrBuf = newStringBuffer(aCtx);
41    strBufPrintf(aStrBuf, "tracingPoint: %zu <<%s>> %s %s\n",
42      curLabel->count, curLabel->name, aCtx->name, theDict->name
43    );
44    jInterp->writeStdOut(jInterp, getCString(aStrBuf));
45    strBufClose(aStrBuf);

46    curLabel->count++;
47  }
```

CHeader : private

```
1   extern void initContextsAPSupport(JoyLoLInterp *jInterp);
```

CCode : default

```
1   void initContextsAPSupport(JoyLoLInterp *jInterp) {
2     extendJoyLoLInRoot(jInterp, "isContext",          "", isContextAP,          "");
3     extendJoyLoLInRoot(jInterp, "definitions",        "", definitionsAP,
4   "");
5     extendJoyLoLInRoot(jInterp, "definitionsIn",      "", definitionsInAP,
6   "");
7     extendJoyLoLInRoot(jInterp, "loadExtension",      "", loadExtensionAP,
8   "");
9     extendJoyLoLInRoot(jInterp, "loadPath",           "", loadPathAP,           "");
```

```
  extendJoyLoLInRoot(jInterp, "load",              "", loadFileAP,          "");
  extendJoyLoLInRoot(jInterp, "lispLoad",          "", lispLoadFileAP,
"");
  extendJoyLoLInRoot(jInterp, "showLoadExtensions", "", showLoadExtensionsAP,
"");
  extendJoyLoLInRoot(jInterp, "showLoadPaths",     "", showLoadPathsAP,
"");
  extendJoyLoLInRoot(jInterp, "showStack",         "", showStackAP,         "");
  extendJoyLoLInRoot(jInterp, "showStackOn",       "", showStackOnAP,
"");
  extendJoyLoLInRoot(jInterp, "showStackOff",      "", showStackOffAP,
"");
  extendJoyLoLInRoot(jInterp, "tracingOn",         "", tracingOnAP,         "");
  extendJoyLoLInRoot(jInterp, "tracingOff",        "", tracingOffAP,
"");
  extendJoyLoLInRoot(jInterp, "verboseOn",         "", verboseOnAP,         "");
  extendJoyLoLInRoot(jInterp, "verboseOff",        "", verboseOffAP,
"");
  extendJoyLoLInRoot(jInterp, "debugOn",           "", debugOnAP,           "");
  extendJoyLoLInRoot(jInterp, "debugOff",          "", debugOffAP,          "");
  extendJoyLoLInRoot(jInterp, "checkingOn",        "", checkingOnAP,
"");
  extendJoyLoLInRoot(jInterp, "checkingOff",       "", checkingOffAP,
"");
  extendJoyLoLInRoot(jInterp, "whatIsThisD",       "", whatIsThisDAP,
"");
  extendJoyLoLInRoot(jInterp, "whatIsThisP",       "", whatIsThisPAP,
"");
  extendJoyLoLInRoot(jInterp, "exitJoylol",        "", exitJoylolAP,
"");
  extendJoyLoLInRoot(jInterp, "tracingPoint",      "", tracingPointAP,
"");
}
```

3.5.7 Evaluation

CCode : default

```
static void traceAction(
  ContextObj *aCtx,
  Symbol     *action,
  JObj       *aLoL
) {
```

```
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  DEBUG(jInterp, "traceAction: %p [%s] %p\n",
    aCtx, action, aLoL);
  if (aLoL && jInterp->debug) {
    assert(asType(aLoL));
    DEBUG(jInterp, "traceAction(lol): %p %p\n",
      aLoL, asType(aLoL));
  }

  StringBufferObj *aStrBuf = newStringBuffer(aCtx);
  strBufPrintf(aStrBuf, "%s: ", action);
  printLoL(aStrBuf, aLoL);
  strBufPrintf(aStrBuf, "\n");
  jInterp->writeStdOut(jInterp, getCString(aStrBuf));
  strBufClose(aStrBuf);
}
```

CHeader : public

```
typedef void (EvalCommandInContext)(
  ContextObj *aCtx,
  JObj   *command
);

#define evalCommandInContext(aCtx, aCommand)      \
  (                                               \
    assert(aCtx),                                 \
    assert(getContextsClass(aCtx->jInterp)        \
      ->evalCommandInContextFunc),                \
    (getContextsClass(aCtx->jInterp)              \
      ->evalCommandInContextFunc(aCtx, aCommand)) \
  )
```

CHeader : private

```
void evalCommandInContextImpl(
  ContextObj *aCtx,
  JObj   *command
);
```

CCode : default

```
void evalCommandInContextImpl(
```

```
2     ContextObj *aCtx,
3     JObj       *command
4   ) {
5     assert(aCtx);
6     JoyLoLInterp *jInterp = aCtx->jInterp;
7     assert(jInterp);
8     //assert(command);

9     DEBUG(jInterp, "evalCommandInContext > %p [%s] %p\n",
10      aCtx, aCtx->name, command);
11    //
12    // push this command onto the top of the process stack
13    //
14    pushCtxProcessImpl(aCtx, command);
15    //
16    while(aCtx->process) {
17      //
18      // ensure we have the most recent dictionary
19      //
20      DictObj *theDict = aCtx->dict;
21      assert(theDict);

22      if (aCtx->tracingOn) {
23        StringBufferObj *aStrBuf = newStringBuffer(aCtx);
24        strBufPrintf(aStrBuf,
25          "\n-----------------------------------------------------\n"
26        );
27        strBufPrintf(aStrBuf,
28          "ctx: %s(%s)\n",
29          aCtx->name, theDict->name
30        );
31        jInterp->writeStdOut(jInterp, getCString(aStrBuf));
32        strBufClose(aStrBuf);
33      }
34      //
35      // pop the next command off the process stack
36      //
37      command = popCtxProcessImpl(aCtx);
38      aCtx->command = command;
39      //assert(command);
40      //
41      if (isCFunction(command)) {
42        //
```

```
43      // if the command is a function.. call the function
44      //
45      if (isCtxCFunction(command)) {
46        assert(asCtxCFunc(command));
47        //
48        // this is a CTX CFunction
49        // (we allow it to change the current context
50        //  in addition to any changes of the data and process stacks
51        //  of either the old or new contexts)
52        //
53        if (aCtx->tracingOn)
54          traceAction(aCtx, "calling(c-ctx)", command);
55        aCtx = (asCtxCFunc(command))(aCtx);
56        assert(aCtx);
57        //
58      } else {
59        assert(asCFunc(command));
60        //
61        // this is a normal CFunction
62        // (this ONLY makes changes to the data and process stacks)
63        //
64        if (aCtx->tracingOn)
65          traceAction(aCtx, "calling(c)", command);
66        (asCFunc(command))(aCtx);
67        //
68      }
69    } else if (isAssertion(command)) {
70      //
71      // this is an assertion ...
72      //   ... so assert it
73      //
74      if (aCtx->tracingOn)
75        traceAction(aCtx, "asserting", command);
76      //
77      if (!evalAssertionInContextImpl(aCtx, (AssertionObj*)command)) {
78        //
79        // this assertion failed...
80        //   ... so report it
81        //
82        pushNullCtxDataImpl(aCtx);
83        pushOnTopCtxDataImpl(aCtx, command);
84        raiseException(aCtx,
85          "assertion failed"
```

```
86          );
87        }
88      } else if (!isSymbol(command)) {
89        //
90        // if the command is not a Symbol ...
91        //  ...  push it onto the top of the data stack
92        //
93        if (aCtx->tracingOn)
94          traceAction(aCtx, "adding(nonSym)", command);
95        pushCtxDataImpl(aCtx, command);
96        //
97      } else {
98        //
99        // if the command is a symbol ...
100       //  ... look up the symbol's association in the dictionary
101       //
102       DictNodeObj* assoc = getSymbolEntry(theDict, asSymbol(command));
103       assert(assoc);
104       //
105       if (!assoc->value) {
106         //
107         // if the association is empty.. push this symbol onto the top
108         // of the data stack (re-evaluating this symbol would lead to an
109         // infinite loop)
110         //
111         if (aCtx->tracingOn)
112           traceAction(aCtx, "adding(noValue)", command);
113         pushCtxDataImpl(aCtx, command);
114         //
115       } else if (isPair(assoc->value)) {
116         //
117         // if the association is a LoL.. push this LoL onto the top of the
118         // process stack
119         //
120         if (aCtx->tracingOn) {
121           traceAction(aCtx, "calling(joylol)", command);
122           traceAction(aCtx, "evaluating", assoc->value);
123         }
124         prependListCtxProcess(aCtx,
125           copyLoL(jInterp, assoc->value));
126         //
127       } else if (isCFunction(assoc->value)) {
128         //
```

```
129         // if the association is a function.. call the function
130         //
131         if (isCtxCFunction(assoc->value)) {
132           assert(asCtxCFunc(assoc->value));
133           //
134           // this is a CTX CFunction
135           //
136           if (aCtx->tracingOn)
137             traceAction(aCtx, "calling(c-ctx)", command);
138           aCtx = (asCtxCFunc(assoc->value))(aCtx);
139           assert(aCtx);
140           //
141         } else {
142           assert(asCFunc(assoc->value));
143           //
144           // this is a normal CFunction
145           //
146           if (aCtx->tracingOn)
147             traceAction(aCtx, "calling(c)", command);
148           (asCFunc(assoc->value))(aCtx);
149           //
150         }
151       } else {
152         //
153         // if the association is NOT a PairAtom or Function...
154         // ... push this new ATOM onto the top of the process stack
155         //
156         if (aCtx->tracingOn) {
157           traceAction(aCtx, "calling(joylol)", command);
158           traceAction(aCtx, "evaluating", assoc->value);
159         }
160         pushCtxProcessImpl(aCtx, assoc->value);
161         //
162       }
163     }
164     if (aCtx->tracingOn) {
165       DEBUG(jInterp, "evalCommandInContext -> tracing%s\n", "");
166       StringBufferObj *aStrBuf = newStringBuffer(aCtx);
167       showCtxData(aCtx, aStrBuf);
168       showCtxProcess(aCtx, aStrBuf);
169       jInterp->writeStdOut(jInterp, getCString(aStrBuf));
170       strBufClose(aStrBuf);
171       DEBUG(jInterp, "evalCommandInContext <- tracing%s\n", "");
```

```
172     }
173   } // aCtx->process is empty
174   DEBUG(jInterp, "evalCommandInContext < %p %p\n", aCtx, command);
175 }
```

CHeader : public

```
1  typedef void (EvalContext)(
2    ContextObj *aCtx
3  );
4
5  #define evalContext(aCtx)                    \
6    (                                          \
7      assert(aCtx),                            \
8      assert(getContextsClass(aCtx->jInterp)   \
9        ->evalContextFunc),                    \
10     (getContextsClass(aCtx->jInterp)         \
11       ->evalContextFunc(aCtx))               \
12   )
```

CHeader : private

```
1  void evalContextImpl(
2    ContextObj *aCtx
3  );
```

CCode : default

```
1  void evalContextImpl(
2    ContextObj *aCtx
3  ) {
4    popCtxProcessIntoImpl(aCtx, aCommand);
5    evalCommandInContext(aCtx, aCommand);
6  }
```

3.5.7.1 Test Suite: evalAssertionInContext

CHeader : public

```
1  typedef Boolean (EvalAssertionInContext)(
2    ContextObj   *aCtx,
3    AssertionObj *anAssertion
4  );
5
6  #define evalAssertionInContext(aCtx, anAssertion)       \
```

```
7   (                                                    \
8     assert(aCtx),                                      \
9     assert(getContextsClass(aCtx->jInterp)             \
10      ->evalAssertionInContextFunc),                   \
11    (getContextsClass(aCtx->jInterp)                   \
12      ->evalAssertionInContextFunc(aCtx, anAssertion))  \
13  )
```

CHeader : private

```
1  extern Boolean evalAssertionInContextImpl(
2    ContextObj   *aCtx,
3    AssertionObj *anAssertion
4  );
```

CCode : default

```
1  Boolean evalAssertionInContextImpl(
2    ContextObj   *aCtx,
3    AssertionObj *anAssertion
4  ) {
5    assert(aCtx);
6    JoyLoLInterp *jInterp = aCtx->jInterp;
7    assert(jInterp);
8    DictObj *theDict = aCtx->dict;
9    assert(theDict);

10   DEBUG(jInterp, "evalAssertionInContext > %p [%s] %p\n",
11     aCtx, aCtx->name, anAssertion);
12
13   char *metaDictName = calloc(10+strlen(theDict->name), sizeof(char));
14   assert(metaDictName);
15   strcat(metaDictName, "meta-");
16   strcat(metaDictName, theDict->name);

17   DictObj *metaDict = newDictionary(
18     jInterp,
19     metaDictName,
20     theDict
21   );

22   char *metaCtxName = calloc(10+strlen(aCtx->name), sizeof(char));
23   assert(metaCtxName);
24   strcat(metaCtxName, "meta-");
```

```
25    strcat(metaCtxName, aCtx->name);

26    ContextObj *metaCtx = newContextImpl(
27      jInterp,
28      metaCtxName,
29      aCtx,
30      metaDict,
31      aCtx->data,
32      NULL
33    );
34    assert(metaCtx);
35    metaCtx->showStack = aCtx->showStack;
36    metaCtx->tracingOn = aCtx->tracingOn;
37    JObj *originalTop  = aCtx->data;

38    Boolean assertionTrue = TRUE;
39
40    JObj *assertionList = asAssertion(anAssertion);
41    while(assertionList && !(metaCtx->exceptionRaised)) {
42      JObj *assertionTest = assertionList;
43      if (isPair(assertionList)) {
44        assertionTest = asCar(assertionList);
45        assertionList = asCdr(assertionList);
46      } else {
47        assertionList = NULL;
48      }
49
50      if (metaCtx->tracingOn)
51        traceAction(aCtx,
52          "\n------------------------\nevalAssertion(test)",
53          assertionTest
54        );
55      prependListCtxProcess(metaCtx, assertionTest);
56      popCtxProcessIntoImpl(metaCtx, aCommand);
57      evalCommandInContextImpl(metaCtx, aCommand);

58      if (metaCtx->data == originalTop) {
59        // our assertion has not returned any result value
60        pushNullCtxDataImpl(metaCtx);
61        pushOnTopCtxDataImpl(metaCtx, assertionTest);
62        raiseExceptionImpl(metaCtx,
63          "assertion has not returned any result value"
64          );
```

```
65      metaCtx->exceptionRaised = TRUE;
66      extractAP(metaCtx);
67    } else if ( !isPair(metaCtx->data) ) {
68      // our assertions have corrupted the data stack
69      pushNullCtxDataImpl(metaCtx);
70      pushOnTopCtxDataImpl(metaCtx, assertionTest);
71      raiseExceptionImpl(metaCtx,
72        "assertion has corrupted the data stack"
73        );
74      metaCtx->exceptionRaised = TRUE;
75      extractAP(metaCtx);
76    } else if ( asCdr(metaCtx->data) != originalTop ) {
77      // our assertions have corrupted the original context's data stack
78      pushNullCtxDataImpl(metaCtx);
79      pushOnTopCtxDataImpl(metaCtx, assertionTest);
80      raiseExceptionImpl(metaCtx,
81        "assertion has returned too many values OR corrupted the data stack"
82        );
83      metaCtx->exceptionRaised = TRUE;
84      extractAP(metaCtx);
85    } else {
86      popCtxDataIntoImpl(metaCtx, assertionResult);
87      if (isFalse(assertionResult)) {
88        assertionTrue = FALSE;
89      }
90    }
91  }

92  if (metaCtx->exceptionRaised) {
93    assertionTrue = FALSE;
94    reportException(metaCtx);
95  }

96  if (metaDictName) free(metaDictName);
97  metaDictName = NULL;

98  if (metaCtxName) free(metaCtxName);
99  metaCtxName = NULL;

100 DEBUG(jInterp, "evalAssertionInContext < %p [%s] %p\n",
101   aCtx, aCtx->name, anAssertion);

102 return assertionTrue;
```

```
103 }
```

Test case

should evaluate assertions

```
AssertPtrNotNull(jInterp);
ContextObj *rootCtx = jInterp->rootCtx;
AssertPtrNotNull(rootCtx);
DictObj *rootDict = rootCtx->dict;
AssertPtrNotNull(rootDict);

JObj *trueBoolean      = newBoolean(jInterp, TRUE);
JObj *aTrueAssertion   = newAssertion(jInterp, trueBoolean);
JObj *falseBoolean     = newBoolean(jInterp, FALSE);
JObj *aFalseAssertion  = newAssertion(jInterp, falseBoolean);
JObj *isBooleanSym     = newSymbol(jInterp, "isBoolean", "CTest", 1);
JObj *dupSym           = newSymbol(jInterp, "dup1D", "CTest", 2);
JObj *complexBody      = concatLists(jInterp,
  trueBoolean,
  newPair(jInterp,
    concatLists(jInterp, dupSym, isBooleanSym),
    NULL
  )
);
JObj *complexAssertion = newAssertion(jInterp, complexBody);

DictObj    *aDict = newDictionary(
  jInterp,
  "assertionCTestsDict",
  rootDict
);
ContextObj* aCtx = newContext(
  jInterp,
  "assertionCTestsCtx",
  NULL,
  aDict,
  NULL,
  NULL
);
AssertPtrNotNull(aCtx);
//aCtx->showStack = TRUE;
//aCtx->tracingOn = TRUE;
```

```
  //jInterp->debug  = TRUE;
  pushCtxData(aCtx, falseBoolean);

  Boolean result =
    evalAssertionInContextImpl(aCtx, (AssertionObj*)aTrueAssertion);
  AssertIntTrue(result);

  result =
    evalAssertionInContextImpl(aCtx, (AssertionObj*)aFalseAssertion);
  AssertIntFalse(result);

  result =
    evalAssertionInContextImpl(aCtx, (AssertionObj*)complexAssertion);
//  AssertIntTrue(result);
```

CHeader : public

```
typedef void (EvalTextInContext)(
  ContextObj *aCtx,
  TextObj    *aText
);

#define evalTextInContext(aCtx, aText)       \
  (                                          \
    assert(aCtx),                            \
    assert(getContextsClass(aCtx->jInterp)   \
      ->evalTextInContextFunc),              \
    (getContextsClass(aCtx->jInterp)         \
      ->evalTextInContextFunc(aCtx, aText)) \
  )
```

CHeader : private

```
extern void evalTextInContextImpl(
  ContextObj *aCtx,
  TextObj    *aText
);
```

CCode : default

```
void evalTextInContextImpl(
  ContextObj *aCtx,
  TextObj    *aText
) {
  assert(aCtx);
```

```
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  DEBUG(jInterp, "evalTextInContext > %p %p\n", aCtx, aText);
  while(TRUE) {
    JObj* aLoL = parseOneSymbol(aText);
    if (aCtx->showStack) {
      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
      strBufPrintf(aStrBuf, "<");
      if (aLoL && isSignal(aLoL) &&
        asSignal(aLoL) == SIGNAL_END_OF_TEXT) {
        strBufPrintf(aStrBuf, " {EOT}");
      } else {
        printLoL(aStrBuf, aLoL);
      }
      strBufPrintf(aStrBuf, "\n");
      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
      strBufClose(aStrBuf);
    }
    if (aLoL && isSignal(aLoL) &&
      asSignal(aLoL) == SIGNAL_END_OF_TEXT) break;
    evalCommandInContext(aCtx, aLoL);
    if (aCtx->showStack) {
      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
      strBufPrintf(aStrBuf, ">");
      printLoL(aStrBuf, aCtx->data);
      strBufPrintf(aStrBuf, "\n");
      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
      strBufClose(aStrBuf);
    }
    if (aCtx->exceptionRaised) break;
  }
  reportException(aCtx);
  DEBUG(jInterp, "evalTextInContext < %p %p\n", aCtx, aText);
}
```

CHeader : public

```
typedef void (EvalArrayOfStringsInContext)(
  ContextObj *aCtx,
  Symbol     *someStrings[]
);

#define evalArrayOfStringsInContext(aCtx, someStrings)      \
```

```
  (                                                          \
    assert(aCtx),                                            \
    assert(getContextsClass(aCtx->jInterp)                   \
      ->evalArrayOfStringsInContextFunc),                    \
    (getContextsClass(aCtx->jInterp)                         \
      ->evalArrayOfStringsInContextFunc(aCtx, someStrings)) \
  )
```

CHeader : private

```
extern void evalArrayOfStringsInContextImpl(
  ContextObj *aCtx,
  Symbol     *someStrings[]
);
```

CCode : default

```
void evalArrayOfStringsInContextImpl(
  ContextObj *aCtx,
  Symbol     *someStrings[]
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  DEBUG(aCtx->jInterp, "evalArrayOfStringsInContext > %p %p\n",
    aCtx, someStrings);
  assert(aCtx);
  TextObj* stringText =
    createTextFromArrayOfStrings(
      aCtx->jInterp,
      someStrings
    );
  evalTextInContext(aCtx, stringText);
  freeText(stringText);
  DEBUG(aCtx->jInterp, "evalArrayOfStringsInContext < %p %p\n",
    aCtx, someStrings);
}
```

CHeader : public

```
typedef void (EvalStringInContext)(
  ContextObj *aCtx,
  Symbol     *aString
);

#define evalStringInContext(aCtx, aString)      \
```

```
  (                                                \
    assert(aCtx),                                  \
    assert(getContextsClass(aCtx->jInterp)         \
      ->evalStringInContextFunc),                  \
    (getContextsClass(aCtx->jInterp)               \
      ->evalStringInContextFunc(aCtx, aString)) \
  )
```

CHeader : private

```
extern void evalStringInContextImpl(
  ContextObj *aCtx,
  Symbol     *aString
);
```

CCode : default

```
void evalStringInContextImpl(
  ContextObj *aCtx,
  Symbol     *aString
) {
  assert(aCtx);
  assert(aCtx->jInterp);
  DEBUG(aCtx->jInterp, "evalStringInContext > %p [%s]\n", aCtx, aString);
  assert(aCtx);
  TextObj* stringText =
    createTextFromString(
      aCtx->jInterp,
      aString
    );
  evalTextInContext(aCtx, stringText);
  freeText(stringText);
  DEBUG(aCtx->jInterp, "evalStringInContext < %p [%s]\n", aCtx, aString);
}
```

3.5.8 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
```

```
  { "subject",          "updated textadept lexer for JoyLoL"},
  { "notes",            ""},
  { NULL,                 NULL}
};
```

CCode : default

```
static int lua_contexts_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_contexts [] = {
  {"gitVersion", lua_contexts_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_contexts (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerContexts(jInterp);
  luaL_newlib(lstate, lua_contexts);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedContexts` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedContexts(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedContexts(
  lua_State *lstate
) {
```

```
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_contexts(lstate);
  lua_setfield(lstate, -2, "joylol.contexts");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.5.9 JoyLoL operators

In this section we develop the primary JoyLoL operators. These operators are modelled upon those used by Manfred von Thun in his language Joy, [Thu94b]. We will implement his full range of unary, binary and ternary operators as defined in the 'General Operators' section of [Thu95]. We provide a duplicate set for each of the data and process stacks of a given context. We will also provide a set which transfer items between the data and process stacks. We will use slightly more mnemonic names.

3.5.9.1 Unary operators

CCode : default

```
static void pop1DAP(ContextObj* aCtx) {
  popCtxDataImpl(aCtx);
}
```

CCode : default

```
static void pop1PAP(ContextObj* aCtx) {
  popCtxProcessImpl(aCtx);
}
```

CCode : default

```
static void dup1DAP(ContextObj* aCtx) {
  peekCtxDataIntoImpl(aCtx, top);
  pushCtxDataImpl(aCtx, copyLoL(aCtx->jInterp, top));
}
```

CCode : default

```
static void dup1PAP(ContextObj* aCtx) {
  peekCtxProcessIntoImpl(aCtx, top);
```

```
  pushCtxProcessImpl(aCtx, copyLoL(aCtx->jInterp, top));
}
```

3.5.9.2 Binary operators

CCode : default

```
static void swap12DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  pushCtxDataImpl(aCtx, top);
  pushCtxDataImpl(aCtx, second);
}
```

CCode : default

```
static void swap12PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, top);
  pushCtxProcessImpl(aCtx, second);
}
```

CCode : default

```
static void pop2DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataImpl(aCtx);
  pushCtxDataImpl(aCtx, top);
}
```

CCode : default

```
static void pop2PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessImpl(aCtx);
  pushCtxProcessImpl(aCtx, top);
}
```

CCode : default

```
static void pop12DAP(ContextObj* aCtx) {
  popCtxDataImpl(aCtx);
  popCtxDataImpl(aCtx);
```

```
}
```

CCode : default

```
static void pop12PAP(ContextObj* aCtx) {
  popCtxProcessImpl(aCtx);
  popCtxProcessImpl(aCtx);
}
```

CCode : default

```
static void dup2DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, copyLoL(aCtx->jInterp, second));
  pushCtxDataImpl(aCtx, top);
}
```

CCode : default

```
static void dup2PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, copyLoL(aCtx->jInterp, second));
  pushCtxProcessImpl(aCtx, top);
}
```

3.5.9.3 Ternary operators

CCode : default

```
static void swap13DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  pushCtxDataImpl(aCtx, top);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, third);
}
```

CCode : default

```
static void swap13PAP(ContextObj* aCtx) {
```

```
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  popCtxProcessIntoImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, top);
  pushCtxProcessImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, third);
}
```

CCode : default

```
static void swap23DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, third);
  pushCtxDataImpl(aCtx, top);
}
```

CCode : default

```
static void swap23PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  popCtxProcessIntoImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, top);
}
```

CCode : default

```
static void rollUp3DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  pushCtxDataImpl(aCtx, top);
  pushCtxDataImpl(aCtx, third);
  pushCtxDataImpl(aCtx, second);
}
```

CCode : default

```
static void rollUp3PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
```

```
  popCtxProcessIntoImpl(aCtx, second);
  popCtxProcessIntoImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, top);
  pushCtxProcessImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, second);
}
```

CCode : default

```
static void rollDown3DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, top);
  pushCtxDataImpl(aCtx, third);
}
```

CCode : default

```
static void rollDown3PAP(ContextObj* aCtx) {
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  popCtxProcessIntoImpl(aCtx, third);
  pushCtxProcessImpl(aCtx, second);
  pushCtxProcessImpl(aCtx, top);
  pushCtxProcessImpl(aCtx, third);
}
```

CCode : default

```
static void choiceDAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  if (isTrue(top)) {
    pushCtxDataImpl(aCtx, second);
  } else {
    pushCtxDataImpl(aCtx, third);
  }
}
```

CCode : default

```
static void choicePAP(ContextObj* aCtx) {
```

```
  popCtxProcessIntoImpl(aCtx, top);
  popCtxProcessIntoImpl(aCtx, second);
  popCtxProcessIntoImpl(aCtx, third);
  if (isTrue(top)) {
    pushCtxProcessImpl(aCtx, second);
  } else {
    pushCtxProcessImpl(aCtx, third);
  }
}
```

3.5.9.4 Quaternary operators

CCode : default

```
static void swap14DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  popCtxDataIntoImpl(aCtx, fourth);
  pushCtxDataImpl(aCtx, top);
  pushCtxDataImpl(aCtx, third);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, fourth);
}
```

CCode : default

```
static void swap24DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
  popCtxDataIntoImpl(aCtx, fourth);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, third);
  pushCtxDataImpl(aCtx, fourth);
  pushCtxDataImpl(aCtx, top);
}
```

CCode : default

```
static void swap34DAP(ContextObj* aCtx) {
  popCtxDataIntoImpl(aCtx, top);
  popCtxDataIntoImpl(aCtx, second);
  popCtxDataIntoImpl(aCtx, third);
```

```
  popCtxDataIntoImpl(aCtx, fourth);
  pushCtxDataImpl(aCtx, third);
  pushCtxDataImpl(aCtx, fourth);
  pushCtxDataImpl(aCtx, second);
  pushCtxDataImpl(aCtx, top);
}
```

3.5.9.5 Registering operators

CHeader : private

```
extern Boolean initContextAOOperators(
  JoyLoLInterp *jInterp
);
```

CCode : default

```
Boolean initContextAOOperators(
  JoyLoLInterp *jInterp
) {
  // unary operators
  extendJoyLoLInRoot(jInterp, "pop1D", "", pop1DAP, "");
  extendJoyLoLInRoot(jInterp, "pop1P", "", pop1PAP, "");
  extendJoyLoLInRoot(jInterp, "dup1D", "", dup1DAP, "");
  extendJoyLoLInRoot(jInterp, "dup1P", "", dup1PAP, "");
  //
  // binary operators
  //
  extendJoyLoLInRoot(jInterp, "swap12D", "", swap12DAP, "");
  extendJoyLoLInRoot(jInterp, "swap12P", "", swap12PAP, "");
  extendJoyLoLInRoot(jInterp, "pop2D",   "", pop2DAP,   "");
  extendJoyLoLInRoot(jInterp, "pop2P",   "", pop2PAP,   "");
  extendJoyLoLInRoot(jInterp, "pop12D",  "", pop12DAP,  "");
  extendJoyLoLInRoot(jInterp, "pop12P",  "", pop12PAP,  "");
  extendJoyLoLInRoot(jInterp, "dup2D",   "", dup2DAP,   "");
  extendJoyLoLInRoot(jInterp, "dup2P",   "", dup2PAP,   "");
  //
  // ternary operators
  //
  extendJoyLoLInRoot(jInterp, "swap13D",    "", swap13DAP,   "");
  extendJoyLoLInRoot(jInterp, "swap13P",    "", swap13PAP,   "");
  extendJoyLoLInRoot(jInterp, "swap23D",    "", swap23DAP,   "");
  extendJoyLoLInRoot(jInterp, "swap23P",    "", swap23PAP,   "");
```

```
27    extendJoyLoLInRoot(jInterp, "rollUp3D",   "", rollUp3DAP,   "");
28    extendJoyLoLInRoot(jInterp, "rollUp3P",   "", rollUp3PAP,   "");
29    extendJoyLoLInRoot(jInterp, "rollDown3D", "", rollDown3DAP, "");
30    extendJoyLoLInRoot(jInterp, "rollDown3P", "", rollDown3PAP, "");
31    extendJoyLoLInRoot(jInterp, "choiceD",    "", choiceDAP,   "");
32    extendJoyLoLInRoot(jInterp, "choiceP",    "", choicePAP,   "");
33    //
34    // quarternary operators
35    //
36    extendJoyLoLInRoot(jInterp, "swap14D",   "", swap14DAP,   "");
37    extendJoyLoLInRoot(jInterp, "swap24D",   "", swap24DAP,   "");
38    extendJoyLoLInRoot(jInterp, "swap34D",   "", swap34DAP,   "");
39
40    return TRUE;
41  }
```

3.5.10 JoyLoL words

CHeader : private

```
1   extern Boolean registerContextWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoAlg
4   );
```

CCode : default

```
1   Boolean registerContextWords(
2     JoyLoLInterp *jInterp,
3     JClass       *theCoAlg
4   ) {
5     initContextsAPControl(jInterp);
6     initContextAOOperators(jInterp);
7     initContextsAPSupport(jInterp);
8     return TRUE;
9   }
```

3.5.11 Code

CHeader : public

```
1   typedef struct context_tracing_label {
```

Code 3.5.11

```
  Symbol                       *name;
  size_t                        count;
  struct context_tracing_label *next;
} CtxTracingLabel;

typedef struct context_object_struct ContextObj;
typedef struct context_object_struct {
  JObj             super;
  JoyLoLInterp    *jInterp;
  Symbol          *name;
  ContextObj      *parent;
  JObj            *data;
  JObj            *command;
  JObj            *process;
  JObj            *messages;
  JObj            *listeners;
  DictObj         *dict;
  Boolean          exceptionRaised;
  Boolean          showSpecifications;
  Boolean          showStack;
  size_t           showDepth;
  Boolean          tracingOn;
  CtxTracingLabel *tracingLabels;
  Boolean          checkingOn;
  Boolean          verbose;
} ContextObj;
```

CHeader : public

```
typedef ContextObj *(NewContext)(
  JoyLoLInterp *jInterp,
  Symbol       *name,
  ContextObj   *parent,
  DictObj      *dict,
  JObj         *dataLoL,
  JObj         *processLoL
);

#define newContext(jInterp, name, parent,     \
  dict, dataLoL, processLoL)                  \
  (                                           \
    assert(getContextsClass(jInterp)          \
      ->newContextFunc),                      \
    (getContextsClass(jInterp)                \
```

```
16      ->newContextFunc(jInterp, name, parent, \
17        dict, dataLoL, processLoL))           \
18    )
```

CHeader : private

```
1   extern ContextObj* newContextImpl(
2     JoyLoLInterp *jInterp,
3     Symbol       *name,
4     ContextObj   *parent,
5     DictObj      *dict,
6     JObj         *dataLoL,
7     JObj         *processLoL
8   );
```

CCode : default

```
1   ContextObj* newContextImpl(
2     JoyLoLInterp *jInterp,
3     Symbol       *name,
4     ContextObj   *parent,
5     DictObj      *dict,
6     JObj         *dataLoL,
7     JObj         *processLoL
8   ) {
9     DEBUG(jInterp, "newContext %p [%s] %p %p %p %p\n",
10      jInterp, name, parent, dict, dataLoL, processLoL);
11    assert(jInterp);
12
13    ContextObj* context =
14      (ContextObj*) newObject(jInterp, ContextsTag);
15    assert(context);
16
17    context->jInterp            = jInterp;
18    context->name               = name;
19    context->parent             = parent;
20    context->dict               = dict;
21    context->data               = dataLoL;
22    context->command            = NULL;
23    context->process            = processLoL;
24    context->messages           = NULL;
25    context->listeners          = NULL;
26    if (jInterp->quiet) {
27      context->showSpecifications = FALSE;
```

```
    context->showStack          = FALSE;
  } else {
    context->showSpecifications = TRUE;
    context->showStack          = TRUE;
  }
  context->showDepth          = 5;
  context->tracingOn          = FALSE;
  context->tracingLabels      = NULL;
  context->checkingOn         = FALSE;
  context->verbose            = FALSE;
  context->exceptionRaised    = FALSE;
  if (jInterp->tracing) {
    context->tracingOn        = TRUE;
    context->showStack        = TRUE;
  }
  if (jInterp->verbose) {
    context->verbose          = TRUE;
  }

  return context;
}

#define asName(aLoL)      (((ContextObj*)(aLoL))->name)
#define asData(aLoL)      (((ContextObj*)(aLoL))->data)
#define asCommand(aLoL)   (((ContextObj*)(aLoL))->command)
#define asProcess(aLoL)   (((ContextObj*)(aLoL))->process)
#define asMessages(aLoL)  (((ContextObj*)(aLoL))->messages)
#define asListeners(aLoL) (((ContextObj*)(aLoL))->listeners)
```

CHeader : private

```
extern Boolean equalityContextsCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityContextsCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
```

```
  size_t        timeToLive
) {
  DEBUG(jInterp, "contextCoAlg->equal a:%p b:%p\n",
    lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA) != ContextsTag) return FALSE;

  if (timeToLive < 1) return TRUE;
  timeToLive -= 1;

  size_t areEqual = TRUE;
  lolEqual(jInterp, areEqual, asData(lolA),      asData(lolB),      timeToLive);
  lolEqual(jInterp, areEqual, asCommand(lolA),   asCommand(lolB),   timeToLive);
  lolEqual(jInterp, areEqual, asProcess(lolA),   asProcess(lolB),   timeToLive);
  lolEqual(jInterp, areEqual, asMessages(lolA),  asMessages(lolB),  timeToLive);
  lolEqual(jInterp, areEqual, asListeners(lolA), asListeners(lolB), timeToLive);
  return areEqual;
}
```

CHeader : private

```
extern Boolean printContextsCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
static void printContextName(
  StringBufferObj *aStrBuf,
  ContextObj      *aCtx
) {
  assert(aCtx);
  if (aCtx->parent) {
    printContextName(aStrBuf, aCtx->parent);
    strBufPrintf(aStrBuf, ":");
  }
  strBufPrintf(aStrBuf, aCtx->name);
}
```

```
12
13  Boolean printContextsCoAlg(
14    StringBufferObj *aStrBuf,
15    JObj            *aLoL,
16    size_t           timeToLive
17  ) {
18    assert(aStrBuf);
19
20    assert(aLoL);
21    assert(asType(aLoL));
22    assert(asTag(aLoL) == ContextsTag);
23    ContextObj* aCtx = (ContextObj*)aLoL;
24
25    if (timeToLive < 1) {
26      strBufPrintf(aStrBuf, "... ");
27      return TRUE;
28    }
29    timeToLive -= 1;

30    size_t printedOk = TRUE;
31    strBufPrintf(aStrBuf, "\n[");
32    printContextName(aStrBuf, aCtx);
33    strBufPrintf(aStrBuf, "(");
34    printLoL(aStrBuf, ((JObj*)aCtx->dict));
35    lolPrintStr(aStrBuf, printedOk, asData(aLoL),
36                ")[ d:( ", ") ", timeToLive);
37    lolPrintStr(aStrBuf, printedOk, asCommand(aLoL),
38                "c:( ", ") ", timeToLive);
39    lolPrintStr(aStrBuf, printedOk, asProcess(aLoL),
40                "p:( ", ") ", timeToLive);
41    lolPrintStr(aStrBuf, printedOk, asMessages(aLoL),
42                "m:( ", ") ", timeToLive);
43    lolPrintStr(aStrBuf, printedOk, asListeners(aLoL),
44                "l:( ", ") ", timeToLive);
45    strBufPrintf(aStrBuf, " ]]\n");
46    return printedOk;
47  }
```

3.5.11.1 Test Suite: registerContexts

CHeader : public

```
1  typedef struct contexts_class_struct {
```

```
2   JClass           super;
3   NewContext      *newContextFunc;
4   ClearCtx        *clearCtxDataFunc;
5   PushCtx         *pushCtxDataFunc;
6   PushCtx         *pushOnTopCtxDataFunc;
7   PushNullCtx     *pushNullCtxDataFunc;
8   PushBooleanCtx  *pushBooleanCtxDataFunc;
9   PushNaturalCtx  *pushNaturalCtxDataFunc;
10  PushSymbolCtx   *pushSymbolCtxDataFunc;
11  PushParsedArrayOfStringsCtx
12    *pushParsedArrayOfStringsCtxDataFunc;
13  PushParsedStringCtx
14    *pushParsedStringCtxDataFunc;
15  PushParsedTextCtx
16    *pushParsedTextCtxDataFunc;
17  PrependListCtx  *prependListCtxDataFunc;
18  PeekCtx         *peekCtxDataFunc;
19  PopCtx          *popCtxDataFunc;
20  ClearCtx        *clearCtxProcessFunc;
21  PushCtx         *pushCtxProcessFunc;
22  PushNullCtx     *pushNullCtxProcessFunc;
23  PushBooleanCtx  *pushBooleanCtxProcessFunc;
24  PushNaturalCtx  *pushNaturalCtxProcessFunc;
25  PushSymbolCtx   *pushSymbolCtxProcessFunc;
26  PushParsedArrayOfStringsCtx
27    *pushParsedArrayOfStringsCtxProcessFunc;
28  PushParsedStringCtx
29    *pushParsedStringCtxProcessFunc;
30  PushParsedTextCtx
31    *pushParsedTextCtxProcessFunc;
32  PrependListCtx  *prependListCtxProcessFunc;
33  PeekCtx         *peekCtxProcessFunc;
34  PopCtx          *popCtxProcessFunc;
35  ExtendJoyLoL    *extendJoyLoLFunc;
36  ExtendCtxJoyLoL *extendCtxJoyLoLFunc;
37  DefineJoyLoL    *defineJoyLoLFunc;
38  DefineContext   *defineContextFunc;
39  DefineNaming    *defineNamingFunc;
40  RaiseException  *raiseExceptionFunc;
41  ReportException
42    *reportExceptionFunc;
43  EvalCommandInContext
44    *evalCommandInContextFunc;
```

```
  EvalContext     *evalContextFunc;
  EvalTextInContext
    *evalTextInContextFunc;
  EvalArrayOfStringsInContext
    *evalArrayOfStringsInContextFunc;
  EvalStringInContext
    *evalStringInContextFunc;
} ContextsClass;
```

CCode : default

```
static Boolean initializeContexts(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  if (!jInterp->rootCtx) {
    //
    // create the globals dictionary
    //
    DictObj* aDict = newDictionary(jInterp, "globals", NULL);
    DictNodeObj *globals = getSymbolEntry(aDict, "globals");
    assert(globals);
    globals->value = (JObj*)aDict;
    //
    // create the root/main context using the
    // globals dictionary
    //
    jInterp->rootCtx =
      newContext(jInterp, "joylol-main", NULL, aDict, NULL, NULL);
    DictNodeObj *mainCtx = getSymbolEntry(aDict, "main");
    assert(mainCtx);
    mainCtx->value = (JObj*)(jInterp->rootCtx);
  }
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerContexts(
  JoyLoLInterp *jInterp
);
```

CCode : default

```
Boolean registerContexts(
  JoyLoLInterp *jInterp
) {
  ContextsClass* theCoAlg    =
    joyLoLCalloc(1, ContextsClass);
  theCoAlg->super.name             = ContextsName;
  theCoAlg->super.objectSize       = sizeof(ContextObj);
  theCoAlg->super.initializeFunc   = initializeContexts;
  theCoAlg->super.registerFunc     = registerContextWords;
  theCoAlg->super.equalityFunc     = equalityContextsCoAlg;
  theCoAlg->super.printFunc        = printContextsCoAlg;
  theCoAlg->newContextFunc         = newContextImpl;

  theCoAlg->clearCtxDataFunc       = clearCtxDataImpl;
  theCoAlg->pushCtxDataFunc        = pushCtxDataImpl;
  theCoAlg->pushOnTopCtxDataFunc   = pushOnTopCtxDataImpl;
  theCoAlg->pushNullCtxDataFunc    = pushNullCtxDataImpl;
  theCoAlg->pushBooleanCtxDataFunc = pushBooleanCtxDataImpl;
  theCoAlg->pushNaturalCtxDataFunc = pushNaturalCtxDataImpl;
  theCoAlg->pushSymbolCtxDataFunc  = pushSymbolCtxDataImpl;
  theCoAlg->pushParsedArrayOfStringsCtxDataFunc =
    pushParsedArrayOfStringsCtxDataImpl;
  theCoAlg->pushParsedStringCtxDataFunc =
    pushParsedStringCtxDataImpl;
  theCoAlg->pushParsedTextCtxDataFunc =
    pushParsedTextCtxDataImpl;
  theCoAlg->prependListCtxDataFunc = prependListCtxDataImpl;
  theCoAlg->peekCtxDataFunc        = peekCtxDataImpl;
  theCoAlg->popCtxDataFunc         = popCtxDataImpl;

  theCoAlg->clearCtxProcessFunc    = clearCtxProcessImpl;
  theCoAlg->pushCtxProcessFunc     = pushCtxProcessImpl;
  theCoAlg->pushNullCtxProcessFunc = pushNullCtxProcessImpl;
  theCoAlg->pushBooleanCtxProcessFunc =
    pushBooleanCtxProcessImpl;
  theCoAlg->pushNaturalCtxProcessFunc =
    pushNaturalCtxProcessImpl;
  theCoAlg->pushSymbolCtxProcessFunc =
    pushSymbolCtxProcessImpl;
  theCoAlg->pushParsedArrayOfStringsCtxProcessFunc =
    pushParsedArrayOfStringsCtxProcessImpl;
  theCoAlg->pushParsedStringCtxProcessFunc =
```

Code 3.5.11

```
    pushParsedStringCtxProcessImpl;
  theCoAlg->pushParsedTextCtxProcessFunc =
    pushParsedTextCtxProcessImpl;
  theCoAlg->prependListCtxProcessFunc =
    prependListCtxProcessImpl;
  theCoAlg->peekCtxProcessFunc      = peekCtxProcessImpl;
  theCoAlg->popCtxProcessFunc      = popCtxProcessImpl;

  theCoAlg->extendJoyLoLFunc       = extendJoyLoLImpl;
  theCoAlg->extendCtxJoyLoLFunc    = extendCtxJoyLoLImpl;
  theCoAlg->defineJoyLoLFunc       = defineJoyLoLImpl;
  theCoAlg->defineContextFunc      = defineContextImpl;
  theCoAlg->defineNamingFunc       = defineNamingImpl;

  theCoAlg->raiseExceptionFunc     = raiseExceptionImpl;
  theCoAlg->reportExceptionFunc    = reportExceptionImpl;

  theCoAlg->evalCommandInContextFunc =
    evalCommandInContextImpl;
  theCoAlg->evalContextFunc        = evalContextImpl;
  theCoAlg->evalTextInContextFunc  = evalTextInContextImpl;
  theCoAlg->evalArrayOfStringsInContextFunc =
    evalArrayOfStringsInContextImpl;
  theCoAlg->evalStringInContextFunc =
    evalStringInContextImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == ContextsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register contexts

```
  // CTestsSetup has already created a jInterp
  // and run registerContexts

  AssertPtrNotNull(jInterp);
```

```
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getContextsClass(jInterp));
ContextsClass *coAlg = getContextsClass(jInterp);
AssertIntTrue(registerContexts(jInterp));
AssertPtrNotNull(getContextsClass(jInterp));
AssertPtrEquals(getContextsClass(jInterp), coAlg);
AssertIntEquals(
  getContextsClass(jInterp)->super.objectSize,
  sizeof(ContextObj)
)
```

3.5.12 Conclusions

CHeader : public

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/assertions.h>
6  #include <joylol/booleans.h>
7  #include <joylol/naturals.h>
8  #include <joylol/signals.h>
9  #include <joylol/symbols.h>
10 #include <joylol/stringBuffers.h>
11 #include <joylol/cFunctions.h>
12 #include <joylol/pairs.h>
13 #include <joylol/texts.h>
14 #include <joylol/parsers.h>
15 #include <joylol/dictNodes.h>
16 #include <joylol/dictionaries.h>
17 #include <joylol/loaders.h>
18 #include <joylol/contexts.h>
19 #include <joylol/contexts-private.h>
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
```

```
requireLuaModule(lstate, "joylol.booleans");
requireLuaModule(lstate, "joylol.naturals");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.parsers");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireStaticallyLinkedContexts(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.6 Cross compilers

3.6.1 Goals

The Cross Compiler is responsible for managing the cross compilation of the JoyLoL fragments into working code.

3.6.2 Code

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```
typedef struct crossCompiler_object_struct {
  JObj     super;
  Symbol  *type;
  DictObj *dict;
} CrossCompilerObj;

#define asCFunc(aLoL) (((CFunctionObj*)(aLoL))->func)
```

3.6.2.1 Test Suite: newCrossCompiler

CHeader : public

```
typedef CrossCompilerObj* (NewCrossCompiler)(
  JoyLoLInterp *jInterp,
  Symbol       *aType
);

#define newCrossCompiler(jInterp, aType)      \
  (                                           \
    assert(getCrossCompilersClass(jInterp)    \
```

```
9       ->newCrossCompilerFunc),                \
10    (getCrossCompilersClass(jInterp)          \
11      ->newCrossCompilerFunc(jInterp, aType)) \
12  )
13  //#define asCrossCompiler(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
1   extern CrossCompilerObj* newCrossCompilerImpl(
2     JoyLoLInterp *jInterp,
3     Symbol       *aType
4   );
```

CCode : default

```
1   CrossCompilerObj* newCrossCompilerImpl(
2     JoyLoLInterp *jInterp,
3     Symbol       *aType
4   ) {
5     assert(jInterp);
6     assert(jInterp->coAlgs);

7     CrossCompilerObj* result =
8       (CrossCompilerObj*)newObject(jInterp, CrossCompilersTag);
9     assert(result);

10    result->type = strdup(aType);
11    result->dict = newDictionary(jInterp, aType, NULL);

12    result->super.type   = jInterp->coAlgs[CrossCompilersTag];

13    return result;
14  }
```

Test case

should create a new crossCompiler

```
  AssertPtrNotNull(jInterp);

  CrossCompilerObj* aNewCrossCompiler =
    newCrossCompiler(jInterp, "ansiC");
  AssertPtrNotNull(aNewCrossCompiler);
  AssertPtrNotNull(asType(aNewCrossCompiler));
```

```
  AssertIntEquals(asTag(aNewCrossCompiler), CrossCompilersTag);
  AssertIntTrue(isAtom(aNewCrossCompiler));
  AssertIntTrue(isCrossCompiler(aNewCrossCompiler));
  AssertIntFalse(isPair(aNewCrossCompiler));
  AssertPtrNotNull(aNewCrossCompiler->dict);
  AssertIntTrue(isDictionary(aNewCrossCompiler->dict));
```

Test case

print CrossCompiler

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  CrossCompilerObj* aLoL =
    newCrossCompiler(jInterp, "ansiC");
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, (JObj*)aLoL);
  AssertStrEquals(getCString(aStrBuf), "crossCompiler ");
  strBufClose(aStrBuf);
```

3.6.2.2 Test Suite: addImplementation

CHeader : public

```
1  typedef void (AddImplementation)(
2    JoyLoLInterp *jInterp,
3    Symbol       *ccType,
4    Symbol       *wordName,
5    Symbol       *implBody
6  );
7
8  #define addImplementation(jInterp, ccType, wordName, implBody)      \
9    (                                                                 \
10     assert(getCrossCompilersClass(jInterp)                         \
11       ->addImplementationFunc),                                    \
12     (getCrossCompilersClass(jInterp)                               \
13       ->addImplementationFunc(jInterp, ccType, wordName, implBody)) \
14   )
```

CHeader : private

```
extern void addImplementationImpl(
  JoyLoLInterp *jInterp,
  Symbol       *ccType,
  Symbol       *wordName,
  Symbol       *implBody
);
```

CCode : default

```
void addImplementationImpl(
  JoyLoLInterp *jInterp,
  Symbol       *ccType,
  Symbol       *wordName,
  Symbol       *implBody
) {
  assert(jInterp);

//  if (jInterp->debug) {
  if (TRUE) {
    StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
    strBufPrintf(aStrBuf,
      "DEBUG addImplementation crossCompiler: [%s]\n", ccType);
    strBufPrintf(aStrBuf, "Word: [%s]\n", wordName);
    strBufPrintf(aStrBuf, "Body: [%s]\n", implBody);
    jInterp->writeStdOut(jInterp, getCString(aStrBuf));
    strBufClose(aStrBuf);
  }

  assert(jInterp->compilers);

  CrossCompilerObj *theCC = NULL;
  if (strcmp(ccType, AnsicName) == 0) {
    theCC = jInterp->compilers[AnsicCC];
  } else if (strcmp(ccType, AnsicLuaName) == 0) {
    theCC = jInterp->compilers[AnsicLuaCC];
  } else if (strcmp(ccType, PureLuaName) == 0) {
    theCC = jInterp->compilers[PureLuaCC];
  }

  if (theCC) {
    jInterp->writeStdOut(jInterp, "addImplementation found theCC\n");
    checkObj(jInterp, theCC, CrossCompilersTag);
    DictObj *ccDict = theCC->dict;
```

```
32      checkObj(jInterp, ccDict, DictionariesTag);

33      DictNodeObj *entry = getSymbolEntry(ccDict, wordName);
34      assert(entry);

35      checkObj(jInterp, entry, DictNodesTag);

36      jInterp->writeStdOut(jInterp, "addImplementation created symbol\n");
37      ImplementationObj *implementation =
38        newImplementation(jInterp, wordName, implBody);
39      jInterp->writeStdOut(jInterp, "addImplementation created fragment\n");

40      checkObj(jInterp, implementation, ImplementationsTag);

41      entry->value = (JObj*)implementation;
42    } else {
43      // raise error -- NEEDS Context!

44      StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
45      strBufPrintf(aStrBuf,
46        "ERROR(impl) could not find the [%s] cross compiler.\n", ccType);
47      strBufPrintf(aStrBuf, "Word: [%s]\n", wordName);
48      strBufPrintf(aStrBuf, "Body: [%s]\n", implBody);
49      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
50      strBufClose(aStrBuf);
51    }
52    jInterp->writeStdOut(jInterp, "addImplementation DONE\n");
53  }
54
55  static int lua_crossCompilers_addImplementation(lua_State *lstate) {
56    getJoyLoLInterpInto(lstate, jInterp);
57    Symbol *ccType   = luaL_checkstring(lstate, 1);
58    Symbol *wordName = luaL_checkstring(lstate, 2);
59    Symbol *implBody = luaL_checkstring(lstate, 3);
60    addImplementationImpl(jInterp, ccType, wordName, implBody);
61    lua_pop(lstate, 3);
62    return 0;
63  }
```

3.6.2.3 Test Suite: addFragment

CCode : default

Code 3.6.2

```
1   static void addFragmentAP(ContextObj* aCtx) {
2     assert(aCtx);
3     assert(aCtx->jInterp);
4     JoyLoLInterp *jInterp = aCtx->jInterp;
5
6     popCtxDataInto(aCtx, ccTypeLoL);
7     if (!isSymbol(ccTypeLoL)) {
8       raiseExceptionMsg(aCtx,
9         "addFragment expected a symbol as ccType");
10      return;
11    }
12    Symbol *ccType = asSymbol(ccTypeLoL);

13    popCtxDataInto(aCtx, wordNameLoL);
14    if (!isSymbol(wordNameLoL)) {
15      raiseExceptionMsg(aCtx,
16        "addFragment expected a symbol as wordName");
17      return;
18    }
19    Symbol *wordName = asSymbol(wordNameLoL);

20    popCtxDataInto(aCtx, fragmentBodyLoL);
21    if (!isSymbol(fragmentBodyLoL)) {
22      raiseExceptionMsg(aCtx,
23        "addFragment expected a symbol as fragmentBody");
24      return;
25    }
26    Symbol *fragmentBody = asSymbol(fragmentBodyLoL);

27  //  if (jInterp->debug) {
28    if (TRUE) {
29      StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
30      strBufPrintf(aStrBuf,
31        "DEBUG addFragment crossCompiler: [%s]\n", ccType);
32      strBufPrintf(aStrBuf, "Word: [%s]\n", wordName);
33      strBufPrintf(aStrBuf, "Body: [%s]\n", fragmentBody);
34      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
35      strBufClose(aStrBuf);
36    }
37    assert(jInterp->compilers);
38
39    CrossCompilerObj *theCC = NULL;
40    if (strcmp(ccType, AnsicName) == 0) {
```

```
41      theCC = jInterp->compilers[AnsicCC];
42    } else if (strcmp(ccType, AnsicLuaName) == 0) {
43      theCC = jInterp->compilers[AnsicLuaCC];
44    } else if (strcmp(ccType, PureLuaName) == 0) {
45      theCC = jInterp->compilers[PureLuaCC];
46    }
47    if (theCC) {
48      checkObj(jInterp, theCC, CrossCompilersTag);
49      DictObj *ccDict = theCC->dict;
50      checkObj(jInterp, ccDict, DictionariesTag);

51      DictNodeObj *entry = getSymbolEntry(ccDict, wordName);
52      assert(entry);
53      FragmentObj *fragment =
54        newFragment(jInterp, wordName, fragmentBody);
55      entry->value = (JObj*)fragment;
56    } else {
57      // raise error -- NEEDS Context!

58      StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
59      strBufPrintf(aStrBuf,
60        "ERROR could not find the [%s] cross compiler.\n", ccType);
61      strBufPrintf(aStrBuf, "Word: [%s]\n", wordName);
62      strBufPrintf(aStrBuf, "Body: [%s]\n", fragmentBody);
63      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
64      strBufClose(aStrBuf);
65    }
66  }
```

Test case

should add a fragment to the appropriate compiler - ansic

```
pushSymbolCtxProcess(jInterp->rootCtx, "addFragment");
pushSymbolCtxProcess(jInterp->rootCtx, "ansic");
pushSymbolCtxProcess(jInterp->rootCtx, "test-ansic");
pushSymbolCtxProcess(jInterp->rootCtx, "this is a test");
pushSymbolCtxProcess(jInterp->rootCtx, "tracingOn");
pushSymbolCtxProcess(jInterp->rootCtx, "showStack");
evalContext(jInterp->rootCtx);
pushSymbolCtxProcess(jInterp->rootCtx, "showStack");
pushSymbolCtxProcess(jInterp->rootCtx, "tracingOff");
evalContext(jInterp->rootCtx);
```

Code 3.6.2

SKIPPED

CCode : default

```
1  static int lua_crossCompilers_addFragment(lua_State *lstate) {
2    getJoyLoLInterpInto(lstate, jInterp);
3    assert(jInterp);

4    pushSymbolCtxProcess(jInterp->rootCtx, "addFragment");
5
6    Symbol *ccType       = luaL_checkstring(lstate, 1);
7    pushSymbolCtxProcess(jInterp->rootCtx, ccType);

8    Symbol *wordName     = luaL_checkstring(lstate, 2);
9    pushSymbolCtxProcess(jInterp->rootCtx, wordName);

10   Symbol *fragmentBody = luaL_checkstring(lstate, 3);
11   pushSymbolCtxProcess(jInterp->rootCtx, fragmentBody);

12   //evalContext(jInterp->rootCtx);

13   lua_pop(lstate, 3);
14   return 0;
15 }
```

Test case

should add a fragment to the appropriate compiler - lua

```
local joylol       = thirddata.joylol
local ccType       = "ansic"
local wordName     = "test"
local fragmentBody = "this is a test"
joylol.crossCompiler.addFragment(ccType, wordName, fragmentBody)
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
print(joylol.showData)
print(joylol.showProcess)
print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
```

SKIPPED

3.6.2.4 Test Suite: isCrossCompiler

CHeader : public

```
#define isCrossCompiler(aLoL)                \
  (                                          \
    (                                        \
      (aLoL) &&                              \
      asType(aLoL) &&                        \
      (asTag(aLoL) == CrossCompilersTag)     \
    ) ?                                      \
      TRUE :                                 \
      FALSE                                  \
  )
```

CHeader : private

```
extern Boolean equalityCrossCompilerCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityCrossCompilerCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "crossCompilerCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != CrossCompilersTag) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
}
```

3.6.2.5 Test Suite: printing crossCompilers

CHeader : private

```
extern Boolean printCrossCompilerCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printCrossCompilerCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == CrossCompilersTag);

  strBufPrintf(aStrBuf, "crossCompiler ");
  return TRUE;
}
```

Test case

should print crossCompilers

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[CrossCompilersTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  CrossCompilerObj* aNewCrossCompiler =
    newCrossCompiler(jInterp, "ansiC");
  AssertPtrNotNull(aNewCrossCompiler);
  printLoL(aStrBuf, (JObj*)aNewCrossCompiler);
  AssertStrEquals(getCString(aStrBuf), "crossCompiler ");
  strBufClose(aStrBuf);
```

3.6.2.6 Test Suite: registerCrossCompilers

CHeader : public

```
typedef struct crossCompilers_class_struct {
  JClass            super;
  NewCrossCompiler  *newCrossCompilerFunc;
  AddImplementation *addImplementationFunc;
} CrossCompilersClass;
```

CCode : default

```
static Boolean initializeCrossCompilers(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);

  CrossCompilerObj *ansic =
    newCrossCompilerImpl(jInterp, AnsicName);
  registerCrossCompiler(jInterp, ansic);
  CrossCompilerObj *ansicLua =
    newCrossCompilerImpl(jInterp, AnsicLuaName);
  registerCrossCompiler(jInterp, ansicLua);
  CrossCompilerObj *pureLua =
    newCrossCompilerImpl(jInterp, PureLuaName);
  registerCrossCompiler(jInterp, pureLua);

  return TRUE;
}
```

CHeader : private

```
extern Boolean registerCrossCompilers(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerCrossCompilers(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  CrossCompilersClass* theCoAlg
    = joyLoLCalloc(1, CrossCompilersClass);
  assert(theCoAlg);
```

```
7     theCoAlg->super.name             = CrossCompilersName;
8     theCoAlg->super.objectSize       = sizeof(CrossCompilerObj);
9     theCoAlg->super.initializeFunc   = initializeCrossCompilers;
10    theCoAlg->super.registerFunc     = registerCrossCompilerWords;
11    theCoAlg->super.equalityFunc     = equalityCrossCompilerCoAlg;
12    theCoAlg->super.printFunc        = printCrossCompilerCoAlg;
13    theCoAlg->newCrossCompilerFunc   = newCrossCompilerImpl;
14    theCoAlg->addImplementationFunc = addImplementationImpl;
15
16    size_t tag =
17      registerJClass(jInterp, (JClass*)theCoAlg);

18    // do a sanity check...
19    assert(tag == CrossCompilersTag);
20    assert(jInterp->coAlgs[tag]);

21    return TRUE;
22  }
```

Test case

should register the CrossCompilers coAlg

```
// CTestsSetup has already created a jInterp
// and run registerCrossCompilers
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getCrossCompilersClass(jInterp));
CrossCompilersClass *coAlg = getCrossCompilersClass(jInterp);
registerCrossCompilers(jInterp);
AssertPtrNotNull(getCrossCompilersClass(jInterp));
AssertPtrEquals(getCrossCompilersClass(jInterp), coAlg);
AssertIntEquals(
  getCrossCompilersClass(jInterp)->super.objectSize,
  sizeof(CrossCompilerObj)
)
```

3.6.3 ANSI-C cross compiler

3.6.4 Lua cross compiler in an ANSI-C JoyLoL

3.6.5 Lua cross compiler

3.6.6 Words

CHeader : private

```
extern Boolean registerCrossCompilerWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerCrossCompilerWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  extendJoyLoLInRoot(jInterp, "addFragment", "", addFragmentAP, "");

  return TRUE;
}
```

3.6.7 Lua functions

CCode : default

```
static int lua_crossCompilers_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_crossCompilers [] = {
  {"gitVersion",        lua_crossCompilers_getGitVersion},
  {"addImplementation", lua_crossCompilers_addImplementation},
  {"addFragment",       lua_crossCompilers_addFragment},
  {NULL, NULL}
};
```

```

int luaopen_joylol_crossCompilers (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerCrossCompilers(jInterp);
  luaL_newlib(lstate, lua_crossCompilers);
  return 1;
}

```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedCrossCompilers` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedCrossCompilers(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedCrossCompilers(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_crossCompilers(lstate);
  lua_setfield(lstate, -2, "joylol.crossCompilers");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.6.8 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_crossCompilers(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
```

```
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/symbols.h>
6  #include <joylol/stringBuffers.h>
7  #include <joylol/dictNodes.h>
8  #include <joylol/dictionaries.h>
9  #include <joylol/texts.h>
10 #include <joylol/cFunctions.h>
11 #include <joylol/fragments.h>
12 #include <joylol/implementations.h>
13 #include <joylol/assertions.h>
14 #include <joylol/contexts.h>
15 #include <joylol/crossCompilers.h>
16 #include <joylol/crossCompilers-private.h>
17 // dictionary
18 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.fragments");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.symbols");
requireStaticallyLinkedCrossCompilers(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.7 Dictionary Nodes

3.7.1 Goals

DictNodes provide a relatively fast look-up table.

3.7.2 Find

3.7.2.1 Test Suite: findSymbolRecurse

CHeader : public

```
typedef DictNodeObj *(FindSymbolRecurse)(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
);

#define findSymbolRecurse(aDict, anAVLNode, aSymbol)      \
  (                                                       \
    assert(aDict),                                        \
    assert(getDictNodesClass(aDict->jInterp)              \
      ->findSymbolRecurseFunc),                           \
    (getDictNodesClass(aDict->jInterp)                    \
      ->findSymbolRecurseFunc(aDict, anAVLNode, aSymbol)) \
  )
```

CHeader : private

```
extern DictNodeObj* findSymbolRecurseImpl(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
);
```

CCode : default

```
DictNodeObj* findSymbolRecurseImpl(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
) {
  if (!anAVLNode) return NULL;
```

```
7   int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
8   if (aStrCmp < 0) {
9     // aSymbol < anAVLNode->symbol // search the LEFT subtree
10    return findSymbolRecurse(aDict, anAVLNode->left, aSymbol);
11  } else if (0 < aStrCmp) {
12    // aSymbol > anAVLNode->symbol // search the RIGHT subtree
13    return findSymbolRecurse(aDict, anAVLNode->right, aSymbol);
14  } else {
15    // aSymbol == anAVLNode->symbol // return this association pair
16    return anAVLNode;
17  }
18  return NULL;
19 }
```

Test case

should find Symbol In Empty Dictionary

```
AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "test", NULL);

AssertPtrNull(findSymbolRecurse(aDict, NULL, "aSymbol"));
```

Test case

should find Symbol In Non Empty Dictionary

```
AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "test", NULL);

DictNodeObj* aNode = newDictNode(jInterp, "aNodeSymbol");
AssertPtrNotNull(aNode);
DictNodeObj* foundAPair = findSymbolRecurse(aDict, aNode, "aNodeSymbol");
AssertPtrNotNull(foundAPair);
AssertStrEquals(foundAPair->symbol, "aNodeSymbol");
```

Test case

should find Symbol Not In Dictionary

```
AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "test", NULL);
```

```
  DictNodeObj* aSimpleDic = newDictNode(jInterp, "aNodeSymbol");
  AssertPtrNotNull(aSimpleDic);
  AssertPtrNull(findSymbolRecurse(aDict, aSimpleDic, "aSymbol"));
```

3.7.2.2 Test Suite: findLUBSymbolRecurse

CHeader : public

```
1  typedef DictNodeObj *(FindLUBSymbolRecurse)(
2    DictObj     *aDict,
3    DictNodeObj *anAVLNode,
4    Symbol      *aSymbol
5  );
6
7  #define findLUBSymbolRecurse(aDict, anAVLNode, aSymbol)       \
8    (                                                           \
9      assert(aDict),                                            \
10     assert(getDictNodesClass(aDict->jInterp)                  \
11       ->findLUBSymbolRecurseFunc),                            \
12     (getDictNodesClass(aDict->jInterp)                        \
13       ->findLUBSymbolRecurseFunc(aDict, anAVLNode, aSymbol))  \
14   )
```

CHeader : private

```
1  DictNodeObj* findLUBSymbolRecurseImpl(
2    DictObj     *aDict,
3    DictNodeObj *anAVLNode,
4    Symbol      *aSymbol
5  );
```

CCode : default

```
1  DictNodeObj* findLUBSymbolRecurseImpl(
2    DictObj     *aDict,
3    DictNodeObj *anAVLNode,
4    Symbol      *aSymbol
5  ) {
6    assert(aDict);
7    if (!anAVLNode) return aDict->firstSymbol;
8
9    DEBUG(aDict->jInterp,
10         "findLUBSymbol %p {%s}[%s] %p %p\n",
11         anAVLNode, anAVLNode->symbol, aSymbol,
```

```
12          anAVLNode->left, anAVLNode->right);
13
14    int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
15    DEBUG(aDict->jInterp, "findLUBSymbol cmp: %d\n", aStrCmp);
16    if (aStrCmp < 0) {
17      // aSymbol < anAVLNode->symbol
18      // the current anAVLNode->symbol is an upper bound
19      // search the LEFT subtree for a smaller upper bound
20      if (anAVLNode->left) {
21        DictNodeObj* aNode = findLUBSymbolRecurse(aDict, anAVLNode->left,
22  aSymbol);
23        if (!aNode) {
24          // there is nothing in the LEFT subtree which is an upper bound
25          // so return this node.
26          return anAVLNode;
27        }
28        // we have found a smaller upper bound... so return it
29        return aNode;
30      }
31      // there is nothing less than this node so return this node
32      return anAVLNode;
33      //
34    } else if (0 < aStrCmp) {
35      // anAVLNode->symbol < symbol
36      // the current anAVLNode->symbol is a lower bound
37      // search the RIGHT subtree for any upper bounds
38      if (anAVLNode->right) {
39        return findLUBSymbolRecurse(aDict, anAVLNode->right, aSymbol);
40      }
41      // there is nothing greater than this node so return NULL to signal failure
42      return NULL;
43      //
44    } else {
45      // aSymbol == anAVLNode->symbol
46      // the current anAVLNode->symbol is the lowest possible upper bound
47      // return it
48      return anAVLNode;
49      //
50    }
51    return aDict->firstSymbol;
52  }
```

3.7.3 Insert

3.7.3.1 Test Suite: insertSymbolRecurse

CHeader : public

```
typedef DictNodeObj *(InsertSymbolRecurse)(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
);

#define insertSymbolRecurse(aDict, anAVLNode, aSymbol)      \
  (                                                         \
    assert(aDict),                                          \
    assert(getDictNodesClass(aDict->jInterp)                \
      ->insertSymbolRecurseFunc),                           \
    (getDictNodesClass(aDict->jInterp)                      \
      ->insertSymbolRecurseFunc(aDict, anAVLNode, aSymbol)) \
  )
```

CHeader : private

```
extern DictNodeObj* insertSymbolRecurseImpl(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
);
```

CCode : default

```
DictNodeObj* insertSymbolRecurseImpl(
  DictObj     *aDict,
  DictNodeObj *anAVLNode,
  Symbol      *aSymbol
) {
  assert(aDict);
  JoyLoLInterp *jInterp = aDict->jInterp;
  assert(jInterp);

  if (!anAVLNode) return newDictNode(jInterp, aSymbol);

  StringBufferObj *aStrBuf =
    (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);

```

```
14    DEBUG(jInterp, "\ninsertSymbolRecurse %p <%s>[%s] %ld:%zu\n",
15          anAVLNode, anAVLNode->symbol, aSymbol,
16          anAVLNode->balance, anAVLNode->height);
17
18    DEBUG(jInterp, "insertSymbolRecurse strncmp %d\n",
19          strcmp(aSymbol, anAVLNode->symbol));
20
21    int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
22    if (aStrCmp < 0) {
23      // aSymbol < anAVLNode->symbol // insert in LEFT subtree
24      if (jInterp->debug) {
25        printDicInto(aStrBuf, anAVLNode, 10);
26        DEBUG(jInterp, ">-insert LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
27              anAVLNode, aSymbol, anAVLNode->balance,
28              anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
29              getCString(aStrBuf));
30        strBufClose(aStrBuf);
31      }
32      DictNodeObj* leftResult =
33        insertSymbolRecurse(aDict, anAVLNode->left, aSymbol);
34      assert(leftResult);
35      if (!anAVLNode->left) {
36        // we have inserted a new node ...
37        // ... insert this new node into the doubly linked list
38        //
39        DictNodeObj* oldPrevious           = anAVLNode->previous;
40        assert(aDict->firstSymbol);
41        if (oldPrevious) oldPrevious->next = leftResult;
42        else aDict->firstSymbol            = leftResult;
43        leftResult->next                   = anAVLNode;
44        leftResult->previous               = oldPrevious;
45        anAVLNode->previous                = leftResult;
46        //
47      }
48      anAVLNode->left = leftResult;
49      reCalculateAVLNodeHeightBalance(anAVLNode);
50      if (jInterp->debug) {
51        printDicInto(aStrBuf, anAVLNode, 10);
52        DEBUG(jInterp, "<-insert LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
53            anAVLNode, aSymbol, anAVLNode->balance,
54            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
55            getCString(aStrBuf));
56        strBufClose(aStrBuf);
```

```
57      }
58      //
59      if (2 < anAVLNode->balance) {
60        assert(anAVLNode->left);
61        if (strcmp(aSymbol, anAVLNode->left->symbol) < 0) {
62          anAVLNode = rotateLeftLeft(aDict, anAVLNode);
63        } else {
64          anAVLNode = rotateLeftRight(aDict, anAVLNode);
65        }
66      }
67    } else if (0 < aStrCmp) {
68      // aSymbol > anAVLNode->symbol // insert in RIGHT subtree
69      if (jInterp->debug) {
70        printDicInto(aStrBuf, anAVLNode, 10);
71        DEBUG(jInterp, ">-insert RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
72            anAVLNode, aSymbol, anAVLNode->balance,
73            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
74            getCString(aStrBuf));
75        strBufClose(aStrBuf);
76      }
77      DictNodeObj* rightResult =
78        insertSymbolRecurse(aDict, anAVLNode->right, aSymbol);
79      if (!anAVLNode->right) {
80        // we have inserted a new node ...
81        // ... insert this new node into the doubly linked list
82        //
83        DictNodeObj* oldNext           = anAVLNode->next;
84        if (oldNext) oldNext->previous = rightResult;
85        rightResult->previous          = anAVLNode;
86        rightResult->next              = oldNext;
87        anAVLNode->next                = rightResult;
88        //
89      }
90      anAVLNode->right = rightResult;
91      reCalculateAVLNodeHeightBalance(anAVLNode);
92      if (jInterp->debug) {
93        printDicInto(aStrBuf, anAVLNode, 10);
94        DEBUG(jInterp, "<-insert RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
95            anAVLNode, aSymbol, anAVLNode->balance,
96            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
97            getCString(aStrBuf));
98        strBufClose(aStrBuf);
99      }
```

```
100     //
101     if (anAVLNode->balance < -2) {
102       assert(anAVLNode->right);
103       if (strcmp(aSymbol, anAVLNode->right->symbol) > 0) {
104         anAVLNode = rotateRightRight(aDict, anAVLNode);
105       } else {
106         anAVLNode = rotateRightLeft(aDict, anAVLNode);
107       }
108     }
109   } else {
110     // aSymbol == anAVLNode->symbol // nothing to do...
111     DEBUG(jInterp,"symols equal <%s>[%s]\n",
112           anAVLNode->symbol, aSymbol);
113   }
114
115   reCalculateAVLNodeHeightBalance(anAVLNode);
116   return anAVLNode;
117 }
```

Test case

should insert Symbol In Dictionary

```
AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

DictNodeObj* aSimpleDic = newDictNode(jInterp, "aNodeSymbol");
DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "aNodeSymbol");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[aNodeSymbol] l:(  ) r:(  ) ");
strBufClose(aStrBuf);
AssertPtrNull(aNewDic->next);
AssertPtrNull(aNewDic->previous);
AssertIntTrue(aDict->firstSymbol != aNewDic);
AssertIntTrue(aDict->firstSymbol != aSimpleDic);
```

Test case

should insert Symbol Not In Dictionary

```
DEBUG(jInterp, "\n%s insert many symbols %s", "---------", "---------");

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp, "20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "15");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  )  ) r:(  ) ");
strBufClose(aStrBuf);
//
// test insertion on the left into doubly linked list
//
AssertPtrNotNull(aNewDic->previous);
AssertPtrNull(aNewDic->next);
AssertPtrEquals(aDict->firstSymbol, aNewDic->previous);
AssertPtrEquals(aDict->firstSymbol->next, aNewDic);
AssertPtrNull(aDict->firstSymbol->previous);
AssertPtrNull(aNewDic->next);

// should invoke an LL
aNewDic = insertSymbolRecurse(aDict, aNewDic, "10");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);
//
// test insertion on the right into doubly linked list
// as well as a rotate right
//
```

```
AssertPtrNotNull(aNewDic->previous);
AssertPtrNotNull(aNewDic->next);
AssertPtrEquals(aDict->firstSymbol, aNewDic->previous);
AssertPtrEquals(aDict->firstSymbol->next, aNewDic);
AssertStrEquals(aDict->firstSymbol->symbol, "10");
AssertStrEquals(aNewDic->next->symbol, "20");
AssertPtrEquals(aNewDic->next->previous, aNewDic);
AssertPtrNull(aNewDic->next->next);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "30");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:( [30] l:(  ) r:(  )  )  ) ");
strBufClose(aStrBuf);

// should invoke an RR
aNewDic = insertSymbolRecurse(aDict, aNewDic, "35");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [20] l:(  ) r:(  )  ) r:( [35] l:(  ) r:(  )
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "25");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [20] l:(  ) r:( [25] l:(  ) r:(  )  )  ) r:(
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [23] l:( [20] l:(  ) r:(  )  ) r:( [25] l:(
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "22");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[23] l:( [15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:( [22] l:(  ) r:(  )  )  )  ) r
```

```
strBufClose(aStrBuf);

// now try and find all of the symbols...
DictNodeObj* aNode = findSymbolRecurse(aDict, aNewDic, "15");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "15");

aNode = findSymbolRecurse(aDict, aNewDic, "20");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "20");

aNode = findSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "23");

checkAVLNode(jInterp, aNewDic);
```

Test case

should insert Symbol LL

```
DEBUG(jInterp, "\n%s should invoke an LL %s\n", "---------", "---------");

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp,"20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "15");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  )  ) r:(  ) ");
strBufClose(aStrBuf);

// should invoke an LL
aNewDic = insertSymbolRecurse(aDict, aNewDic, "10");
```

```
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);
```

Test case

should insert Symbol RR

```
DEBUG(jInterp, "\n%s should invoke an RR %s\n", "---------", "---------");

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp,"20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "25");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:(  ) r:( [25] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);

// should invoke an RR
aNewDic = insertSymbolRecurse(aDict, aNewDic, "30");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[25] l:( [20] l:(  ) r:(  )  ) r:( [30] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "35");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[25] l:( [20] l:(  ) r:(  )  ) r:( [30] l:(  ) r:( [35] l:(  ) r:(  )  )  ) ");
```

```
strBufClose(aStrBuf);
```

Test case

should insert Symbol LR

```
DEBUG(jInterp, "\n%s should invoke an LR %s\n", "---------", "---------");

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp,"20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "15");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  )  ) r:(  ) ");
strBufClose(aStrBuf);

// should invoke an LR
aNewDic = insertSymbolRecurse(aDict, aNewDic, "17");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[17] l:( [15] l:(  ) r:(  )  ) r:( [20] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);
```

Test case

should insert Symbol RL

```
DEBUG(jInterp, "\n%s should invoke an RL %s\n", "---------", "---------");

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);
```

```
StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp,"20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "25");
AssertPtrNotNull(aNewDic);
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:(  ) r:( [25] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);

// should invoke an RL
aNewDic = insertSymbolRecurse(aDict, aNewDic, "22");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[22] l:( [20] l:(  ) r:(  )  ) r:( [25] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);
```

Test case

should insert Symbol randomly

```
srand(time(NULL));

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

DictNodeObj* avlDic = newDictNode(jInterp,"0");
aDict->root         = avlDic;
aDict->firstSymbol  = avlDic;

for (int i = 0; i < 1000; i++) {
  char itoa[100];
  sprintf(itoa, "%03d", rand() % 100);
  avlDic = insertSymbolRecurse(aDict, avlDic, itoa);
}
DictNodeObj *aNode = findSymbolRecurse(aDict, avlDic, "0");
if (!aNode) avlDic = insertSymbolRecurse(aDict, avlDic, "0");
```

```
  aNode = findSymbolRecurse(aDict, avlDic, "100");
  if (!aNode) avlDic = insertSymbolRecurse(aDict, avlDic, "100");
  checkAVLNode(jInterp, avlDic);

  aNode = findSymbolRecurse(aDict, avlDic, "0");
  AssertPtrNotNull(aNode);
  AssertStrEquals(aNode->symbol, "0");
  AssertPtrEquals(aDict->firstSymbol, aNode);

  aNode = findSymbolRecurse(aDict, avlDic, "100");
  AssertPtrNotNull(aNode);
  AssertStrEquals(aNode->symbol, "100");
  aNode = aDict->firstSymbol;
  while (aNode->next) aNode = aNode->next;
  AssertStrEquals(aNode->symbol, "100");
  checkAVLNode(jInterp, avlDic);

//  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
//  printDictInto(aStrBuf, avlDic);
//  printf("%s\n", getCString(jInterp, aStrBuf));
//  printf("avl node height: %zu\n", deepCalculateAVLNodeHeight(avlDic));
//  printf("avl node height: %zu\n", avlDic->height);
//  printf("avl node balance: %d\n", avlDic->balance);
```

Test case

should insert Symbol linearly

```
  AssertPtrNotNull(jInterp);
  DictObj *aDict = newDictionary(jInterp, "tests", NULL);

  DictNodeObj* avlDic = newDictNode(jInterp,"10");
  aDict->root         = avlDic;
  aDict->firstSymbol  = avlDic;

  for (int i = 10; i < 100; i++) {
    char itoa[100];
    sprintf(itoa, "%03d", i);
    avlDic = insertSymbolRecurse(aDict, avlDic, itoa);
  }
  checkAVLNode(jInterp, avlDic);

//  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
```

```
//  printDictInto(aStrBuf, avlDic);
//  printf("%s\n", getCString(jInterp, aStrBuf));
//  printf("avl node height: %zu\n", deepCalculateAVLNodeHeight(avlDic));
//  printf("avl node height: %zu\n", avlDic->height);
//  printf("avl node balance: %d\n", avlDic->balance);

  AssertIntEquals(avlDic->height, deepCalculateAVLNodeHeight(avlDic));
  AssertIntEquals(avlDic->height, 8);
  AssertIntEquals(avlDic->balance, -2);
```

3.7.4 Delete

3.7.4.1 Test Suite: deleteSymbol

CHeader : public

```
1  typedef DictNodeObj *(DeleteSymbolRecurse)(
2    DictObj     *aDict,
3    DictNodeObj *anAVLNode,
4    Symbol      *aSymbol
5  );
6
7  #define deleteSymbolRecurse(aDict, anAVLNode, aSymbol)      \
8    (                                                         \
9      assert(aDict),                                          \
10     assert(getDictNodesClass(aDict->jInterp)                \
11       ->deleteSymbolRecurseFunc),                           \
12     (getDictNodesClass(aDict->jInterp)                      \
13       ->deleteSymbolRecurseFunc(aDict, anAVLNode, aSymbol)) \
14   )
```

CHeader : private

```
1  extern DictNodeObj* deleteSymbolRecurseImpl(
2    DictObj     *aDict,
3    DictNodeObj *anAVLNode,
4    Symbol      *aSymbol
5  );
```

CCode : default

```
1  DictNodeObj* deleteSymbolRecurseImpl(
2    DictObj     *aDict,
```

```
3     DictNodeObj *anAVLNode,
4     Symbol      *aSymbol
5   ) {
6     assert(aDict);
7     JoyLoLInterp *jInterp = aDict->jInterp;
8     assert(jInterp);

9     if (!anAVLNode) return NULL;
10
11    StringBufferObj *aStrBuf =
12      (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
13
14    DEBUG(jInterp, "\ndeleteSymbol %p <%s>[%s] %ld:%zu\n",
15          anAVLNode, anAVLNode->symbol, aSymbol,
16          anAVLNode->balance, anAVLNode->height);
17
18    DEBUG(jInterp, "deleteSymbol strncmp %d\n",
19          strcmp(aSymbol, anAVLNode->symbol));
20
21    int aStrCmp = strcmp(aSymbol, anAVLNode->symbol);
22    if (aStrCmp < 0) {
23      // aSymbol < anAVLNode->symbol // delete from LEFT subtree
24      if (jInterp->debug) {
25        printDicInto(aStrBuf, anAVLNode, 10);
26        DEBUG(jInterp, ">-delete LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
27              anAVLNode, aSymbol, anAVLNode->balance,
28              anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
29              getCString(aStrBuf));
30        strBufClose(aStrBuf);
31      }
32
33      anAVLNode->left =
34        deleteSymbolRecurse(aDict, anAVLNode->left, aSymbol);
35      reCalculateAVLNodeHeightBalance(anAVLNode);
36      if (jInterp->debug) {
37        printDicInto(aStrBuf, anAVLNode, 10);
38        DEBUG(jInterp, "<-delete LEFT subtree %p [%s] %ld:%zu=%zu %s\n",
39            anAVLNode, aSymbol, anAVLNode->balance,
40            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
41            getCString(aStrBuf));
42        strBufClose(aStrBuf);
43      }
44      //
```

```
45      if (anAVLNode->balance < -2) {
46        if (
47          anAVLNode->right &&
48          strcmp(aSymbol, anAVLNode->right->symbol) < 0
49        ) {
50          anAVLNode = rotateRightRight(aDict, anAVLNode);
51        } else {
52          anAVLNode = rotateRightLeft(aDict, anAVLNode);
53        }
54      }
55    } else if (0 < aStrCmp) {
56      // aSymbol > anAVLNode->symbol // delete in RIGHT subtree
57      if (jInterp->debug) {
58        printDicInto(aStrBuf, anAVLNode, 10);
59        DEBUG(jInterp, ">-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
60            anAVLNode, aSymbol, anAVLNode->balance,
61            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
62            getCString(aStrBuf));
63        strBufClose(aStrBuf);
64      }
65      anAVLNode->right =
66        deleteSymbolRecurse(aDict, anAVLNode->right, aSymbol);
67      reCalculateAVLNodeHeightBalance(anAVLNode);
68      if (jInterp->debug) {
69        printDicInto(aStrBuf, anAVLNode, 10);
70        DEBUG(jInterp, "<-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
71            anAVLNode, aSymbol, anAVLNode->balance,
72            anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
73            getCString(aStrBuf));
74        strBufClose(aStrBuf);
75      }
76      //
77      if (2 < anAVLNode->balance) {
78        if (
79          anAVLNode->left &&
80          strcmp(aSymbol, anAVLNode->left->symbol) > 0
81        ) {
82          anAVLNode = rotateLeftLeft(aDict, anAVLNode);
83        } else {
84          anAVLNode = rotateLeftRight(aDict, anAVLNode);
85        }
86      }
87    } else {
```

```
88      //
89      // aSymbol == anAVLNode->symbol
90      //
91      DEBUG(jInterp,"symols equal <%s>[%s]\n",
92            anAVLNode->symbol, aSymbol);
93      if (anAVLNode->right) {
94        //
95        // we need to find the next node greater than this one (gtNode)
96        // copy it(gtNode) to this node
97        // and then delete it(gtNode) from right branch of this node
98        //
99        // SINCE we are copying nodes, the doubly linked list is still
100       // correct... except that it has a duplicate entry...
101       // so long as we have a right node... this duplication propogates
102       // to the right...
103       //
104       DictNodeObj *gtNode = anAVLNode->right;
105       while ( gtNode->left ) {
106         gtNode = gtNode->left;
107       }
108       copyDictNodeFromTo(jInterp, gtNode, anAVLNode);
109       anAVLNode->right =
110         deleteSymbolRecurse(aDict, anAVLNode->right, anAVLNode->symbol);
111       reCalculateAVLNodeHeightBalance(anAVLNode);
112       //
113       if (jInterp->debug) {
114         printDicInto(aStrBuf, anAVLNode, 10);
115         DEBUG(jInterp, "<-delete RIGHT subtree %p [%s] %ld:%zu=%zu %s\n",
116             anAVLNode, aSymbol, anAVLNode->balance,
117             anAVLNode->height, deepCalculateAVLNodeHeight(anAVLNode),
118             getCString(aStrBuf));
119         strBufClose(aStrBuf);
120       }
121       //
122       if (2 < anAVLNode->balance) {
123         if (
124           anAVLNode->left &&
125           strcmp(aSymbol, anAVLNode->left->symbol) > 0
126           ) {
127           anAVLNode = rotateLeftLeft(aDict, anAVLNode);
128         } else {
129           anAVLNode = rotateLeftRight(aDict, anAVLNode);
130         }
```

```
131       }
132     } else {
133       //
134       // this node will become unlinked from the AVL tree
135       // SO unlink this node from the doubly linked list
136       // as well...
137       //
138       DictNodeObj* oldPrevious           = anAVLNode->previous;
139       if (oldPrevious) oldPrevious->next = anAVLNode->next;
140       else aDict->firstSymbol            = anAVLNode->next;
141       if (anAVLNode->next) {
142         anAVLNode->next->previous        = oldPrevious;
143       }
144       //
145       // unlink this node..
146       //
147       anAVLNode->next     = NULL;
148       anAVLNode->previous = NULL;
149       //
150       return anAVLNode->left;
151     }
152   }
153
154   reCalculateAVLNodeHeightBalance(anAVLNode);
155   return anAVLNode;
156 }
```

Test case

should delete Symbols from simple Dictionary

```
AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

DictNodeObj* aSimpleDic = newDictNode(jInterp, "20");
aDict->root             = aSimpleDic;
aDict->firstSymbol      = aSimpleDic;

DictNodeObj* aNewDic = insertSymbolRecurse(aDict, aSimpleDic, "15");
AssertPtrNotNull(aNewDic);
```

```
AssertPtrEquals(aSimpleDic, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  )  ) r:(  ) ");
strBufClose(aStrBuf);
//
// test insertion on the left into doubly linked list
//
AssertPtrNotNull(aNewDic->previous);
AssertPtrNull(aNewDic->next);
AssertPtrEquals(aDict->firstSymbol, aNewDic->previous);
AssertPtrEquals(aDict->firstSymbol->next, aNewDic);
AssertPtrNull(aDict->firstSymbol->previous);
AssertPtrNull(aNewDic->next);

// should invoke an LL
aNewDic = insertSymbolRecurse(aDict, aNewDic, "10");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:(  )  ) ");
strBufClose(aStrBuf);
//
// test insertion on the right into doubly linked list
// as well as a rotate right
//
AssertPtrNotNull(aNewDic->previous);
AssertPtrNotNull(aNewDic->next);
AssertPtrEquals(aDict->firstSymbol, aNewDic->previous);
AssertPtrEquals(aDict->firstSymbol->next, aNewDic);
AssertStrEquals(aDict->firstSymbol->symbol, "10");
AssertStrEquals(aNewDic->next->symbol, "20");
AssertPtrEquals(aNewDic->next->previous, aNewDic);
AssertPtrNull(aNewDic->next->next);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "30");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:( [30] l:(  ) r:(  )  )  ) ");
strBufClose(aStrBuf);

// should invoke an RR
```

```
aNewDic = insertSymbolRecurse(aDict, aNewDic, "35");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [20] l:(  ) r:(  )  ) r:( [35] l:(  ) r:(  )
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "25");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [20] l:(  ) r:( [25] l:(  ) r:(  )  )  ) r:(
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[15] l:( [10] l:(  ) r:(  )  ) r:( [30] l:( [23] l:( [20] l:(  ) r:(  )  ) r:( [25] l:(
strBufClose(aStrBuf);

aNewDic = insertSymbolRecurse(aDict, aNewDic, "22");
AssertPtrNotNull(aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[23] l:( [15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:( [22] l:(  ) r:(  )  )  )  ) r
strBufClose(aStrBuf);

// now try and find all of the symbols...
DictNodeObj* aNode = findSymbolRecurse(aDict, aNewDic, "15");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "15");

aNode = findSymbolRecurse(aDict, aNewDic, "20");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "20");

aNode = findSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "23");

checkAVLNode(jInterp, aNewDic);
```

```
// now delete a symbol which is NOT in the dictionary
aNode = findSymbolRecurse(aDict, aNewDic, "21");
AssertPtrNull(aNode);

aNode = deleteSymbolRecurse(aDict, aNewDic, "21");
AssertPtrEquals(aNode, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[23] l:( [15] l:( [10] l:(  ) r:(  )  ) r:( [20] l:(  ) r:( [22] l:(  ) r:(  )  )  )  ) r
strBufClose(aStrBuf);

// now try and find all of the symbols...
aNode = findSymbolRecurse(aDict, aNewDic, "15");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "15");

aNode = findSymbolRecurse(aDict, aNewDic, "20");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "20");

aNode = findSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "23");

checkAVLNode(jInterp, aNewDic);

// now delete a symbol which IS in the dictionary
aNode = findSymbolRecurse(aDict, aNewDic, "15");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "15");

aNode = deleteSymbolRecurse(aDict, aNewDic, "15");
AssertPtrEquals(aNode, aNewDic);
printDicInto(aStrBuf, aNewDic, 10);
AssertStrEquals(getCString(aStrBuf),
"[23] l:( [20] l:( [10] l:(  ) r:(  )  ) r:( [22] l:(  ) r:(  )  )  ) r:( [30] l:( [25] l
strBufClose(aStrBuf);

// now try and find all of the symbols...
aNode = findSymbolRecurse(aDict, aNewDic, "15");
AssertPtrNull(aNode);

aNode = findSymbolRecurse(aDict, aNewDic, "20");
```

```
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "20");

aNode = findSymbolRecurse(aDict, aNewDic, "23");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "23");

checkAVLNode(jInterp, aNewDic);
```

Test case

should randomly delete Symbols from randomly built dictionary

```
srand(time(NULL));

AssertPtrNotNull(jInterp);
DictObj *aDict = newDictionary(jInterp, "tests", NULL);

DictNodeObj* avlDic = newDictNode(jInterp,"000");
aDict->root         = avlDic;
aDict->firstSymbol  = avlDic;

for (int i = 0; i < 1000; i++) {
  char itoa[100];
  sprintf(itoa, "%03d", rand() % 100);
  avlDic = insertSymbolRecurse(aDict, avlDic, itoa);
}
avlDic = insertSymbolRecurse(aDict, avlDic, "000");
avlDic = insertSymbolRecurse(aDict, avlDic, "100");
checkAVLNode(jInterp, avlDic);

DictNodeObj *aNode = findSymbolRecurse(aDict, avlDic, "000");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "000");
AssertPtrEquals(aDict->firstSymbol, aNode);

aNode = findSymbolRecurse(aDict, avlDic, "100");
AssertPtrNotNull(aNode);
AssertStrEquals(aNode->symbol, "100");
aNode = aDict->firstSymbol;
while (aNode->next) aNode = aNode->next;
AssertStrEquals(aNode->symbol, "100");
```

```
  for (int i = 0; i < 1000; i++) {
    char itoa[100];
    int randNum = rand() % 100;
    //
    if (randNum == 0 || randNum == 100) continue;
    //
    sprintf(itoa, "%03d", randNum);
    avlDic = deleteSymbolRecurse(aDict, avlDic, itoa);
    DictNodeObj *aNode = aDict->firstSymbol;
    AssertStrEquals(aNode->symbol, "000");
    while (aNode->next) {
      AssertStrNotEquals(aNode->symbol, itoa);
      aNode = aNode->next;
    }
    AssertStrEquals(aNode->symbol, "100");
    while (aNode->previous) {
      AssertStrNotEquals(aNode->symbol, itoa);
      aNode = aNode->previous;
    }
    AssertStrEquals(aNode->symbol, "000");
    checkAVLNode(jInterp, avlDic);
  }

//  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
//  printDictInto(avlDic);
//  printf("%s\n", getCString(jInterp, aStrBuf));
//  printf("avl node height: %zu\n", deepCalculateAVLNodeHeight(avlDic));
//  printf("avl node height: %zu\n", avlDic->height);
//  printf("avl node balance: %d\n", avlDic->balance);
```

3.7.5 Rotate

CHeader : private

```
1 extern DictNodeObj* rotateLeft(
2   DictObj     *aDict,
3   DictNodeObj *anAVLNode
4 );
```

CCode : default

```
1 DictNodeObj* rotateLeft(
2   DictObj     *aDict,
```

Rotate 3.7.5

```
  DictNodeObj *anAVLNode
) {
  assert(aDict);
  JoyLoLInterp *jInterp = aDict->jInterp;
  assert(jInterp);

  StringBufferObj *aStrBuf =
    (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, ">-rotateLeft %p %ld:%zu=%zu %s\n",
          anAVLNode, anAVLNode->balance, anAVLNode->height,
          deepCalculateAVLNodeHeight(anAVLNode),
          getCString(aStrBuf));
    strBufClose(aStrBuf);
  }
  assert(anAVLNode->right);

  DictNodeObj* newRoot = anAVLNode->right;
  anAVLNode->right = newRoot->left;
  newRoot->left    = anAVLNode;

  reCalculateAVLNodeHeightBalance(anAVLNode);
  reCalculateAVLNodeHeightBalance(newRoot);

  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, "<o-rotateLeft %p %ld:%zu=%zu %s\n",
          anAVLNode, anAVLNode->balance, anAVLNode->height,
          deepCalculateAVLNodeHeight(anAVLNode),
          getCString(aStrBuf));
    strBufClose(aStrBuf);
    printDicInto(aStrBuf, newRoot, 10);
    DEBUG(jInterp, "<n-rotateLeft %p %ld:%zu=%zu %s\n",
          newRoot, newRoot->balance, newRoot->height,
          deepCalculateAVLNodeHeight(newRoot),
          getCString(aStrBuf));
    strBufClose(aStrBuf);
  }
  assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
  assert(newRoot->height == deepCalculateAVLNodeHeight(newRoot));
  return newRoot;
```

```
}
```

CHeader : private

```
extern DictNodeObj* rotateRight(
  DictObj     *aDict,
  DictNodeObj *anAVLNode
);
```

CCode : default

```
DictNodeObj* rotateRight(
  DictObj     *aDict,
  DictNodeObj *anAVLNode
) {
  assert(aDict);
  JoyLoLInterp *jInterp = aDict->jInterp;
  assert(jInterp);

  StringBufferObj *aStrBuf =
    (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, ">-rotateRight %p %ld:%zu=%zu %s\n",
          anAVLNode, anAVLNode->balance, anAVLNode->height,
          deepCalculateAVLNodeHeight(anAVLNode),
          getCString(aStrBuf));
    strBufClose(aStrBuf);
  }
  assert(anAVLNode->left);

  DictNodeObj* newRoot = anAVLNode->left;
  anAVLNode->left = newRoot->right;
  newRoot->right  = anAVLNode;

  reCalculateAVLNodeHeightBalance(anAVLNode);
  reCalculateAVLNodeHeightBalance(newRoot);

  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, "<o-rotateRight %p %ld:%zu=%zu %s\n",
          anAVLNode, anAVLNode->balance, anAVLNode->height,
          deepCalculateAVLNodeHeight(anAVLNode),
          getCString(aStrBuf));
```

```
33      strBufClose(aStrBuf);
34      printDicInto(aStrBuf, newRoot, 10);
35      DEBUG(jInterp, "<n-rotateRight %p %ld:%zu=%zu %s\n",
36            newRoot, newRoot->balance, newRoot->height,
37            deepCalculateAVLNodeHeight(newRoot),
38            getCString(aStrBuf));
39      strBufClose(aStrBuf);
40    }
41    assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
42    assert(newRoot->height == deepCalculateAVLNodeHeight(newRoot));
43    return newRoot;
44  }
```

CHeader : private

```
1   extern DictNodeObj* rotateLeftLeft(
2     DictObj     *aDict,
3     DictNodeObj *anAVLNode
4   );
```

CCode : default

```
1   DictNodeObj* rotateLeftLeft(
2     DictObj     *aDict,
3     DictNodeObj *anAVLNode
4   ) {
5     assert(aDict);
6     JoyLoLInterp *jInterp = aDict->jInterp;
7     assert(jInterp);

8     if (jInterp->debug) {
9       StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
10      printDicInto(aStrBuf, anAVLNode, 10);
11      DEBUG(jInterp, "LL %p %s\n",
12        anAVLNode, getCString(aStrBuf));
13      strBufClose(aStrBuf);
14    }
15    return rotateRight(aDict, anAVLNode);
16  }
```

CHeader : private

```
1   extern DictNodeObj* rotateLeftRight(
2     DictObj     *aDict,
3     DictNodeObj *anAVLNode
```

```
);
```

CCode : default

```
DictNodeObj* rotateLeftRight(
  DictObj     *aDict,
  DictNodeObj *anAVLNode
) {
  assert(aDict);
  JoyLoLInterp *jInterp = aDict->jInterp;
  assert(jInterp);

  StringBufferObj *aStrBuf =
    (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, "0-LR %p %s\n",
      anAVLNode, getCString(aStrBuf));
    strBufClose(aStrBuf);
  }
  anAVLNode->left = rotateLeft(aDict, anAVLNode->left);
  if (jInterp->debug) {
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, "1-LR %p %s\n",
      anAVLNode, getCString(aStrBuf));
    strBufClose(aStrBuf);
  }
  return rotateRight(aDict, anAVLNode);
}
```

CHeader : private

```
extern DictNodeObj* rotateRightLeft(
  DictObj     *aDict,
  DictNodeObj *anAVLNode
);
```

CCode : default

```
DictNodeObj* rotateRightLeft(
  DictObj     *aDict,
  DictNodeObj *anAVLNode
) {
  assert(aDict);
  JoyLoLInterp *jInterp = aDict->jInterp;
```

```
7     assert(jInterp);

8     StringBufferObj *aStrBuf =
9       (jInterp->debug ? newStringBuffer(jInterp->rootCtx) : NULL);
10    if (jInterp->debug) {
11      printDicInto(aStrBuf, anAVLNode, 10);
12      DEBUG(jInterp, "0-RL %p %s\n",
13        anAVLNode, getCString(aStrBuf));
14      strBufClose(aStrBuf);
15    }
16    anAVLNode->right = rotateRight(aDict, anAVLNode->right);
17    if (jInterp->debug) {
18      printDicInto(aStrBuf, anAVLNode, 10);
19      DEBUG(jInterp, "1-RL %p %s\n",
20        anAVLNode, getCString(aStrBuf));
21      strBufClose(aStrBuf);
22    }
23    return rotateLeft(aDict, anAVLNode);
24  }
```

CHeader : private

```
1   extern DictNodeObj* rotateRightRight(
2     DictObj     *aDict,
3     DictNodeObj *anAVLNode
4   );
```

CCode : default

```
1   DictNodeObj* rotateRightRight(
2     DictObj     *aDict,
3     DictNodeObj *anAVLNode
4   ) {
5     assert(aDict);
6     JoyLoLInterp *jInterp = aDict->jInterp;
7     assert(jInterp);

8     if (jInterp->debug) {
9       StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
10      printDicInto(aStrBuf, anAVLNode, 10);
11      DEBUG(jInterp, "RR %p %s\n",
12        anAVLNode, getCString(aStrBuf));
13      strBufClose(aStrBuf);
14    }
```

```
  return rotateLeft(aDict, anAVLNode);
}
```

3.7.6 Check

3.7.6.1 Test Suite: reCalculateAVLNodeHeight

CHeader : private

```
extern void reCalculateAVLNodeHeightBalance(DictNodeObj* anAVLNode);
```

CCode : default

```
void reCalculateAVLNodeHeightBalance(DictNodeObj* anAVLNode) {
  if (!anAVLNode) return;

  if (!anAVLNode->left && !anAVLNode->right) {
    anAVLNode->height  = 1;
    anAVLNode->balance = 0;
  } else if (!anAVLNode->left) {
    anAVLNode->height  =  1 + anAVLNode->right->height;
    anAVLNode->balance = -1 - anAVLNode->right->height;
  } else if (!anAVLNode->right) {
    anAVLNode->height  = 1 + anAVLNode->left->height;
    anAVLNode->balance = 1 + anAVLNode->left->height;
  } else if (anAVLNode->left->height < anAVLNode->right->height) {
    anAVLNode->height  = 1 + anAVLNode->right->height;
    anAVLNode->balance = anAVLNode->left->height - anAVLNode->right->height;
  } else {
    anAVLNode->height  = 1 + anAVLNode->left->height;
    anAVLNode->balance = anAVLNode->left->height - anAVLNode->right->height;
  }
}
```

Test case

should computer correct AVLNode heights

```
  AssertPtrNotNull(jInterp);

  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);
```

Check 3.7.6

```
DictNodeObj* aNode = newDictNode(jInterp, "20");
AssertPtrNotNull(aNode);
printDicInto(aStrBuf, aNode, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:(  ) r:(  ) ");
strBufClose(aStrBuf);

AssertIntEquals(aNode->height, 1);
reCalculateAVLNodeHeightBalance(aNode);
AssertIntEquals(aNode->height, 1);
AssertIntEquals(aNode->balance, 0);

aNode->left = newDictNode(jInterp, "15");
printDicInto(aStrBuf, aNode, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  ) ) r:(  ) ");
strBufClose(aStrBuf);
reCalculateAVLNodeHeightBalance(aNode);
AssertIntEquals(aNode->height, 2);
AssertIntEquals(aNode->balance, 2);

aNode->right = newDictNode(jInterp, "25");
printDicInto(aStrBuf, aNode, 10);
AssertStrEquals(getCString(aStrBuf),
"[20] l:( [15] l:(  ) r:(  ) ) r:( [25] l:(  ) r:(  ) ) ");
strBufClose(aStrBuf);
reCalculateAVLNodeHeightBalance(aNode);
AssertIntEquals(aNode->height, 2);
AssertIntEquals(aNode->balance, 0);
```

CHeader : private

```
1  extern size_t deepCalculateAVLNodeHeight(DictNodeObj* anAVLNode);
```

CCode : default

```
1  size_t deepCalculateAVLNodeHeight(DictNodeObj* anAVLNode) {
2    if (!anAVLNode) return 0;
3
4    size_t leftHeight = 1 + deepCalculateAVLNodeHeight(anAVLNode->left);
5    size_t rightHeight = 1 + deepCalculateAVLNodeHeight(anAVLNode->right);
6
7    if (leftHeight > rightHeight) return leftHeight;
```

```
  return rightHeight;
}
```

CHeader : private

```
extern Boolean checkAVLNode(
  JoyLoLInterp *jInterp,
  DictNodeObj  *anAVLNode
);
```

CCode : default

```
Boolean checkAVLNode(
  JoyLoLInterp *jInterp,
  DictNodeObj  *anAVLNode
) {
  assert(jInterp);

  if (!anAVLNode) return TRUE;
  if (jInterp->debug) {
    StringBufferObj *aStrBuf =
      newStringBuffer(jInterp->rootCtx);
    printDicInto(aStrBuf, anAVLNode, 10);
    DEBUG(jInterp, "checkAVLNode %p %ld:%zu=%zu %s\n",
          anAVLNode, anAVLNode->balance, anAVLNode->height,
          deepCalculateAVLNodeHeight(anAVLNode),
          getCString(aStrBuf));
    strBufClose(aStrBuf);
  }

  if (anAVLNode->left) {
      DEBUG(jInterp, "car>-checkAVLNode %p\n", anAVLNode);
      checkAVLNode(jInterp, anAVLNode->left);
      assert(0 < strcmp(anAVLNode->symbol,
                        anAVLNode->left->symbol));
    DEBUG(jInterp, "car<-checkAVLNode %p\n", anAVLNode);
  }

  if (anAVLNode->right) {
    DEBUG(jInterp, "cdr>-checkAVLNode %p\n", anAVLNode);
    checkAVLNode(jInterp, anAVLNode->right);
    assert(strcmp(anAVLNode->symbol,
                  anAVLNode->right->symbol) < 0);
    DEBUG(jInterp, "cdr<-checkAVLNode %p\n", anAVLNode);
```

```
32    }
33
34    assert(anAVLNode->height == deepCalculateAVLNodeHeight(anAVLNode));
35
36    return TRUE;
37  }
```

3.7.7 Lua interface

CCode : default

```
1   static const KeyValues gitVersionKeyValues[] = {
2     { "authorName",      "Stephen Gaito"},
3     { "commitDate",      "2018-12-03"},
4     { "commitShortHash", "38e0564"},
5     { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
6     { "subject",         "updated textadept lexer for JoyLoL"},
7     { "notes",           ""},
8     { NULL,                NULL}
9   };
```

CCode : default

```
1   static int lua_dictNodes_getGitVersion (lua_State *lstate) {
2     const char* aKey   = lua_tostring(lstate, 1);
3     if (aKey) {
4       getGitVersionInto(gitVersionKeyValues, aKey, aValue);
5       lua_pushstring(lstate, aValue);
6     } else {
7       lua_pushstring(lstate, "no valid key provided");
8     }
9     return 1;
10  }
11
12  static const struct luaL_Reg lua_dictNodes [] = {
13    {"gitVersion", lua_dictNodes_getGitVersion},
14    {NULL, NULL}
15  };
16
17  int luaopen_joylol_dictNodes (lua_State *lstate) {
18    getJoyLoLInterpInto(lstate, jInterp);
19    registerDictNodes(jInterp);
20    luaL_newlib(lstate, lua_dictNodes);
```

```
21    return 1;
22  }
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedDictNodes` does just this.

CHeader : public

```
1  Boolean requireStaticallyLinkedDictNodes(
2    lua_State *lstate
3  );
```

CCode : default

```
1  Boolean requireStaticallyLinkedDictNodes(
2    lua_State *lstate
3  ) {
4    lua_getglobal(lstate, "package");
5    lua_getfield(lstate, -1, "loaded");
6    luaopen_joylol_dictNodes(lstate);
7    lua_setfield(lstate, -2, "joylol.dictNodes");
8    lua_setfield(lstate, -2, "loaded");
9    lua_pop(lstate, 1);
10   return TRUE;
11 }
```

3.7.8 JoyLoL words

CHeader : private

```
1  extern Boolean registerDictNodeWords(
2    JoyLoLInterp *jInterp,
3    JClass       *theCoAlg
4  );
```

CCode : default

```
1  Boolean registerDictNodeWords(
2    JoyLoLInterp *jInterp,
3    JClass       *theCoAlg
4  ) {
5    return TRUE;
6  }
```

3.7.9 Code

CHeader : public

```
typedef struct dictNode_object_struct DictNodeObj;

typedef struct dictNode_object_struct {
  JObj         super;
  Symbol      *symbol;
  JObj        *preObs;
  JObj        *value;
  JObj        *postObs;
  DictNodeObj *left;
  DictNodeObj *right;
  DictNodeObj *previous;
  DictNodeObj *next;
  size_t       height;
  long         balance;
} DictNodeObj;
```

3.7.9.1 Test Suite: copyDictNodeFromTo

CHeader : public

```
typedef void CopyDictNodeFromTo(
  DictNodeObj *fromNode,
  DictNodeObj *toNode
);

#define copyDictNodeFromTo(jInterp, fromNode, toNode) \
  (                                                   \
    assert(getDictNodesClass(jInterp)                 \
      ->copyDictNodeFromToFunc),                      \
    (getDictNodesClass(jInterp)                       \
      ->copyDictNodeFromToFunc(fromNode, toNode))     \
  )
```

CHeader : private

```
extern void copyDictNodeFromToImpl(
  DictNodeObj *fromNode,
  DictNodeObj *toNode
);
```

CCode : default

```
void copyDictNodeFromToImpl(
  DictNodeObj *fromNode,
  DictNodeObj *toNode
) {
  assert(fromNode);
  assert(toNode);
  toNode->super.flags = fromNode->super.flags;
  toNode->symbol      = fromNode->symbol;
  toNode->preObs      = fromNode->preObs;
  toNode->value       = fromNode->value;
  toNode->postObs     = fromNode->postObs;
}
```

3.7.9.2 Test Suite: newDict

CHeader : public

```
typedef DictNodeObj *(NewDictNode)(
  JoyLoLInterp *jInterp,
  Symbol       *aSym
);

#define newDictNode(jInterp, aSym)      \
  (                                     \
    assert(getDictNodesClass(jInterp)   \
      ->newDictNodeFunc),               \
    (getDictNodesClass(jInterp)         \
      ->newDictNodeFunc(jInterp, aSym)) \
  )
```

CHeader : private

```
extern DictNodeObj *newDictNodeImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aSym
);

#define dictAsSymbol(aNode)   ((DictNodeObj*)(aNode))->symbol
#define dictAsPreObs(aNode)   ((DictNodeObj*)(aNode))->preObs
#define dictAsValue(aNode)    ((DictNodeObj*)(aNode))->value
#define dictAsPostObs(aNode)  ((DictNodeObj*)(aNode))->postObs
#define dictAsLeft(aNode)     ((DictNodeObj*)(aNode))->left
```

Code 3.7.9

```
#define dictAsRight(aNode)    ((DictNodeObj*)(aNode))->right
#define dictAsPrevious(aNode) ((DictNodeObj*)(aNode))->previous
#define dictAsNext(aNode)     ((DictNodeObj*)(aNode))->next
#define dictAsHeight(aNode)   ((DictNodeObj*)(aNode))->height
#define dictAsBalance(aNode)  ((DictNodeObj*)(aNode))->balance
```

CCode : default

```
// We implement our dictionary as an AVL binary tree using AVLNodes.
//
// Our implementation is inspired by:
// The Crazy Programmer's "Program for AVL Tree in C" (Neeraj Mishra)
// http://www.thecrazyprogrammer.com/2014/03/c-program-for-avl-tree-implementation.html
// and by:
// Jianye Hao's CSC2100B Tutorial 4 "Binary and AVL Trees in C"
// https://www.cse.cuhk.edu.hk/irwin.king/_media/teaching/csc2100b/tu4.pdf
//
// At the moment we only insert and search (we never delete).
//
// ANY AVLTree node can be the root of a new dictionary.
//

DictNodeObj *newDictNodeImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aSym
) {
  assert(jInterp);
  assert(jInterp->coAlgs);
  assert(DictNodesTag < jInterp->numCoAlgs);
  assert(jInterp->coAlgs[DictNodesTag]);

  assert(aSym);

  DEBUG(jInterp, "newDictNode [%s]\n", aSym);
  JObj* newNode   = newObject(jInterp, DictNodesTag);
  dictAsSymbol(newNode)   = strdup(aSym);
  dictAsPreObs(newNode)   = NULL;
  dictAsValue(newNode)    = NULL;
  dictAsPostObs(newNode)  = NULL;
  dictAsLeft(newNode)     = NULL;
  dictAsRight(newNode)    = NULL;
  dictAsPrevious(newNode) = NULL;
  dictAsNext(newNode)     = NULL;
  dictAsHeight(newNode)   = 1;
```

```
35    dictAsBalance(newNode)  = 0;
36    return (DictNodeObj*)newNode;
37  }
```

Test case

should add a new dictNode object (AVL node)

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[DictNodesTag]);

  DictNodeObj* aNode = newDictNode(jInterp, "aNodeSymbol");
  AssertPtrNotNull(aNode);
  AssertPtrNotNull(asType(aNode));
  AssertIntEquals(asTag(aNode), DictNodesTag)
  AssertPtrNotNull(dictAsSymbol(aNode));
  AssertStrEquals(dictAsSymbol(aNode), "aNodeSymbol");
  AssertPtrNull(dictAsValue(aNode));
  AssertPtrNull(dictAsLeft(aNode));
  AssertPtrNull(dictAsRight(aNode));
  AssertPtrNull(dictAsPrevious(aNode));
  AssertPtrNull(dictAsNext(aNode));
  AssertIntEquals(dictAsHeight(aNode), 1);
  AssertIntEquals(dictAsBalance(aNode), 0);
```

CHeader : private

```
1   extern Boolean printDicionaryJObjInto(
2     StringBufferObj *aStrBuf,
3     JObj            *anAVLNode,
4     size_t           timeToLive
5   );
6   extern Boolean printDicInto(
7     StringBufferObj *aStrBuf,
8     DictNodeObj     *anAVLNode,
9     size_t           timeToLive
10  );
```

CCode : default

```
1   Boolean printDictionaryJObjInto(
2     StringBufferObj *aStrBuf,
3     JObj            *anAVLNode,
```

```
  size_t           timeToLive
) {
  if (!anAVLNode) return FALSE;
  if (asTag(anAVLNode) != DictNodesTag) return FALSE;
  return printDicInto(
    aStrBuf,
    (DictNodeObj*)anAVLNode,
    timeToLive
  );
}

Boolean printDicInto(
  StringBufferObj *aStrBuf,
  DictNodeObj     *anAVLNode,
  size_t           timeToLive
) {
  if (!anAVLNode) return TRUE;
  if (timeToLive < 1) {
    strBufPrintf(aStrBuf, "... ");
    return TRUE;
  }
  timeToLive -= 1;

  strBufPrintf(aStrBuf, "[%s] l:( ", anAVLNode->symbol);
  printDicInto(aStrBuf, anAVLNode->left, timeToLive);
  strBufPrintf(aStrBuf, " ) r:( ");
  printDicInto(aStrBuf, anAVLNode->right, timeToLive);
  strBufPrintf(aStrBuf, " ) ");
  return TRUE;
}
```

3.7.9.3 Test Suite: registerDictNodes

CHeader : public

```
typedef struct dictNodes_class_struct {
  JClass super;
  NewDictNode          *newDictNodeFunc;
  CopyDictNodeFromTo   *copyDictNodeFromToFunc;
  FindSymbolRecurse    *findSymbolRecurseFunc;
  InsertSymbolRecurse  *insertSymbolRecurseFunc;
  DeleteSymbolRecurse  *deleteSymbolRecurseFunc;
  FindLUBSymbolRecurse *findLUBSymbolRecurseFunc;
```

```
} DictNodesClass;
```

CCode : default

```
static Boolean initializeDictNodes(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerDictNodes(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerDictNodes(JoyLoLInterp *jInterp) {
  assert(jInterp);

  DictNodesClass* theCoAlg =
    joyLoLCalloc(1, DictNodesClass);
  assert(theCoAlg);

  theCoAlg->super.name              = DictNodesName;
  theCoAlg->super.objectSize        = sizeof(DictNodeObj);
  theCoAlg->super.initializeFunc    = initializeDictNodes;
  theCoAlg->super.registerFunc      = registerDictNodeWords;
  theCoAlg->super.equalityFunc      = NULL;
  theCoAlg->super.printFunc         = printDictionaryJObjInto;
  theCoAlg->newDictNodeFunc         = newDictNodeImpl;
  theCoAlg->copyDictNodeFromToFunc  = copyDictNodeFromToImpl;
  theCoAlg->findSymbolRecurseFunc   = findSymbolRecurseImpl;
  theCoAlg->insertSymbolRecurseFunc =
    insertSymbolRecurseImpl;
  theCoAlg->deleteSymbolRecurseFunc =
    deleteSymbolRecurseImpl;
  theCoAlg->findLUBSymbolRecurseFunc =
    findLUBSymbolRecurseImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);
```

```
  // do a sanity check...
  assert(tag == DictNodesTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the DictNodes coAlg

```
  // CTestsSetup has already created a jInterp
  // and run regiserDictNodes
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(getDictNodesClass(jInterp));
  DictNodesClass *coAlg =
    getDictNodesClass(jInterp);
  AssertIntTrue(registerDictNodes(jInterp));
  AssertPtrNotNull(getDictNodesClass(jInterp));
  AssertPtrEquals(getDictNodesClass(jInterp), coAlg);
  AssertIntEquals(
    getDictNodesClass(jInterp)->super.objectSize,
    sizeof(DictNodeObj)
  );
```

3.7.10 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/cFunctions.h>
#include <joylol/stringBuffers.h>
#include <joylol/symbols.h>
#include <joylol/texts.h>
#include <joylol/assertions.h>
#include <joylol/dictionaries.h>
#include <joylol/dictNodes.h>
```

```
12 #include <joylol/contexts.h>
13 #include <joylol/dictNodes-private.h>
14 // dictionary
15 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.dictionaries");
requireStaticallyLinkedDictNodes(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.8 Dictionaries

3.8.1 Goals

A Dictionary....

3.8.2 Code

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```c
typedef struct dictionary_object_struct {
  JObj          super;
  JoyLoLInterp *jInterp;
  Symbol       *name;
  DictObj      *parent;
  DictNodeObj  *root;
  DictNodeObj  *firstSymbol;
} DictObj;

#define asCFunc(aLoL) (((CFunctionObj*)(aLoL))->func)
```

3.8.2.1 Test Suite: newDictionary

CHeader : public

```c
typedef DictObj* (NewDictionary)(
  JoyLoLInterp *jInterp,
  Symbol       *name,
  DictObj      *parent
);
```

Code 3.8.2

```
#define newDictionary(jInterp, name, parent)       \
  (                                                \
    assert(getDictionariesClass(jInterp)           \
      ->newDictionaryFunc),                        \
    (getDictionariesClass(jInterp)                 \
      ->newDictionaryFunc(jInterp, name, parent)) \
  )
```

CHeader : private

```
extern DictObj* newDictionaryImpl(
  JoyLoLInterp *jInterp,
  Symbol       *name,
  DictObj      *parent
);
```

CCode : default

```
DictObj* newDictionaryImpl(
  JoyLoLInterp *jInterp,
  Symbol       *name,
  DictObj      *parent
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  DictObj* result =
    (DictObj*)newObject(jInterp, DictionariesTag);
  result->jInterp     = jInterp;
  result->name        = strdup(name);
  result->parent      = parent;
  result->root        = NULL;
  result->firstSymbol = NULL;
  assert(result);

  result->super.type  = jInterp->coAlgs[DictionariesTag];

  return result;
}
```

Test case

should create a new Dictionary

```
  AssertPtrNotNull(jInterp);
```

```
JObj* aNewDictionary = newDictionary(jInterp, "aName", NULL);
AssertPtrNotNull(aNewDictionary);
AssertPtrEquals(aNewDictionary->jInterp, jInterp);
AssertStrEquals(aNewDictionary->name, "aName");
AssertPtrNull(aNewDictionary->parent);
AssertPtrNull(aNewDictionary->root);
AssertPrtNull(aNewDictionary->firstSymbol);
AssertPtrNotNull(asType(aNewDictionary));
AssertIntEquals(asTag(aNewDictionary), DictionariesTag);
AssertIntTrue(isAtom(aNewDictionary));
AssertIntTrue(isDictionary(aNewDictionary));
AssertIntFalse(isPair(aNewDictionary));
```

CHeader : public

```
#define isDict(aLoL)                    \
  (                                     \
    (                                   \
      (aLoL) &&                         \
      asType(aLoL) &&                   \
      (asTag(aLoL) == DictionariesTag)  \
    ) ?                                 \
      TRUE :                            \
      FALSE                             \
  )
```

3.8.2.2 Test Suite: findSymbol

CHeader : private

```
typedef DictNodeObj* (FindSymbol)(
  DictObj *aDict,
  Symbol  *aSymbol
);

extern DictNodeObj* findSymbolInThisDictionary(
  DictObj *aDict,
  Symbol  *aSymbol
);

extern DictNodeObj* findSymbol(
  DictObj *aDict,
  Symbol  *aSymbol
```

```
);
```

CCode : default

```
DictNodeObj* findSymbolInThisDictionary(
  DictObj *aDict,
  Symbol  *aSymbol
) {
  if (!aSymbol) return NULL;
  assert(aDict);
  return findSymbolRecurse(aDict, aDict->root, aSymbol);
}

DictNodeObj* findSymbol(
  DictObj *aDict,
  Symbol  *aSymbol
) {
  if (!aSymbol) return NULL;
  while (aDict) {
    //
    // Look for this symbol in this naming scope
    //
    DictNodeObj *aDictNode =
      findSymbolRecurse(aDict, aDict->root, aSymbol);
    if (aDictNode) return aDictNode;
    //
    // We have not found this symbol in this naming scope
    // so look in the parent's naming scope
    //
    aDict = aDict->parent;
  }
  //
  // Alas, we have not found this symbol in any naming scope
  //
  return NULL;
}
```

Test case

should find symbols in parent dictionary

```
  AssertPtrNotNull(jInterp);

  DictObj* parentDict = newDictionary(jInterp, "parent", NULL);
```

```
AssertPtrNotNull(parentDict);
AssertPtrNull(parentDict->parent);
DictObj* childDict  = newDictionary(jInterp, "child", parentDict);
AssertPtrNotNull(childDict);
AssertPtrNotNull(childDict->parent);
AssertPtrEquals(childDict->parent, parentDict);
//
DictNodeObj* parentSym = findSymbol(childDict, "test");
AssertPtrNull(parentSym);
//
parentSym =
  createSymbolInThisDictionary(parentDict, "test");
AssertPtrNotNull(parentSym);
AssertStrEquals(parentSym->symbol, "test");
//
DictNodeObj* testSym = findSymbol(childDict, "test");
AssertPtrNotNull(testSym);
AssertPtrEquals(testSym, parentSym);
//
DictNodeObj* childSym =
  createSymbolInThisDictionary(childDict, "test");
AssertPtrNotNull(childSym);
AssertPtrNotEquals(parentSym, childSym);
AssertStrEquals(childSym->symbol, "test");
//
testSym = findSymbol(childDict, "test");
AssertPtrNotNull(testSym);
AssertPtrEquals(testSym, childSym);
AssertPtrNotEquals(testSym, parentSym);
//
testSym = findSymbol(parentDict, "test");
AssertPtrNotNull(testSym);
AssertPtrNotEquals(testSym, childSym);
AssertPtrEquals(testSym, parentSym);
//
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);
printLoL(aStrBuf, (JObj*)childDict);
AssertStrEquals(getCString(aStrBuf), "dict:parent.child ");
strBufClose(aStrBuf);
```

3.8.2.3 Test Suite: insertSymbol

CCode : default

```
1  static DictNodeObj* insertSymbol(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  ) {
5    assert(aDict);
6    assert(aSymbol);
7
8    // lazy initialization
9    if (!aDict->root) {
10     assert(aDict->jInterp);
11     DictNodeObj* firstNode = newDictNode(aDict->jInterp, aSymbol);
12     aDict->root            = firstNode;
13     aDict->firstSymbol     = firstNode;
14     return firstNode;
15   }
16
17   return insertSymbolRecurse(aDict, aDict->root, aSymbol);
18 }
```

3.8.2.4 Test Suite: deleteSymbol

CHeader : public

```
1  typedef void (DeleteSymbol)(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  );
5
6  #define deleteSymbol(aDict, aSymbol)            \
7    (                                             \
8      assert(aDict->jInterp),                     \
9      assert(getDictionariesClass(aDict->jInterp) \
10       ->deleteSymbolFunc),                      \
11     (getDictionariesClass(aDict->jInterp)       \
12       ->deleteSymbolFunc(aDict, aSymbol))       \
13   )
```

CHeader : private

```
1  extern void deleteSymbolImpl(
2    DictObj *aDict,
```

```
  Symbol  *aSymbol
);
```

CCode : default

```
void deleteSymbolImpl(
  DictObj *aDict,
  Symbol  *aSymbol
) {
  if (!aSymbol) return;
  assert(aDict);
  //
  // Look for the first dot
  //
  char* symPrefix = strdup(aSymbol);
  char* restOfSym = strchr(symPrefix, '.');
  //
  if (!restOfSym) {
    //
    // no dot found so this is the end of the recursion
    // just delete the symbol
    //
    aDict->root =
      deleteSymbolRecurse(aDict, aDict->root, aSymbol);
    free(symPrefix); // this is *our* duplicate of aSymbol
    return;
  }
  //
  // a dot has been found...
  // so split the symbol into two a the first dot...
  //
  *restOfSym = 0; // terminate the symPrefix at the dot
  restOfSym++;    // move onto the next char
  //
  // now look for the prefix dictionary
  // start by looking in the dictionary provided
  //
  // there is NO dot in the prefix...
  // so just simply find the symbol
  //
  DictNodeObj* aDictSym = findSymbol(aDict, symPrefix);
  if (!aDictSym || !isDictionary(aDictSym->value)) {
    //
    // there is no prefix dictionary
```

Code 3.8.2

```
    // there is nothing to delete...
    // so free our duplicate of aSymbol
    // and return
    //
    free(symPrefix);
    return;
  }
  //
  // a prefix dictionary HAS been found
  // so setup the prefix dictionary
  //
  DictObj* prefixDict = (DictObj*)(aDictSym->value);
  //
  // and delete the restOfSym from the prefixDict
  //
  deleteSymbolImpl(prefixDict, restOfSym);
  //
  free(symPrefix); // this is *our* duplicate of aSymbol
}
```

Test case

should delete random symbols from a randomly created dictionary

```
  srand(time(NULL));

  AssertPtrNotNull(jInterp);
  DictObj *aDict = newDictionary(jInterp, "tests", NULL);

  for (int i = 0; i < 1000; i++) {
    char itoa[100];
    sprintf(itoa, "%03d", rand() % 100);
    createSymbolInThisDictionary(aDict, itoa);
  }
  createSymbolInThisDictionary(aDict, "000");
  createSymbolInThisDictionary(aDict, "100");

  DictNodeObj *aNode = findSymbol(aDict, "000");
  AssertPtrNotNull(aNode);
  AssertStrEquals(aNode->symbol, "000");
  AssertPtrEquals(aDict->firstSymbol, aNode);

  aNode = findSymbol(aDict, "100");
```

```
  AssertPtrNotNull(aNode);
  AssertStrEquals(aNode->symbol, "100");
  aNode = aDict->firstSymbol;
  while (aNode->next) aNode = aNode->next;
  AssertStrEquals(aNode->symbol, "100");

  for (int i = 0; i < 1000; i++) {
    char itoa[100];
    int randNum = rand() % 100;
    //
    if (randNum == 0 || randNum == 100) continue;
    //
    sprintf(itoa, "%03d", randNum);
    deleteSymbol(aDict, itoa);
    DictNodeObj *aNode = aDict->firstSymbol;
    AssertStrEquals(aNode->symbol, "000");
    while (aNode->next) {
      AssertStrNotEquals(aNode->symbol, itoa);
      aNode = aNode->next;
    }
    AssertStrEquals(aNode->symbol, "100");
    while (aNode->previous) {
      AssertStrNotEquals(aNode->symbol, itoa);
      aNode = aNode->previous;
    }
    AssertStrEquals(aNode->symbol, "000");
  }
```

3.8.2.5 Test Suite: findLUBSymbol

CHeader : public

```
1  typedef DictNodeObj *(FindLUBSymbol)(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  );
5
6  #define findLUBSymbol(aDict, aSymbol)           \
7    (                                             \
8      assert(aDict),                              \
9      assert(getDictionariesClass(aDict->jInterp) \
10       ->findLUBSymbolFunc),                     \
11     (getDictionariesClass(aDict->jInterp)       \
```

```
12      ->findLUBSymbolFunc(aDict, aSymbol))      \
13    )
```

CHeader : private

```
1  extern DictNodeObj* findLUBSymbolImpl(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  );
```

CCode : default

```
1  DictNodeObj* findLUBSymbolImpl(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  ) {
5    assert(aDict);
6    if (!aSymbol) return aDict->firstSymbol;
7    return findLUBSymbolRecurse(aDict, aDict->root, aSymbol);
8  }
```

3.8.2.6 Test Suite: createSymbolInThisDictionary

CHeader : private

```
1  extern DictNodeObj* createSymbolInThisDictionary(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  );
```

CCode : default

```
1  DictNodeObj* createSymbolInThisDictionary(
2    DictObj *aDict,
3    Symbol  *aSymbol
4  ) {
5    assert(aDict);
6    if (!aSymbol) return NULL;
7    DictNodeObj* aSym =
8      findSymbolInThisDictionary(aDict, aSymbol);
9    if (!aSym) {
10     aDict->root = insertSymbol(aDict, aSymbol);
11     aSym = findSymbolInThisDictionary(aDict, aSymbol);
12   }
13   return aSym;
```

```
}
```

3.8.2.7 Test Suite: getSymbolEntry

CCode : default

```
DictNodeObj* walkEntryPath(
  DictObj    *aDict,
  Symbol     *aSymbol,
  FindSymbol *symbolFinder
) {
  assert(aDict);
  assert(aSymbol);
  DEBUG(aDict->jInterp,
    "walkEntryPath: [%s](%p) [%s] %p\n",
    aDict->name, aDict, aSymbol, symbolFinder
  );
  //
  // Look for the first dot
  //
  char* symPrefix = strdup(aSymbol);
  char* restOfSym = strchr(symPrefix, '.');
  //
  if (!restOfSym) {
    DEBUG(aDict->jInterp, "no dot found in [%s]\n", aSymbol);
    //
    // no dot found so this is the end of the recursion
    // just find the symbol or create one
    //
    DictNodeObj* aSym = symbolFinder(aDict, aSymbol);
    if (!aSym) {
      DEBUG(aDict->jInterp,
        "creating new entry in [%s](%p) using [%s]\n",
        aDict->name, aDict, aSymbol
      );
      aSym = createSymbolInThisDictionary(aDict, aSymbol);
    }
    free(symPrefix); // this is *our* duplicate of aSymbol
    DEBUG(aDict->jInterp,
      "using symbol [%s](%p) value: %p\n",
      (aSym ? aSym->symbol : ""),
      aSym,
      (aSym ? aSym->value : NULL)
```

```
38      );
39      return aSym;
40    }
41    //
42    // a dot has been found...
43    // so split the symbol into two at the first dot...
44    //
45    *restOfSym = 0; // terminate the symPrefix at the dot
46    restOfSym++;    // move onto the next char
47    DEBUG(aDict->jInterp,
48      "dot found in [%s] splitting into [%s] and [%s]\n",
49      aSymbol, symPrefix, restOfSym
50    );
51
52    //
53    // setup the prefix dictionary
54    //
55    DictObj* prefixDict = NULL;
56    //
57    // now look for the prefix dictionary
58    // start by looking in the dictionary provided
59    //
60    DictNodeObj *aDictSym =
61      walkEntryPath(aDict, symPrefix, symbolFinder);
62    assert(aDictSym);
63    //
64    if (!aDictSym->value) {
65      //
66      // no dictionary found...
67      // create a new one...
68      // and use it as the prefix Dictionary
69      //
70      // WHICH PARENT DICTIONARY SHOULD WE USE?
71      //
72      prefixDict =
73        newDictionary(aDict->jInterp, symPrefix, aDict);
74      aDictSym->value = (JObj*)prefixDict;
75      DEBUG(aDict->jInterp,
76        "no dictionary found in [%s](%p) created new dictionary: [%s](%p)\n",
77        symPrefix, aDict, prefixDict->name, prefixDict
78      );
79      //
80    } else {
```

```
81      //
82      // The prefix entry has been found...
83      //
84      if (isDictionary(aDictSym->value)) {
85        //
86        // the symbol IS a dictionary
87        // so use it as the prefix dictionary
88        //
89        prefixDict = (DictObj*)(aDictSym->value);
90        DEBUG(aDict->jInterp,
91          "prefix dictionary found: [%s](%p)\n",
92          prefixDict->name, prefixDict
93        );
94        //
95      } else {
96        //
97        // the symbol is NOT a dictionary...
98        // so replace the dot with an '_'
99        restOfSym--;
100       *restOfSym = '_';
101       // and retry...
102       //
103       // use the original dictionary
104       //
105       prefixDict = aDict;
106       //
107       // and use the modified symbol
108       //
109       restOfSym = symPrefix;
110       DEBUG(aDict->jInterp,
111         "non-dictionary found for prefix [%s]\n", symPrefix
112       );
113       DEBUG(aDict->jInterp,
114         "re-trying with dictionary: [%s](%p) and symbol [%s]\n",
115         prefixDict->name, prefixDict, restOfSym
116       );
117     }
118   }
119   assert(prefixDict);
120   //
121   DictNodeObj* aSym =
122     walkEntryPath(prefixDict, restOfSym, symbolFinder);
123   //
```

```
  free(symPrefix); // this is *our* duplicate of aSymbol
  DEBUG(aDict->jInterp,
    "using symbol [%s](%p) value: %p\n",
    (aSym ? aSym->symbol : ""),
    aSym,
    (aSym ? aSym->value : NULL)
  );

  return aSym;
}
```

CHeader : public

```
typedef DictNodeObj *(GetSymbolEntry)(
  DictObj *aDict,
  Symbol  *aSymbol
);

#define getSymbolEntry(aDict, aSymbol)          \
  (                                             \
    assert(aDict),                              \
    assert(getDictionariesClass(aDict->jInterp) \
      ->getSymbolEntryFunc),                    \
    (getDictionariesClass(aDict->jInterp)       \
      ->getSymbolEntryFunc(aDict, aSymbol))     \
  )
#define getSymbolEntryInChild(aDict, aSymbol)       \
  (                                                 \
    assert(aDict),                                  \
    assert(getDictionariesClass(aDict->jInterp)     \
      ->getSymbolEntryFunc),                        \
    (getDictionariesClass(aDict->jInterp)           \
      ->getSymbolEntryInChildFunc(aDict, aSymbol))  \
  )
```

CHeader : private

```
extern DictNodeObj* getSymbolEntryImpl(
  DictObj *aDict,
  Symbol  *aSymbol
);

extern DictNodeObj* getSymbolEntryInChildImpl(
  DictObj *aDict,
```

```
  Symbol  *aSymbol
);
```

CCode : default

```
DictNodeObj* getSymbolEntryImpl(
  DictObj *aDict,
  Symbol  *aSymbol
) {
  assert(aDict);
  assert(aSymbol);
  DEBUG(aDict->jInterp,
    "getSymbolEntry: %p [%s]\n", aDict, aSymbol
  );
  return walkEntryPath(
    aDict,
    aSymbol,
    findSymbol
  );
}

DictNodeObj* getSymbolEntryInChildImpl(
  DictObj *aDict,
  Symbol  *aSymbol
) {
  assert(aDict);
  assert(aSymbol);
  DEBUG(aDict->jInterp,
    "getSymbolEntryInChild: %p [%s]\n", aDict, aSymbol
  );
  return walkEntryPath(
    aDict,
    aSymbol,
    findSymbolInThisDictionary
  );
}
```

Test case

should get and delete dotted symbols

```
  AssertPtrNotNull(jInterp);

  DictObj* parentDict =
```

Code 3.8.2

```
  newDictionary(jInterp, "parent", NULL);
AssertPtrNotNull(parentDict);
AssertPtrNull(parentDict->parent);
DictNodeObj* parentEntry =
  createSymbolInThisDictionary(parentDict, "parent");
AssertPtrNotNull(parentEntry);
parentEntry->value = (JObj*)parentDict;

DictObj* childDict =
  newDictionary(jInterp, "child", parentDict);
AssertPtrNotNull(childDict);
AssertPtrNotNull(childDict->parent);
AssertPtrEquals(childDict->parent, parentDict);
DictNodeObj* childEntry =
  createSymbolInThisDictionary(parentDict, "child");
AssertPtrNotNull(childEntry);
childEntry->value = (JObj*)childDict;

DictNodeObj* testSym =
  createSymbolInThisDictionary(childDict, "test");
AssertPtrNotNull(testSym);
AssertStrEquals(testSym->symbol, "test");

DictNodeObj* foundSym =
  getSymbolEntry(childDict, "test");
AssertPtrNotNull(foundSym);
AssertPtrEquals(foundSym, testSym);

foundSym = getSymbolEntry(parentDict, "child.test");
AssertPtrNotNull(foundSym);
AssertPtrEquals(foundSym, testSym);

foundSym = getSymbolEntry(childDict, "child.test");
AssertPtrNotNull(foundSym);
AssertPtrEquals(foundSym, testSym);

DictNodeObj* aTestSym =
  getSymbolEntry(parentDict, "assert.test");
AssertPtrNotNull(aTestSym);
AssertPtrNotEquals(aTestSym, testSym);
AssertStrEquals(aTestSym->symbol, "test");

DictNodeObj* aDictSym =
```

```
    getSymbolEntry(childDict, "assert");
  AssertPtrNotNull(aDictSym);
  AssertPtrNotNull(aDictSym->value);
  AssertIntTrue(isDictionary(aDictSym->value));
  DictObj* aDict = (DictObj*)(aDictSym->value);

  foundSym = getSymbolEntry(aDict, "test");
  AssertPtrNotNull(foundSym);
  AssertPtrEquals(foundSym, aTestSym);

  deleteSymbol(childDict, "assert.test");
  foundSym = findSymbol(childDict, "assert");
  AssertPtrNotNull(foundSym);
  AssertPtrEquals(foundSym, aDictSym);
  AssertPtrNotNull(foundSym->value);
  AssertIntTrue(isDictionary(foundSym->value));
  AssertPtrEquals(foundSym->value, (JObj*)aDict);

  foundSym = findSymbol(aDict, "test");
  AssertPtrNull(foundSym);

  foundSym = findSymbol(childDict, "test");
  AssertPtrNotNull(foundSym);
```

3.8.2.8 Test Suite: getAsSymbol

CHeader : public

```
1  typedef JObj *(GetAsSymbol)(
2    DictObj *aDict,
3    Symbol  *aSymbol,
4    Symbol  *fileName,
5    size_t   line
6  );
7
8  #define getAsSymbol(aDict, aSymbol, fileName, line)       \
9    (                                                       \
10     assert(aDict),                                        \
11     assert(getDictionariesClass(aDict->jInterp)           \
12       ->getAsSymbolFunc),                                 \
13     (getDictionariesClass(aDict->jInterp)                 \
14       ->getAsSymbolFunc(aDict, aSymbol, fileName, line))  \
```

```
15    )
```

CHeader : private

```
1  extern JObj* getAsSymbolImpl(
2    DictObj *aDict,
3    Symbol  *aSymbol,
4    Symbol  *fileName,
5    size_t   line
6  );
```

CCode : default

```
1  JObj* getAsSymbolImpl(
2    DictObj *aDict,
3    Symbol  *aSymbol,
4    Symbol  *fileName,
5    size_t   line
6  ) {
7    assert(aDict);
8    DictNodeObj* aSym = getSymbolEntryImpl(aDict, aSymbol);
9    assert(aSym);
10   return newSymbol(aDict->jInterp, aSym->symbol, fileName, 0);
11 }
```

3.8.2.9 Test Suite: listDefinitions

CHeader : public

```
1  typedef void (ListDefinitions)(
2    DictObj         *aDict,
3    StringBufferObj *aStrBuf
4  );
5
6  #define listDefinitions(aDict, aStrBuf)          \
7    (                                              \
8      assert(aDict),                               \
9      assert(getDictionariesClass(aDict->jInterp) \
10       ->listDefinitionsFunc),                    \
11     (getDictionariesClass(aDict->jInterp)        \
12       ->listDefinitionsFunc(aDict, aStrBuf))     \
13   )
```

CHeader : private

```
extern void listDefinitionsImpl(
  DictObj         *aDict,
  StringBufferObj *aStrBuf
);
```

CCode : default

```
void listDefinitionsImpl(
  DictObj         *aDict,
  StringBufferObj *aStrBuf
) {
  assert(aDict);
  DictNodeObj* curNode = aDict->firstSymbol;
  while(curNode) {
    if (curNode->value) {
      strBufPrintf(aStrBuf,"%s == ", curNode->symbol);
      printLoL(aStrBuf, curNode->value);
      strBufPrintf(aStrBuf,"\n");
    }
    curNode = curNode->next;
  }
}
```

Test case

print Dictionary

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  DictObj* aLoL = newDictionary(jInterp, "tests", NULL);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, (JObj*)aLoL);
  AssertStrEquals(getCString(aStrBuf), "dict:tests ");
  strBufClose(aStrBuf);
```

3.8.2.10 Test Suite: isDictionary

CHeader : public

```
#define isDictionary(aLoL)              \
```

```
  (                                       \
    (                                     \
      (aLoL) &&                           \
      asType(aLoL) &&                     \
      (asTag(aLoL) == DictionariesTag)    \
    ) ?                                   \
      TRUE :                              \
      FALSE                               \
  )
```

CHeader : private

```
extern Boolean equalityDictionaryCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityDictionaryCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "dictionaryCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != DictionariesTag) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
}
```

3.8.2.11 Test Suite: printing dictionaries

CHeader : private

```
extern Boolean printDictionaryCoAlg(
  StringBufferObj *aStrBuf,
```

```
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
static void printDictionaryName(
  StringBufferObj *aStrBuf,
  DictObj         *aDict
) {
  assert(aDict);
  if (aDict->parent) {
    printDictionaryName(aStrBuf, aDict->parent);
    strBufPrintf(aStrBuf, ".");
  } else {
    strBufPrintf(aStrBuf, "dict:");
  }
  strBufPrintf(aStrBuf, aDict->name);
}

Boolean printDictionaryCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == DictionariesTag);

  DictObj* theDict = (DictObj*)aLoL;

  printDictionaryName(aStrBuf, theDict);
  strBufPrintf(aStrBuf, " ");

  return TRUE;
}
```

Test case

should print dictionaries

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[DictionariesTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
```

```
  AssertPtrNotNull(aStrBuf);

  DictObj* aNewDictionary = newDictionary(jInterp, "tests", NULL);
  AssertPtrNotNull(aNewDictionary);
  printLoL(aStrBuf, (JObj*)aNewDictionary);
  AssertStrEquals(getCString(aStrBuf), "dict:tests ");
  strBufClose(aStrBuf);
```

3.8.2.12 Test Suite: registerDictionaries

CHeader : public

```
1  typedef struct dictionaries_class_struct {
2    JClass           super;
3    NewDictionary   *newDictionaryFunc;
4    GetSymbolEntry  *getSymbolEntryFunc;
5    GetSymbolEntry  *getSymbolEntryInChildFunc;
6    GetAsSymbol     *getAsSymbolFunc;
7    DeleteSymbol    *deleteSymbolFunc;
8    FindLUBSymbol   *findLUBSymbolFunc;
9    ListDefinitions *listDefinitionsFunc;
10 } DictionariesClass;
```

CCode : default

```
1  static Boolean initializeDictionaries(
2    JoyLoLInterp *jInterp,
3    JClass       *aJClass
4  ) {
5    assert(jInterp);
6    assert(aJClass);
7    return TRUE;
8  }
```

CHeader : private

```
1  extern Boolean registerDictionaries(JoyLoLInterp *jInterp);
```

CCode : default

```
1  Boolean registerDictionaries(JoyLoLInterp *jInterp) {
2    assert(jInterp);
3    assert(jInterp->coAlgs);
```

```
4   DictionariesClass* theCoAlg
5     = joyLoLCalloc(1, DictionariesClass);
6   assert(theCoAlg);

7   theCoAlg->super.name                = DictionariesName;
8   theCoAlg->super.objectSize          = sizeof(DictObj);
9   theCoAlg->super.initializeFunc      = initializeDictionaries;
10  theCoAlg->super.registerFunc        = registerDictionaryWords;
11  theCoAlg->super.equalityFunc        = equalityDictionaryCoAlg;
12  theCoAlg->super.printFunc           = printDictionaryCoAlg;
13  theCoAlg->newDictionaryFunc         = newDictionaryImpl;
14  theCoAlg->getSymbolEntryFunc        = getSymbolEntryImpl;
15  theCoAlg->getSymbolEntryInChildFunc = getSymbolEntryInChildImpl;
16  theCoAlg->getAsSymbolFunc           = getAsSymbolImpl;
17  theCoAlg->deleteSymbolFunc          = deleteSymbolImpl;
18  theCoAlg->findLUBSymbolFunc         = findLUBSymbolImpl;
19  theCoAlg->listDefinitionsFunc       = listDefinitionsImpl;
20  size_t tag =
21    registerJClass(jInterp, (JClass*)theCoAlg);

22  // do a sanity check...
23  assert(tag == DictionariesTag);
24  assert(jInterp->coAlgs[tag]);

25  return TRUE;
26 }
```

Test case

should register the Dictionaries coAlg

```
// CTestsSetup has already created a jInterp
// and run registerDictionaries
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getDictionariesClass(jInterp));
DictionariesClass *coAlg = getDictionariesClass(jInterp);
registerDictionaries(jInterp);
AssertPtrNotNull(getDictionariesClass(jInterp));
AssertPtrEquals(getDictionariesClass(jInterp), coAlg);
AssertIntEquals(
  getDictionariesClass(jInterp)->super.objectSize,
  sizeof(DictObj)
```

```
)
```

3.8.3 Words

CCode : default

```
static void lookupAP(ContextObj* aCtx) {
  assert(aCtx);
  popCtxDataInto(aCtx, name2lookup);
  popCtxDataInto(aCtx, aDict);

  if (!isSymbol(name2lookup)) {
    raiseExceptionMsg(aCtx,
      "lookup requires a symbol as top");
    return;
  }

  if (!isDict(aDict)) {
    raiseExceptionMsg(aCtx,
      "lookup requires a dictionary as second");
    return;
  }

  DictNodeObj* entry =
    findSymbol((DictObj*)aDict, asSymbol(name2lookup));
  JObj* entryValue = NULL;
  if (entry) entryValue = entry->value;

  pushCtxData(aCtx, entryValue);
}
```

CHeader : private

```
extern Boolean registerDictionaryWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerDictionaryWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
```

```
  assert(jInterp);
  extendJoyLoLInRoot(jInterp, "lookup", "", lookupAP, "");
  return TRUE;
}
```

3.8.4 Lua functions

CCode : default

```
static int lua_dictionaries_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_dictionaries [] = {
  {"gitVersion", lua_dictionaries_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_dictionaries (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerDictionaries(jInterp);
  luaL_newlib(lstate, lua_dictionaries);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedDictionaries` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedDictionaries(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedDictionaries(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_dictionaries(lstate);
  lua_setfield(lstate, -2, "joylol.dictionaries");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.8.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_dictionaries(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/symbols.h>
#include <joylol/dictNodes.h>
#include <joylol/texts.h>
#include <joylol/cFunctions.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/dictionaries.h>
#include <joylol/dictionaries-private.h>
// dictionary
// printer
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
```

```
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictNodes");
requireStaticallyLinkedDictionaries(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.9 Code fragments

3.9.1 Goals

A code fragment contains a code fragment which can be used by the cross compiler.

3.9.2 Code

CHeader : public

```
typedef struct fragment_object_struct {
  JObj    super;
  Symbol *name;
  Symbol *body;
} FragmentObj;
```

3.9.2.1 Test Suite: newFragment

CHeader : public

```
typedef FragmentObj* (NewFragment)(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
);

#define newFragment(jInterp, aName, aBody)      \
  (                                             \
    assert(getFragmentsClass(jInterp)           \
      ->newFragmentFunc),                       \
    (getFragmentsClass(jInterp)                 \
      ->newFragmentFunc(jInterp, aName, aBody)) \
  )
//#define asFragment(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern FragmentObj* newFragmentImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
```

Code 3.9.2

```
);
```

CCode : default

```
FragmentObj* newFragmentImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  FragmentObj *result =
    (FragmentObj*)newObject(jInterp, FragmentsTag);
  assert(result);

//  result->super.type  = jInterp->coAlgs[FragmentsTag];
  result->name        = strdup(aName);
  result->body        = strdup(aBody);

  return result;
}
```

Test case

should create a new fragment

```
  AssertPtrNotNull(jInterp);

  FragmentObj* aNewFragment =
    newFragment(jInterp, "ansiC", "a fragment body");
  AssertPtrNotNull(aNewFragment);
  AssertPtrNotNull(asType(aNewFragment));
  AssertIntEquals(asTag(aNewFragment), FragmentsTag);
  AssertIntTrue(isAtom(aNewFragment));
  AssertIntTrue(isFragment(aNewFragment));
  AssertIntFalse(isPair(aNewFragment));
```

Test case

print Fragment

```
  AssertPtrNotNull(jInterp);
```

```
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

FragmentObj* aLoL =
  newFragment(jInterp, "ansiC", "a fragment body");
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, (JObj*)aLoL);
AssertStrEquals(getCString(aStrBuf), "fragment ");
strBufClose(aStrBuf);
```

3.9.2.2 Test Suite: isFragment

CHeader : public

```
#define isFragment(aLoL)              \
  (                                   \
    (                                 \
      (aLoL) &&                       \
      asType(aLoL) &&                 \
      (asTag(aLoL) == FragmentsTag)   \
    ) ?                               \
      TRUE :                          \
      FALSE                           \
  )
```

CHeader : private

```
extern Boolean equalityFragmentCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityFragmentCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "fragmentCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
```

```
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != FragmentsTag) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
}
```

3.9.2.3 Test Suite: printing fragments

CHeader : private

```
extern Boolean printFragmentCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printFragmentCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == FragmentsTag);

  strBufPrintf(aStrBuf, "fragment ");
  return TRUE;
}
```

Test case

should print fragments

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[FragmentsTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);
```

```
  FragmentObj* aNewFragment =
    newFragment(jInterp, "ansiC", "a fragment body");
  AssertPtrNotNull(aNewFragment);
  printLoL(aStrBuf, (JObj*)aNewFragment);
  AssertStrEquals(getCString(aStrBuf), "fragment ");
  strBufClose(aStrBuf);
```

3.9.2.4 Test Suite: registerFragments

CHeader : public

```
1  typedef struct fragments_class_struct {
2    JClass       super;
3    NewFragment      *newFragmentFunc;
4  } FragmentsClass;
```

CCode : default

```
1  static Boolean initializeFragments(
2    JoyLoLInterp *jInterp,
3    JClass   *aJClass
4  ) {
5    assert(jInterp);
6    assert(aJClass);
7    return TRUE;
8  }
```

CHeader : private

```
1  extern Boolean registerFragments(JoyLoLInterp *jInterp);
```

CCode : default

```
1  Boolean registerFragments(JoyLoLInterp *jInterp) {
2    assert(jInterp);
3    assert(jInterp->coAlgs);

4    FragmentsClass* theCoAlg
5      = joyLoLCalloc(1, FragmentsClass);
6    assert(theCoAlg);

7    theCoAlg->super.name           = FragmentsName;
8    theCoAlg->super.objectSize     = sizeof(FragmentObj);
9    theCoAlg->super.initializeFunc = initializeFragments;
```

```
  theCoAlg->super.registerFunc   = registerFragmentWords;
  theCoAlg->super.equalityFunc   = equalityFragmentCoAlg;
  theCoAlg->super.printFunc      = printFragmentCoAlg;
  theCoAlg->newFragmentFunc = newFragmentImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == FragmentsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Fragments coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerFragments
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getFragmentsClass(jInterp));
  FragmentsClass *coAlg = getFragmentsClass(jInterp);
  registerFragments(jInterp);
  AssertPtrNotNull(getFragmentsClass(jInterp));
  AssertPtrEquals(getFragmentsClass(jInterp), coAlg);
  AssertIntEquals(
    getFragmentsClass(jInterp)->super.objectSize,
    sizeof(FragmentObj)
  )
```

3.9.3 Words

CHeader : private

```
extern Boolean registerFragmentWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerFragmentWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  return TRUE;
}
```

3.9.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_fragments_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_fragments [] = {
  {"gitVersion", lua_fragments_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_fragments (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerFragments(jInterp);
```

```
20    luaL_newlib(lstate, lua_fragments);
21    return 1;
22  }
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedFragments` does just this.

CHeader : public

```
1  Boolean requireStaticallyLinkedFragments(
2    lua_State *lstate
3  );
```

CCode : default

```
1  Boolean requireStaticallyLinkedFragments(
2    lua_State *lstate
3  ) {
4    lua_getglobal(lstate, "package");
5    lua_getfield(lstate, -1, "loaded");
6    luaopen_joylol_fragments(lstate);
7    lua_setfield(lstate, -2, "joylol.fragments");
8    lua_setfield(lstate, -2, "loaded");
9    lua_pop(lstate, 1);
10   return TRUE;
11 }
```

3.9.5 Conclusions

CHeader : public

CHeader : private

```
1  extern size_t joylol_register_fragments(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/stringBuffers.h>
6  #include <joylol/dictNodes.h>
```

```
7  #include <joylol/texts.h>
8  #include <joylol/cFunctions.h>
9  #include <joylol/assertions.h>
10 #include <joylol/contexts.h>
11 #include <joylol/fragments.h>
12 #include <joylol/fragments-private.h>
13 // dictionary
14 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedFragments(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.10 JoyLoL implementations

3.10.1 Goals

A code implementation contains a collection of code fragments which can be used by the cross compiler.

3.10.2 Code

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```
typedef struct implementation_object_struct {
  JObj    super;
  Symbol *name;
  Symbol *body;
} ImplementationObj;
```

3.10.2.1 Test Suite: newImplementation

CHeader : public

```
typedef ImplementationObj* (NewImplementation)(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
);

#define newImplementation(jInterp, aName, aBody)      \
  (                                                   \
    assert(getImplementationsClass(jInterp)           \
      ->newImplementationFunc),                       \
```

Code 3.10.2

```
    (getImplementationsClass(jInterp)                  \
      ->newImplementationFunc(jInterp, aName, aBody)) \
  )
//#define asImplementation(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern ImplementationObj* newImplementationImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
);
```

CCode : default

```
ImplementationObj* newImplementationImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  ImplementationObj *result =
    (ImplementationObj*)newObject(jInterp, ImplementationsTag);
  assert(result);

//  result->super.type  = jInterp->coAlgs[ImplementationsTag];
  result->name        = strdup(aName);
  result->body        = strdup(aBody);

  return result;
}
```

Test case

should create a new Implementation

```
  AssertPtrNotNull(jInterp);

  ImplementationObj* aNewImplementation =
    newImplementation(jInterp, "ansiC", "a implementation body");
  AssertPtrNotNull(aNewImplementation);
  AssertPtrNotNull(asType(aNewImplementation));
```

```
AssertIntEquals(asTag(aNewImplementation), ImplementationsTag);
AssertIntTrue(isAtom(aNewImplementation));
AssertIntTrue(isImplementation(aNewImplementation));
AssertIntFalse(isPair(aNewImplementation));
```

Test case

print Implementation

```
AssertPtrNotNull(jInterp);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

ImplementationObj* aLoL =
  newImplementation(jInterp, "ansiC", "a implementation body");
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, (JObj*)aLoL);
AssertStrEquals(getCString(aStrBuf), "implementation ");
strBufClose(aStrBuf);
```

3.10.2.2 Test Suite: isImplementation

CHeader : public

```
#define isImplementation(aLoL)             \
  (                                        \
    (                                      \
      (aLoL) &&                            \
      asType(aLoL) &&                      \
      (asTag(aLoL) == ImplementationsTag)  \
    ) ?                                    \
      TRUE :                               \
      FALSE                                \
  )
```

CHeader : private

```
extern Boolean equalityImplementationCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
```

```
);
```

CCode : default

```
Boolean equalityImplementationCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "implementationCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != ImplementationsTag) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
}
```

3.10.2.3 Test Suite: printing implementations

CHeader : private

```
extern Boolean printImplementationCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printImplementationCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == ImplementationsTag);

  strBufPrintf(aStrBuf, "implementation ");
  return TRUE;
```

```
11 }
```

Test case

should print implementations

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs[ImplementationsTag]);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

ImplementationObj* aNewImplementation =
  newImplementation(jInterp, "ansiC", "a implementation body");
AssertPtrNotNull(aNewImplementation);
printLoL(aStrBuf, (JObj*)aNewImplementation);
AssertStrEquals(getCString(aStrBuf), "implementation ");
strBufClose(aStrBuf);
```

3.10.2.4 Test Suite: registerImplementations

CHeader : public

```
1 typedef struct implementations_class_struct {
2   JClass       super;
3   NewImplementation      *newImplementationFunc;
4 } ImplementationsClass;
```

CCode : default

```
1 static Boolean initializeImplementations(
2   JoyLoLInterp *jInterp,
3   JClass   *aJClass
4 ) {
5   assert(jInterp);
6   assert(aJClass);
7   return TRUE;
8 }
```

CHeader : private

```
1 extern Boolean registerImplementations(JoyLoLInterp *jInterp);
```

Code 3.10.3

CCode : default

```
Boolean registerImplementations(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  ImplementationsClass* theCoAlg
    = joyLoLCalloc(1, ImplementationsClass);
  assert(theCoAlg);

  theCoAlg->super.name           = ImplementationsName;
  theCoAlg->super.objectSize     = sizeof(ImplementationObj);
  theCoAlg->super.initializeFunc = initializeImplementations;
  theCoAlg->super.registerFunc   = registerImplementationWords;
  theCoAlg->super.equalityFunc   = equalityImplementationCoAlg;
  theCoAlg->super.printFunc      = printImplementationCoAlg;
  theCoAlg->newImplementationFunc = newImplementationImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == ImplementationsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Implementations coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerImplementations
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getImplementationsClass(jInterp));
  ImplementationsClass *coAlg = getImplementationsClass(jInterp);
  registerImplementations(jInterp);
  AssertPtrNotNull(getImplementationsClass(jInterp));
  AssertPtrEquals(getImplementationsClass(jInterp), coAlg);
  AssertIntEquals(
    getImplementationsClass(jInterp)->super.objectSize,
    sizeof(ImplementationObj)
  )
```

3.10.3 Words

CHeader : private

```
extern Boolean registerImplementationWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerImplementationWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  return TRUE;
}
```

3.10.4 Lua functions

CCode : default

```
static int lua_implementations_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_implementations [] = {
  {"gitVersion", lua_implementations_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_implementations (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerImplementations(jInterp);
```

```
  luaL_newlib(lstate, lua_implementations);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedImplementations` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedImplementations(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedImplementations(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_implementations(lstate);
  lua_setfield(lstate, -2, "joylol.implementations");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.10.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_implementations(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
```

```
7  #include <joylol/texts.h>
8  #include <joylol/cFunctions.h>
9  #include <joylol/assertions.h>
10 #include <joylol/contexts.h>
11 #include <joylol/implementations.h>
12 #include <joylol/implementations-private.h>
13 // dictionary
14 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedImplementations(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.11 JoyLoL interpreter

3.11.1 Overview

The whole of the JoyLoL language is oriented around the concept of lists. For *pure* JoyLoL, all CoAlgebras are implemented directly as lists of lists. *The only things there are, in* pure *JoyLoL, are lists.*

For the *impure* version of JoyLoL, provided in this release, we permit the existence of CoAlgebras which are implemented using computational structures related to the underlying computer architecture upon which the JoyLoL interpreter is running. Potentially, impure versions of JoyLoL might run faster. However the cost of greater performance is the greater possibility that the underlying implementation might not be as rigorously proven correct as a pure version of JoyLoL. Equally importantly, *impure* versions of JoyLoL, allow, 'a bear of little brain', like myself, to think in terms of higher level concepts while programming the fundamental algorithms in the theory associated with JoyLoL.

Since no general purpose computer processor is designed as a pure list processor, *any implementation* of JoyLoL (pure or impure) will of necessity contain non-list code which is used to *implement* lists. The purpose of this CoAlgebraic extension is to provide as simple and transparent an implementation of lists as possible.

In fact, *this* (core) CoAlgebraic extension does not actually implement lists. The implementation of lists is reserved for the `Pairs` CoAlgebraic extension developed in a separate document.

This `CoAlgs` CoAlgebraic extension provides the infrastructure with which to both organize the currently loaded collection of CoAlgebras, as well as provide `JObj`s which represent individual instances of any loaded CoAlgebra. Of particular importance is that these `JObj`s are automatically garbage collected.

The CoAlgebras 'extension' provides the basis upon which the whole of the JoyLoL implementation is built. It provides a first class interface.

We begin our implementation by pre-defining the three most important structures. We need to pre-define them here, since these three structures are mutually recursive.

- A `JoyLoLInterp` represents the JoyLoL interpreter. It includes the collection of all loaded CoAlgebraic extensions, as well as the object memory.

  CHeader : public

```
1  typedef struct joylol_interpreter_struct JoyLoLInterp;
```

- A `CoAlgebra` represents the standard (class) data and (class or instance) methods associated with the implementation of a CoAlgebraic extension.

  CHeader : public

```
typedef struct joylol_class_struct JClass;
```

- A `JObj` represents a relatively opaque object to one of many possible *instances* of a CoAlgebra.

  CHeader : public

```
typedef struct joylol_object_struct JObj;
```

The memory extension provides a first class JoyLoL implementation of 'memory'. A memory block has:

- a size
- an item size
- a next item
- a full indicator

Things you can do with a memory block:

- get a new item
- get a new block

The lists extension is the core JoyLoL extension. It provides the core lists of lists structures. It *only* provides the most basic JoyLoL expressions. It has no ability to preform any computation other than to build up List of Lists structures.

I suggest we use [generational garbage collection](http://wiki.c2.com/?GenerationalGarbageCollection) together with immutable objects so that objects in more recent "heaps" can only point to older "heaps" and not visa versa. This means that, if the individual "heaps" are first class Lua objects, then we can use the finalization of the heap object to ["stop and copy"](http://wiki.c2.com/?StopAndCopy) live JoyLoL objects into older heaps just before the newer heap is reclaimed by the Lua GC. Essentially we are letting the Lua GC drive the JoyLoL GC.

3.11.1.1 Required CoAlgebraic extensions

For any given (impure) implementation of a JoyLoL interpreter, there will be a collection of CoAlgebraic extensions *required* for the interpreter to function. By listing this collection of required extensions, we can give each one a unique and well known integer identifier, which can be used in the running code to quickly verify the class type of a given `JObj`.

Listed in the order in which they must be initialized, the required extensions for this implementation are:

CHeader : public

```
#define UnusedTag            0
#define JInterpsTag          1
#define DictionariesTag      2
#define AssertionsTag        3
#define BooleansTag          4
#define CFunctionsTag        5
#define CoAlgebrasTag        6
#define ContextsTag          7
#define CrossCompilersTag    8
#define DictNodesTag         9
#define FragmentsTag        10
#define ImplementationsTag  11
#define LoadersTag          12
#define LuaFunctionsTag     13
#define NaturalsTag         14
#define PairsTag            15
#define ParsersTag          16
#define RulesTag            17
#define SignalsTag          18
#define StringBuffersTag    19
#define SymbolsTag          20
#define TemplatesTag        21
#define TextsTag            22
#define NumRequiredCoAlgs   23
#define JInterpsName         "jInterps"         /*  1 */
#define DictionariesName     "dictionaries"     /*  2 */
#define AssertionsName       "assertions"       /*  3 */
#define BooleansName         "booleans"         /*  4 */
#define CFunctionsName       "cFunctions"       /*  5 */
#define CoAlgebrasName       "coAlgebras"       /*  6 */
#define ContextsName         "contexts"         /*  7 */
#define CrossCompilersName   "crossCompilers"   /*  8 */
#define DictNodesName        "dictNodes"        /*  9 */
#define FragmentsName        "fragments"        /* 10 */
#define ImplementationsName "implementations"  /* 11 */
#define LoadersName          "loaders"          /* 12 */
#define LuaFunctionsName     "luaFunctions"     /* 13 */
#define NaturalsName         "naturals"         /* 14 */
#define PairsName            "pairs"            /* 15 */
#define ParsersName          "parsers"          /* 16 */
```

```
41 #define RulesName            "rules"            /* 17 */
42 #define SignalsName          "signals"          /* 18 */
43 #define StringBuffersName    "stringBuffers"    /* 19 */
44 #define SymbolsName          "symbols"          /* 20 */
45 #define TemplatesName        "templates"        /* 21 */
46 #define TextsName            "texts"            /* 22 */
```

Since each CoAlgebraic extension can be developed separately, we need to ensure there is a strict *semantic* version control. We use the `CoAlgVersion` structure to contain the corresponding Major, Minor, and Patch version numbers for a given extension. Since our JoyLoL interpreter might depend upon specific aspects of extension's the 'Application Programming Interface' (API), our list of required extensions, below, also contains the minimum required compatible version of the extension.

CHeader : public

```
1 // see: http://semver.org/
2 //
3 typedef struct coalg_version_struct {
4   size_t major;
5   size_t minor;
6   size_t patch;
7 } CoAlgVersion;
```

When each CoAlgebraic extension is registered, its name is compared to the following list of required extensions. If a match is found then the corresponding (and well known) index is used to store a pointer to the `CoAlgebra` structure for the given extension. If a match is found the extension's current version is compared to the minimally required version. The registration of the extension is only accepted if these versions are compatible.

CHeader : private

```
1 typedef struct required_objects_struct {
2   const char*  name;
3   size_t       tag;
4   CoAlgVersion version;
5 } RequiredObjects;
6
7 extern RequiredObjects requiredCoAlgs[];
```

CCode : interpreter

```
1 RequiredObjects requiredCoAlgs[] = {
2   { JInterpsName,        JInterpsTag,        {0, 1, 0 }}, //  1
3   { DictionariesName,    DictionariesTag,    {0, 1, 0 }}, //  2
```

```
  { AssertionsName,       AssertionsTag,       {0, 1, 0 }}, //  3
  { BooleansName,         BooleansTag,         {0, 1, 0 }}, //  4
  { CFunctionsName,       CFunctionsTag,       {0, 1, 0 }}, //  5
  { CoAlgebrasName,       CoAlgebrasTag,       {0, 1, 0 }}, //  6
  { ContextsName,         ContextsTag,         {0, 1, 0 }}, //  7
  { CrossCompilersName,   CrossCompilersTag,   {0, 1, 0 }}, //  8
  { DictNodesName,        DictNodesTag,        {0, 1, 0 }}, //  9
  { FragmentsName,        FragmentsTag,        {0, 1, 0 }}, // 10
  { ImplementationsName, ImplementationsTag,   {0, 1, 0 }}, // 11
  { LoadersName,          LoadersTag,          {0, 1, 0 }}, // 12
  { LuaFunctionsName,     LuaFunctionsTag,     {0, 1, 0 }}, // 13
  { NaturalsName,         NaturalsTag,         {0, 1, 0 }}, // 14
  { PairsName,            PairsTag,            {0, 1, 0 }}, // 15
  { ParsersName,          ParsersTag,          {0, 1, 0 }}, // 16
  { RulesName,            RulesTag,            {0, 1, 0 }}, // 17
  { SignalsName,          SignalsTag,          {0, 1, 0 }}, // 18
  { StringBuffersName,    StringBuffersTag,    {0, 1, 0 }}, // 19
  { SymbolsName,          SymbolsTag,          {0, 1, 0 }}, // 20
  { TemplatesName,        TemplatesTag,        {0, 1, 0 }}, // 21
  { TextsName,            TextsTag,            {0, 1, 0 }}, // 22
  { NULL,                 0,                   {0, 0, 0 }}
};
```

LuaCode : default

```
local joylol = { }

-- load all required CoAlgebraic extensions -- creation phase

joylol.jInterps        = require 'joylol.jInterps'        --  1
joylol.dicionaries     = require 'joylol.dictionaries'    --  2
joylol.assertions      = require 'joylol.assertions'      --  3
joylol.booleans        = require 'joylol.booleans'        --  4
joylol.cFunctions      = require 'joylol.cFunctions'      --  5
joylol.coAlgebras      = require 'joylol.coAlgebras'      --  6
joylol.contexts        = require 'joylol.contexts'        --  7
joylol.crossCompilers  = require 'joylol.crossCompilers'  --  8
joylol.dictNodes       = require 'joylol.dictNodes'       --  9
joylol.fragments       = require 'joylol.fragments'       -- 10
joylol.implementations = require 'joylol.implementations' -- 11
joylol.loaders         = require 'joylol.loaders'         -- 12
joylol.luaFunctions    = require 'joylol.luaFunctions'    -- 13
joylol.naturals        = require 'joylol.naturals'        -- 14
joylol.pairs           = require 'joylol.pairs'           -- 15
```

```
joylol.parsers         = require 'joylol.parsers'         -- 16
joylol.rules           = require 'joylol.rules'           -- 17
joylol.signals         = require 'joylol.signals'         -- 18
joylol.stringBuffers   = require 'joylol.stringBuffers'   -- 19
joylol.symbols         = require 'joylol.symbols'         -- 20
joylol.templates       = require 'joylol.templates'       -- 21
joylol.texts           = require 'joylol.texts'           -- 22

-- load all required CoAlgebraic extensions -- initialization phase

joylol.jInterps.initializeAllRequired()
```

3.11.2 Required cross compilers

One of the key objectives of the JoyLoL interpreter is to ensure as much as possible of its own code and the code of any of its extensions is proven correct. While it is theoretically impossible to ever prove any implementation of reality to be 100%correct, we can ensure that those code fragements which can only be tests as opposed to proven correct, is as small as possible.

It is the `CrossCompiler` CoAlgebra which manages the process of proving *implemented* JoyLoL words are proven correct. The `CrossCompiler` coalgebras do this by first verifying and then assembling JoyLoL words from adequately tests code fragments in a particular programming language. There is one `CrossCompiler` for each programming language as embedded in a computer system.

For the current JoyLoL implementation, the required `CrossCompilers` are:

CHeader : public

```
#define UnusedCC                0
#define AnsicCC                 1
#define AnsicLuaCC              2
#define PureLuaCC               3
#define NumRequiredCrossCompilers 4
#define AnsicName     "ansic"     /*  1 */
#define AnsicLuaName  "ansicLua"  /*  2 */
#define PureLuaName   "pureLua"   /*  3 */
```

CHeader : private

```
extern RequiredObjects requiredCompilers[];
```

CCode : interpreter

```
RequiredObjects requiredCompilers[] = {
  { AnsicName,       AnsicCC,     {0, 1, 0 }}, //  1
```

```
  { AnsicLuaName,    AnsicLuaCC,  {0, 1, 0 }}, //  2
  { PureLuaName,     PureLuaCC,   {0, 1, 0 }}, //  3
  { NULL,             0,          {0, 0, 0 }}
};
```

3.11.3 Semi-standard typedefs

Finally we provide a number of semi-standard typdefs to provide a semantic meaning to various typical variable uses.

CHeader : public

```
typedef size_t     Boolean;
typedef const char Symbol;
```

3.11.4 JObjs

Since the implementation of any particular CoAlgebraic extension will of necessity make use of instance specific data stored in a `JObject`, we *begin* by providing the implementation of `JObj`. Together with any extension specific part, all of our `JObj`s have the same three part base structure.

3.11.4.1 Type part

The first (type) part is a `CoAlgebra*` pointer to the data structure which represents the CoAlgebra for which the `JObj` is an instance. This CoAlgebra pointer ensures that the implementation code knows what the given instance, is an instance *for*, as well as what it can *do*.

3.11.4.2 Tag part

The second (tag) part is an unsigned integer index into the `JoyLoLInterp`'s vector of registered `CoAlgebra`s. While the tags of all the required `CoAlgebraic` extensions have been defined above and are, hence, 'well known'. Tags for all non-required extensions are assigned as the extension is registered with the interpreter.

CHeader : public

```
#ifdef __LP64__
typedef uint32_t TagType;
#else
typedef uint16_t TagType;
```

```
#endif
```

3.11.4.3 Flag part

The third (flag) part is a collection of bits to provide useful meta-flags associated with a particula instance. At least one of these meta-flags are reserved by the JoyLoL interpreter to signal ongoing garbage collection. Since any object is potentially part of a cyclic structure, these meta-flags ensure garbage collection does not fall into infinite cycles.

All non-reserved meta-flags may be used by the implementation of a CoAlgebra extension for its own internal purposes. Typically meta-flags might be used to signal how to interpret the data stored in the third (data) part of the object. For the `Naturals` CoAlgebraic extension, a meta-flag will be used to signal that the object's data part is a pointer to a Gnu Multi-precision integer, rather than to a double word integer. This allows significant speed optimizations in the typical cases, but allows for full data representations in rare but important cases.

All reserved meta-flags will be located in low order bits of the flag data word. This ensures that any CoAlgebraic extension which makes use of meta-flags can simply rotate the reserved flags off the end of the word before making use of the non-reserved flags. In particular a CoAlgebraic extension *could* interpret its flags as an integer or pointer. Such interpretations are private to each extension, and should *not* be relied upon by code which is not part of the code's own extension.

As 'global' meta-flags we reserve the following three *low-order* bits together with a Mask of all three bits and the number of reserved bits to shift (left).

CHeader : public

```
#define MARK_SWEEP_FLAG     0x1L
#define RESERVED_FLAG_MASK  0x1L
#define RESERVED_FLAG_SHIFT 1
#ifdef __LP64__
typedef uint32_t FlagsType;
#else
typedef uint16_t FlagsType;
#endif
```

We can now define the *base* `JObj` as:

CHeader : public

```
typedef struct joylol_object_struct {
  JClass    *type;
  TagType    tag;
  FlagsType  flags; // an arbitrary collection of bits
} JObj;
```

```
6
7   #define asType(aLoL)  (((JObj*)(aLoL))->type)
8   #define asTag(aLoL)   (((JObj*)(aLoL))->tag)
9   #define asFlags(aLoL) (((JObj*)(aLoL))->flags)
10  #define isFlagSet(aLoL, bitMask)        \
11    (((JObj*)(aLoL))->flags & (bitMask))
12  #define isFlagClear(aLoL, bitMask)          \
13    ((~(((JObj*)(aLoL))->flags)) & (bitMask))
14  #define checkObj(jInterp, theObj, theTag) \
15    (                                       \
16      assert(jInterp),                      \
17      assert(theObj),                       \
18      assert(theTag),                       \
19      assert((((JObj*)(theObj))->type) ==   \
20        ((jInterp)->coAlgs[theTag]))        \
21    )
```

3.11.5 isAtom isPair

A common requirement is to determine whether or not a given `JObj` is an 'atom' or a 'pair'. Quite simply we define anything that is not an instance of `Pairs` an 'atom'.

CHeader : public

```
1   #define isAtom(anObject) \
2     ((anObject) && (asTag(anObject) != PairsTag))

3   #define isPair(anObject) \
4     ((anObject) && (asTag(anObject) == PairsTag))
```

3.11.6 Object Memory

Since a JoyLoL interpreter is essentially a list processor, any JoyLoL program will create (and orphan) a very large number of list `Pairs` over the course of a computation. This means that we need to make the allocation and eventual garbage collection of orphaned `JObjs` as efficient as possible.

We do this by defining a `ObjectMemory` as a pair of pointers. The first pointer points to a linked list of free objects. The second pointer points to a linked list of `ObjectBlocks`.

CHeader : public

```
1   typedef struct object_block_struct ObjectBlock;
```

```
2
3   //typedef struct object_memory_struct {
4   //  JObj*    freeObjects;
5   //  ObjectBlock* rootObjectBlock;
6   //} ObjectMemory;
7
8   typedef struct object_block_struct {
9     size_t       objectSize;
10    void*        block;
11    ObjectBlock* nextBlock;
12  } ObjectBlock;
```

3.11.6.1 Test Suite: addObjectBlock

CHeader : private

```
1   #define OBJECT_BLOCK_SIZE 1024
2   extern void addObjectBlock(JClass *theClass);
```

CCode : objects

```
1   void addObjectBlock(JClass* theClass) {
2     assert(theClass);
3     assert(theClass->jInterp);

4     size_t objectSize = theClass->objectSize;
5
6     DEBUG(theClass->jInterp, "addObjectBlock > %p [%s] %zu %zu\n",
7       theClass, theClass->name, objectSize, (size_t)OBJECT_BLOCK_SIZE);

8     // obtain a new object block
9     ObjectBlock* aNewObjectBlock =
10      (ObjectBlock*)calloc(1, sizeof(ObjectBlock));
11    assert( IS_MEM_ALIGNED(aNewObjectBlock) );
12
13    aNewObjectBlock->objectSize = objectSize;
14    aNewObjectBlock->nextBlock  = NULL;

15    // integrate this new object block into the linked list of
16    // object blocks
17    if ( theClass->rootObjectBlock ) {
18      ObjectBlock *lastObjectBlock = theClass->rootObjectBlock;
19      while ( lastObjectBlock->nextBlock ) {
```

```
20        lastObjectBlock = lastObjectBlock->nextBlock;
21      }
22      assert(lastObjectBlock->nextBlock == NULL);
23      lastObjectBlock->nextBlock = aNewObjectBlock;
24    } else {
25      theClass->rootObjectBlock = aNewObjectBlock;
26    }
27    DEBUG(theClass->jInterp, "addObjectBlock = %p %p\n",
28      theClass, aNewObjectBlock);
29
30    // make sure this object block has some JObjs
31    aNewObjectBlock->block =
32      calloc(OBJECT_BLOCK_SIZE, objectSize);
33    assert( aNewObjectBlock->block );
34    assert( IS_MEM_ALIGNED(aNewObjectBlock->block) );

35    // add these new JObjs to the free list
36    void* nextObject = aNewObjectBlock->block;
37    for (size_t i = 1 ; i < OBJECT_BLOCK_SIZE ; i++) {
38      assert( IS_MEM_ALIGNED(nextObject) );
39      *(void**)nextObject = nextObject + objectSize;
40      nextObject += objectSize;
41    }
42    *(void**)nextObject =
43      (void*)theClass->freeObjects;
44    theClass->freeObjects = aNewObjectBlock->block;
45    DEBUG(theClass->jInterp, "addObjectBlock < %p %p %p\n",
46      theClass, aNewObjectBlock, aNewObjectBlock->block);
47  }
```

Test case

must add new object block

```
  // create the first list block and make sure it is
  // properly integrated into linked list of list blocks

  AssertPtrNotNull(jInterp);
  JClass *theClass = jInterp->coAlgs[JInterpsTag];
  AssertPtrNotNull(theClass);

  AssertPtrNull(theClass->rootObjectBlock);
  AssertPtrNull(theClass->freeObjects);
```

```
  addObjectBlock(theClass);
  AssertPtrNotNull(theClass->rootObjectBlock);
  AssertPtrNotNull(theClass->rootObjectBlock->block);
  AssertPtrNull(theClass->rootObjectBlock->nextBlock);
  AssertPtrNotNull(theClass->freeObjects);

  // check to make sure freeObjects list is correctly linked
  void* nextObject  = theClass->rootObjectBlock->block;
  size_t objectSize = theClass->rootObjectBlock->objectSize;
  AssertIntEquals(objectSize, theClass->objectSize);

  for ( size_t i = 1 ; i < OBJECT_BLOCK_SIZE ; i++, nextObject += objectSize
) {
    AssertIntTrue(IS_MEM_ALIGNED(nextObject));
    AssertIntZero(asTag(nextObject));
    AssertIntZero(asFlags(nextObject));
    AssertPtrEquals( asType(nextObject), nextObject + objectSize);
  }
  AssertPtrNull(*((void**)nextObject));
  AssertPtrEquals(theClass->freeObjects,
    theClass->rootObjectBlock->block);

  // add another object block
  void* oldFreeObjects = theClass->freeObjects;
  addObjectBlock(theClass);
  AssertPtrNotNull(theClass->rootObjectBlock->nextBlock);
  AssertPtrNotNull(theClass->rootObjectBlock->nextBlock->block);
  AssertPtrNull(theClass->rootObjectBlock->nextBlock->nextBlock);
  AssertPtrNotNull(theClass->freeObjects);

  // check to make sure freeObjects list is correctly linked
  nextObject = theClass->rootObjectBlock->nextBlock->block;

  for (size_t i = 1 ; i < OBJECT_BLOCK_SIZE ; i++, nextObject += objectSize
) {
    AssertIntTrue(IS_MEM_ALIGNED(nextObject));
    AssertIntZero(asTag(nextObject));
    AssertIntZero(asFlags(nextObject));
    AssertPtrEquals(asType(nextObject), nextObject + objectSize);
  }
  AssertPtrEquals(asType(nextObject), oldFreeObjects);
  AssertPtrEquals(theClass->freeObjects,
    theClass->rootObjectBlock->nextBlock->block);
```

3.11.6.2 Garbage collection

Given that we explicitly implement memory pools for all fixed sized structures, the highest churn will be in the creation and release of small strings through interactions directly with the user and with the template engine. Since all strings used by JoyLoL will be part of the `Symbols` CoAlgebraic extension, we have complete control over the use of pointers to strings. This suggests that we allocate all 'small strings' using our own implementation of a collection of string pools. This allows us to periodically compact under-utilized pools which are highly fragmented by copying strings to either completely new string pools or inside existing string pools. By having a series of string pools we have the ability to provide the pools with a generational structure so that 'older' pools would tend to fragment less.

However before we implement this strategy we should begin by instrumenting the use of small strings so we can gain an understanding of the spectrum of string sizes.

'Large strings' will tend to be produced as the output of either the user interaction or the template engine. We can reduce churn in these large strings by ensuring the user interaction interface and template engine either uses a list of strings (which the interface/engine implicitly concatenates), or uses an explicit string buffer. Initially any such string buffer can be based upon GNU Lib C's open_memstream interface, though we could directly implement our own string buffers should that be needed.

CHeader : private

```
1  extern void collectGarbage(JClass *theClass);
```

CCode : objects

```
1  void collectGarbage(JClass *theClass) {
2    assert(theClass);

3    DEBUG(theClass->jInterp, "collectGarbage %p\n", theClass);
4    //
5    // add a garbage collection mark-sweep here
6    //
7    // we will use a tri/quad colour mark-sweep algorithm
8    // similar to that used by LuaJIT v3.0
9    // see: http://wiki.luajit.org/New-Garbage-Collector
10   // see: https://en.wikipedia.org/wiki/Tracing_garbage_collection
11   // our "grey" list will be simply scanning a given
12   // object block for currently grey markings... and
13   // we keep a pointer to the current object block and
```

```
  // where in the block we last checked for grey markings.
  // this means we can have an incremental mark/trace cycle
  // which can be run in small increments in each call to eval.
  //
}
```

3.11.6.3 Test Suite: newObject

To allocate a new `JObj` the first free object is taken from the linked list of free objects. If there are no remaining free objects, then we first attempt to collect any garbage and then if this fails to get any new free objects, a new `ObjectBlock` is allocated together with its collection of free objects.

CHeader : public

```
#define newObject(jInterp, aTag)                                    \
  (                                                                 \
    assert(getJInterpClass(jInterp)->newObjectFunc),                \
    (getJInterpClass(jInterp)->newObjectFunc)(jInterp, aTag)        \
  )
```

CHeader : private

```
extern JObj* newObjectImpl(
  JoyLoLInterp* jInterp,
  TagType aTag
);
```

CCode : objects

```
JObj* newObjectImpl(
  JoyLoLInterp* jInterp,
  TagType aTag
) {
  assert(jInterp);
  DEBUG(jInterp, "newObjectImpl(start) %p %zu\n", jInterp, (size_t)aTag);

  assert(jInterp->coAlgs);
  assert(aTag < jInterp->numCoAlgs);
  JClass *theClass = jInterp->coAlgs[aTag];
  assert(theClass);
  assert(theClass->tag == aTag);

  DEBUG(jInterp, "newObjectImpl freeObjects %p objectBlock %p\n",
    theClass->freeObjects,
```

```
    theClass->rootObjectBlock
  );

  // ensure there are some free objects
  if ( ! theClass->freeObjects )
    collectGarbage(theClass);
  if ( ! theClass->freeObjects )
    addObjectBlock(theClass);

  assert(theClass->freeObjects);

  JObj* aNewObject   = theClass->freeObjects;
  theClass->freeObjects = (JObj*)(aNewObject->type);

  asType(aNewObject)  = jInterp->coAlgs[aTag];
  asTag(aNewObject)   = aTag;
  asFlags(aNewObject) = 0;

  DEBUG(jInterp, "newObjectImpl(done) %p %zu\n", aNewObject, (size_t)aTag);
  return aNewObject;
}
```

Test case

Allocate one new JObj

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
TagType aTag = JInterpsTag;
JClass *theClass = jInterp->coAlgs[aTag];
AssertPtrNotNull(theClass);

// a block has already been assigned from the last test
// so we simply throw it away...
theClass->freeObjects = NULL;
theClass->rootObjectBlock = NULL;

// get one new object to ensure our lazily
// initialized object memory structures
// are initialized...
JObj* aNewObject = newObject(jInterp, aTag);

AssertPtrNotNull(theClass->freeObjects);
```

```
AssertPtrNotNull(theClass->rootObjectBlock);
AssertPtrNotNull(aNewObject);
AssertPtrNotNull(asType(aNewObject));
AssertPtrEquals(asType(aNewObject), theClass);
AssertIntEquals(asTag(aNewObject), aTag);
AssertIntZero(asFlags(aNewObject));
AssertPtrEquals(aNewObject,
  theClass->rootObjectBlock->block);

// now get one more object to ensure
// we can properly deal with object blocks/memory

JObj* oldFreeObjects =
  theClass->freeObjects;
JObj* newfreeObjects =
  (JObj*)theClass->freeObjects->type;

aNewObject = newObject(jInterp, JInterpsTag);

AssertPtrNotNull(aNewObject);
AssertPtrNotNull(asType(aNewObject));
AssertPtrEquals(asType(aNewObject), jInterp->coAlgs[JInterpsTag]);
AssertIntEquals(asTag(aNewObject), JInterpsTag);
AssertIntZero(asFlags(aNewObject));
AssertPtrEquals(oldFreeObjects, aNewObject);
AssertPtrEquals(newfreeObjects, theClass->freeObjects);
```

Test case

Allocate lots of new JObjs

```
AssertPtrNotNull(jInterp);

JObj* aNewObject = NULL;
for ( size_t i = 0; i < 3*OBJECT_BLOCK_SIZE; i++ ) {
  aNewObject = newObject(jInterp, JInterpsTag);
}
AssertPtrNotNull(aNewObject);
AssertPtrNotNull(asType(aNewObject));
AssertPtrEquals(asType(aNewObject), jInterp->coAlgs[JInterpsTag]);
AssertIntEquals(asTag(aNewObject), JInterpsTag);
AssertIntZero(asFlags(aNewObject));
```

3.11.7 JoyLoL interpreter

In this section we concentrate on the C code required to implement the kernel JoyLoL CoAlgebra. The CoAlgebra, `CoAlg`, is the base of the JoyLoL interpreter. This CoAlgebra has two parts, a 'class' part, and an 'instance' part. This class part manages the loading, registration and listing of new 'external' CoAlgebras. The instance part, discussed in the next section, manages the creation and removal of CoAlgebra 'values' (more commonly know as 'object' instances).

Any given implementation of the overall JoyLoL *system* will consist of a number of CoAlgebraic extensions which are loaded into a JoyLoL 'core'. Any given core will provide some resources for the extensions to use. Before we can document the generic structure of an extension, we need to provide a method to allow the JoyLoL interpreter extension to obtain these resources from the core.

3.11.7.1 Test Suite: get Lua-state global JoyLoL-Callback LightUserData

Each JoyLoL core implementation will have a number of specific resources such as C-functions which it needs to register with the JoyLoL interpreter. In particular the core specific Input/Output will need to be available to the JoyLoL interpreter soon after it loads in order to allow it to correctly communicate with the core and hence the user. The `getJoyLoLCallbackInto` and `setJoyLoLCallbackFrom` macros provide this capability by registering a well known 'callback' C-function *before* the JoyLoL interpreter is loaded.

The callback C-function takes an unsigned integer as its sole argument and returns a `void*` to the returned resource.

CHeader : public

```
typedef void *(JoyLoLCallback)(
  lua_State*,
  size_t
);
```

The currently well known resources are:

CHeader : public

```
#define JoyLoLCallback_StdOutMethod   1
#define JoyLoLCallback_StdErrMethod   2
#define JoyLoLCallback_Verbose        3
#define JoyLoLCallback_Debug          4
#define JoyLoLCallback_Trace          5
#define JoyLoLCallback_Quiet          6
#define JoyLoLCallback_ConfigFile     7
#define JoyLoLCallback_UserPath       8
```

```
#define JoyLoLCallback_LocalPath      9
#define JoyLoLCallback_SystemPath     10
typedef void (StdOutputMethod)(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
);
```

The core implementation can then use the `setJoyLoLCallbackFrom` macro to store the address of its specific callback C-function as a LightUserData in the Lua Registry at the `JoyLoLCallbackKey`. The JoyLoL interpreter, once it is loaded, can get the address of the callback C-function using the `getJoyLoLCallbackInto` macro.

This means that `joyLoLCallbackKey` must be a unique value, but has no meaningful value *other than* its uniqueness. To do this we transliterate the first 4 characters of the string 'JyLCallbackKey' into hexadecimal using the standard `ASCII` character codes.

CHeader : public

```
//                          J y L C a l l b a c k K e y
#define joyLoLCallbackKey 0x4A794C43L
#define getJoyLoLCallbackInto(lstate, aCallback)  \
  lua_rawgetp(                                    \
    lstate,                                       \
    LUA_REGISTRYINDEX,                            \
    (void *)joyLoLCallbackKey                     \
  );                                              \
  JoyLoLCallback* aCallback =                     \
    (JoyLoLCallback*)lua_touserdata (lstate, -1); \
  lua_pop(lstate, 1);                             \
  if (!aCallback) {                               \
    /*return*/ luaL_error(lstate, "%s%s%s%s",     \
      "\nERROR:\n",                               \
      "  Could not get the Lua registered\n",     \
      "  JoyLoLCallback method!\n",               \
      "  Have you required joylol.core.xxx?\n");  \
  }
```

CHeader : public

```
#define setJoyLoLCallbackFrom(lstate, aCallback)    \
  lua_pushlightuserdata(lstate, (void *)aCallback); \
  lua_rawsetp(                                      \
    lstate,                                         \
    LUA_REGISTRYINDEX,                              \
```

```
    (void *)joyLoLCallbackKey                    \
)
```

Test case

should get a Lua-State global JoyLoL-Callback LightUserData

```
// NOTE we MUST test using ctestsCallback
// OR we MUST reset the call back to ctestsCallback
//
JoyLoLCallback *origCallback = ctestsCallback;
setJoyLoLCallbackFrom(lstate, origCallback);
getJoyLoLCallbackInto(lstate, aCallback);
AssertPtrNotNull(aCallback);
AssertPtrEquals(origCallback, aCallback);
getJoyLoLCallbackInto(lstate, newCallback);
AssertPtrNotNull(newCallback);
AssertPtrEquals(origCallback, newCallback);
```

3.11.8 CoAlgebra extensions

The ANSI-C implementation of a CoAlgebra is as a simple `struct` which contains the following items:

- a `size_t` value which is unique for each registered CoAlgebra and hence acting as a test for identity,
- a `Symbol` value which provides a human readable *name* for the CoAlgebra,
- a `C-function` which (recursively) tests for equality of two CoAlgebra instances of given type of CoAlgebra.
- a `C-function` with (re)registers a given CoAlgebra with the JoyLoL interpreter. This registration function's primary purpose is two fold. It registers the CoAlgebra with the central collection of known CoAlgebras (to be discussed in the next subsection, below), as well as registers the individual JoyLoL functions provided by the CoAlgebra with the interpreter's dictionary of words.

CHeader : public

```
typedef Boolean (JClassInitialize)(
  JoyLoLInterp*,
  JClass*
);
```

```
5
6   typedef Boolean (JObjEquality)(
7     JoyLoLInterp *jInterp,
8     JObj         *lolA,
9     JObj         *lolB,
10    size_t        timeToLive
11  );
12
13  typedef struct string_buffer_object_struct StringBufferObj;
14  typedef Boolean (JObjPrint)(
15    StringBufferObj *aStBuf,
16    JObj            *aLoL,
17    size_t           timeToLive
18  );
19
20  typedef JObj* (JObjCopy)(
21    JoyLoLInterp*,
22    JObj*
23  );
24
25  typedef Boolean (JObjRelease)(
26    JoyLoLInterp*,
27    JObj*
28  );
29
30  typedef void*   CoAlgData;
31
32  typedef struct joylol_class_struct {
33    Symbol           *name;
34    JoyLoLInterp     *jInterp;
35    size_t            tag;
36    size_t            objectSize;
37    size_t            numObjectsPerBlock;
38    JObj             *freeObjects;
39    ObjectBlock      *rootObjectBlock;
40    JClassInitialize *initializeFunc;
41    JClassInitialize *registerFunc;
42    JObjEquality     *equalityFunc;
43    JObjPrint        *printFunc;
44    JObjCopy         *copyFunc;
45    JObjRelease      *releaseFunc;
46  } JClass;
47
```

```
48  typedef JObj* (JInterpNewObject)(
49    JoyLoLInterp *jInterp,
50    TagType       aTag
51  );
52
53  typedef size_t (JInterpRegisterJClass)(
54    JoyLoLInterp *jInterp,
55    JClass   *theJClass
56  );
57
58  typedef struct crossCompiler_object_struct CrossCompilerObj;
59  typedef size_t (JInterpRegisterCrossCompiler)(
60    JoyLoLInterp     *jInterp,
61    CrossCompilerObj *aCompiler
62  );
63
64  typedef void (JInterpInitializeAllLoaded)(
65    lua_State    *lstate,
66    JoyLoLInterp *jInterp
67  );
68
69  typedef struct joylolinterp_class_struct {
70    JClass                         super;
71    JInterpNewObject              *newObjectFunc;
72    JInterpRegisterJClass         *registerJClassFunc;
73    JInterpRegisterCrossCompiler *registerCrossCompilerFunc;
74    JInterpInitializeAllLoaded    *initializeAllLoadedFunc;
75    JInterpInitializeAllLoaded    *registerAllLoadedFunc;
76  } JoyLoLInterpClass;
```

3.11.8.1 JoyLoLInterp structures

The `JoyLoLInterp` structure provides access to all of the CoAlgebras known to a given JoyLoL interpreter. As such it will be accessed repeatedly, so should probably be accessible with as few memory accesses as possible.

CHeader : public

```
1  typedef struct dictNode_object_struct DictNodeObj;
2
3  typedef struct loader_object_struct LoaderObj;
4  typedef struct context_object_struct ContextObj;
5  typedef struct dictionary_object_struct DictObj;
```

```
6  typedef struct crossCompiler_object_struct CrossCompilerObj;
7
8  typedef struct joylol_interpreter_struct {
9    lua_State        *lstate;
10   size_t            numCoAlgs;
11   size_t            maxNumCoAlgs;
12   JClass          **coAlgs;
13   LoaderObj        *loader;
14   ContextObj       *rootCtx;
15   size_t            numCompilers;
16   size_t            maxNumCompilers;
17   CrossCompilerObj **compilers;
18   Boolean           verbose;
19   Boolean           debug;
20   Boolean           tracing;
21   Boolean           quiet;
22   StdOutputMethod  *writeStdOut;
23   StdOutputMethod  *writeStdErr;
24 } JoyLoLInterp;
25
26 #define getJClass(jInterp, aTag)     \
27   (                                  \
28     assert(jInterp),                 \
29     assert(aTag),                    \
30     assert((jInterp)->coAlgs),       \
31     assert((jInterp)->coAlgs[aTag]), \
32     ((jInterp)->coAlgs[aTag])        \
33   )
34 #define getJInterpClass(jInterp)                              \
35   ((JoyLoLInterpClass*)getJClass(jInterp, JInterpsTag))
36 typedef struct assertions_class_struct AssertionsClass;
37 #define getAssertionsClass(jInterp)                           \
38   ((AssertionsClass*)getJClass(jInterp, AssertionsTag))
39 typedef struct booleans_class_struct BooleansClass;
40 #define getBooleansClass(jInterp)                             \
41   ((BooleansClass*)getJClass(jInterp, BooleansTag))
42 typedef struct cFunctions_class_struct CFunctionsClass;
43 #define getCFunctionsClass(jInterp)                           \
44   ((CFunctionsClass*)getJClass(jInterp, CFunctionsTag))
45 typedef struct coAlgebras_class_struct CoAlgebrasClass;
46 #define getCoAlgebrasClass(jInterp)                           \
47   ((CoAlgebrasClass*)getJClass(jInterp, CoAlgebrasTag))
48 typedef struct contexts_class_struct ContextsClass;
```

```
49  #define getContextsClass(jInterp)                                   \
50    ((ContextsClass*)getJClass(jInterp, ContextsTag))
51  typedef struct crossCompilers_class_struct CrossCompilersClass;
52  #define getCrossCompilersClass(jInterp)                             \
53    ((CrossCompilersClass*)getJClass(jInterp, CrossCompilersTag))
54  typedef struct dictNodes_class_struct DictNodesClass;
55  #define getDictNodesClass(jInterp)                                  \
56    ((DictNodesClass*)getJClass(jInterp, DictNodesTag))
57  typedef struct dictionaries_class_struct DictionariesClass;
58  #define getDictionariesClass(jInterp)                               \
59    ((DictionariesClass*)getJClass(jInterp, DictionariesTag))
60  typedef struct fragments_class_struct FragmentsClass;
61  #define getFragmentsClass(jInterp)                                  \
62    ((FragmentsClass*)getJClass(jInterp, FragmentsTag))
63  typedef struct implementations_class_struct ImplementationsClass;
64  #define getImplementationsClass(jInterp)                                \
65    ((ImplementationsClass*)getJClass(jInterp, ImplementationsTag))
66  typedef struct loaders_class_struct LoadersClass;
67  #define getLoadersClass(jInterp)                                    \
68    ((LoadersClass*)getJClass(jInterp, LoadersTag))
69  typedef struct luaFunctions_class_struct LuaFunctionsClass;
70  #define getLuaFunctionsClass(jInterp)                               \
71    ((LuaFunctionsClass*)getJClass(jInterp, LuaFunctionsTag))
72  typedef struct naturals_class_struct NaturalsClass;
73  #define getNaturalsClass(jInterp)                                   \
74    ((NaturalsClass*)getJClass(jInterp, NaturalsTag))
75  typedef struct pairs_class_struct PairsClass;
76  #define getPairsClass(jInterp)                                      \
77    ((PairsClass*)getJClass(jInterp, PairsTag))
78  typedef struct parsers_class_struct ParsersClass;
79  #define getParsersClass(jInterp)                                    \
80    ((ParsersClass*)getJClass(jInterp, ParsersTag))
81  typedef struct rules_class_struct RulesClass;
82  #define getRulesClass(jInterp)                                  \
83    ((RulesClass*)getJClass(jInterp, RulesTag))
84  typedef struct signals_class_struct SignalsClass;
85  #define getSignalsClass(jInterp)                                \
86    ((SignalsClass*)getJClass(jInterp, SignalsTag))
87  typedef struct stringBuffers_class_struct StringBuffersClass;
88  #define getStringBuffersClass(jInterp)                              \
89    ((StringBuffersClass*)getJClass(jInterp, StringBuffersTag))
90  typedef struct symbols_class_struct SymbolsClass;
91  #define getSymbolsClass(jInterp)                                    \
```

```
92    ((SymbolsClass*)getJClass(jInterp, SymbolsTag))

93  typedef struct templates_class_struct TemplatesClass;
94  #define getTemplatesClass(jInterp)                          \
95    ((TemplatesClass*)getJClass(jInterp, TemplatesTag))
96  typedef struct texts_class_struct TextsClass;
97  #define getTextsClass(jInterp)                              \
98    ((TextsClass*)getJClass(jInterp, TextsTag))
```

3.11.8.2 Test Suite: newJoyLoLInterp

To be able to either test or use a JoyLoL interpreter, we must first be able to obtain a new one. The (privately defined) `newJoyLoLInterp` creates a new `JoyLoLInterp` structure and ensures it is properly initialized. Note that we *could* have multiple instances of `JoyLoLInterp` in a running program.

CHeader : private

```
1   extern JoyLoLInterp* newJoyLoLInterp(lua_State *lstate);
```

CCode : interpreter

```
1   JoyLoLInterp* newJoyLoLInterp(lua_State *lstate) {
2     // we need to create a new JoyLoLInterp structure...
3     //
4     JoyLoLInterp* jInterp = joyLoLCalloc(1, JoyLoLInterp);
5     assert(jInterp);
6
7     jInterp->lstate = lstate;

8     // before we do anything else we need to install the
9     // core's Output methods
10    getJoyLoLCallbackInto(lstate, getCoreResources);
11    assert(getCoreResources);
12    jInterp->writeStdOut =
13      getCoreResources(lstate, JoyLoLCallback_StdOutMethod);
14    jInterp->writeStdErr =
15      getCoreResources(lstate, JoyLoLCallback_StdErrMethod);
16    if (!(jInterp->writeStdOut) || !(jInterp->writeStdErr)) {
17      /*return*/ luaL_error(lstate, "%s%s",
18        "\nERROR:\n",
19        "  Could not get the core output methods\n");
20    }
21    jInterp->verbose =
```

```
22    (Boolean)getCoreResources(lstate, JoyLoLCallback_Verbose);

23  jInterp->debug =
24    (Boolean)getCoreResources(lstate, JoyLoLCallback_Debug);

25  jInterp->tracing =
26    (Boolean)getCoreResources(lstate, JoyLoLCallback_Trace);
27
28  jInterp->quiet =
29    (Boolean)getCoreResources(lstate, JoyLoLCallback_Quiet);
30
31  DEBUG(jInterp, "Creating jInterp %p\n", jInterp);

32  size_t maxNum =
33    NumRequiredCoAlgs + JOYLOL_COALGS_INCREMENT;
34  jInterp->numCoAlgs    = NumRequiredCoAlgs;
35  jInterp->maxNumCoAlgs = maxNum;
36  jInterp->coAlgs       = joyLoLCalloc(maxNum, JClass*);

37  DEBUG(jInterp, "Created coAlgs vector %p %zu %zu\n",
38    jInterp->coAlgs, jInterp->numCoAlgs, jInterp->maxNumCoAlgs);

39  for (size_t i = 0; i < jInterp->maxNumCoAlgs; i++) {
40    jInterp->coAlgs[i] = NULL;
41  }
42
43  DEBUG(jInterp, "Initialized coAlgs %p\n", jInterp);

44  maxNum =
45    NumRequiredCrossCompilers + JOYLOL_COMPILERS_INCREMENT;
46  jInterp->numCompilers    = NumRequiredCrossCompilers;
47  jInterp->maxNumCompilers = maxNum;
48  jInterp->compilers       = joyLoLCalloc(maxNum, CrossCompilerObj*);
49
50  DEBUG(jInterp, "Created crossCompilers vector %p %zu %zu\n",
51    jInterp->compilers,
52    jInterp->numCompilers,
53    jInterp->maxNumCompilers);
54
55  for (size_t i = 0; i < jInterp->maxNumCompilers; i++) {
56    jInterp->compilers[i] = NULL;
57  }
58
```

```
59    DEBUG(jInterp, "Initialized compilers %p\n", jInterp);
60
61    //jInterp->dict    = NULL;
62    jInterp->loader  = NULL;
63    jInterp->rootCtx = NULL;
64    DEBUG(jInterp, "Initialized jInterp %p\n", jInterp);
65
66    return jInterp;
67  }
```

Test case

should create a valid JoyLoLInterp instance

```
JoyLoLInterp *jInterp = newJoyLoLInterp(lstate);
AssertPtrNotNull(jInterp);

AssertIntEquals(jInterp->numCoAlgs,
  NumRequiredCoAlgs);
AssertIntEquals(jInterp->maxNumCoAlgs,
  NumRequiredCoAlgs + JOYLOL_COALGS_INCREMENT);
AssertPtrNotNull(jInterp->coAlgs);

for (size_t i = 0; i < jInterp->maxNumCoAlgs; i++) {
  AssertPtrNull(jInterp->coAlgs[i]);
}

AssertPtrNull(jInterp->rootCtx);
AssertPtrNull(jInterp->loader);
```

3.11.8.3 Test Suite: get Lua-state global JoyLoLInterp LightUserData

We want the ability to have different JoyLoL interpreters for each Lua state. However, we also need fast access to the C-implementations of the loaded CoAlgebras registered with a given JoyLoL interpreter. To achieve this we use the Lua Registry with the 'well-known' key `joyLoLInterpKey`. We store a pointer to the JoyLoL interpreter `JoyLoLInterp`, as a LightUserData, in the Lua registry under the key `joyLoLInterpKey`, so that it is only accessible by the C implementation of any CoAlgebra. This ensures that the pointer to `JoyLoLInterp` is unique for any particular Lua state, but can be different for each distinct Lua state.

This means that `joyLoLInterpKey` must be a unique value, but has no meaningful value *other than* its uniqueness. To do this we transliterate the first 4 characters of the string 'JoyLoLInterpKey' into hexadecimal using the standard `ASCII` character codes.

CHeader : public

```
1  //                        J y L I n t e r p K e y
2  #define joyLoLInterpKey 0x4A794C49L
3  #define getJoyLoLInterpInto(lstate, jInterp)      \
4    lua_rawgetp(                                    \
5      lstate,                                       \
6      LUA_REGISTRYINDEX,                            \
7      (void *)joyLoLInterpKey                       \
8    );                                              \
9    JoyLoLInterp* jInterp =                         \
10     (JoyLoLInterp*)lua_touserdata (lstate, -1);   \
11   lua_pop(lstate, 1);                             \
12   if (!jInterp) {                                 \
13     /*return*/ luaL_error(lstate, "%s%s%s%s",     \
14       "\nERROR:\n",                                \
15       "  Could not get the Lua registered\n",      \
16       "  JoyLoLInterp instance!\n",                \
17       "  Have you required joylol.jInterps?\n");   \
18   }
```

CHeader : private

```
1  #define setJoyLoLInterpInto(lstate, jInterp)      \
2    JoyLoLInterp *jInterp =                         \
3      newJoyLoLInterp(lstate);                      \
4    lua_pushlightuserdata(lstate, (void *)jInterp); \
5    lua_rawsetp(                                    \
6      lstate,                                       \
7      LUA_REGISTRYINDEX,                            \
8      (void *)joyLoLInterpKey                       \
9    )
```

Test case

should get a Lua-State global JoyLoLInterp LightUserData

```
getJoyLoLInterpInto(lstate, jInterp);
AssertPtrNotNull(jInterp);
AssertPtrEquals(jInterp->lstate, lstate);
AssertPtrNotNull(jInterp->coAlgs);
```

```
  AssertIntEquals(jInterp->numCoAlgs, NumRequiredCoAlgs);
  AssertIntEquals(jInterp->maxNumCoAlgs,
    NumRequiredCoAlgs + JOYLOL_COALGS_INCREMENT);
  getJoyLoLInterpInto(lstate, newJInterp);
  AssertPtrNotNull(newJInterp);
  AssertPtrEquals(jInterp, newJInterp);
```

3.11.8.4 Test Suite: registerCoAlgebra

CHeader : public

```
#define registerJClass(jInterp, aTag)                                  \
  (                                                                    \
    assert(getJInterpClass(jInterp)->registerJClassFunc),              \
    (getJInterpClass(jInterp)->registerJClassFunc)(jInterp, aTag)      \
  )
```

CHeader : private

```
size_t registerJClassImpl(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : interpreter

```
size_t registerJClassImpl(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  assert(jInterp->coAlgs);
  assert(theCoAlg);

  theCoAlg->jInterp = jInterp;

  for(size_t i = 0; i < jInterp->numCoAlgs; i++ ) {
    if (jInterp->coAlgs[i]) {
      if (strcmp(theCoAlg->name,
        jInterp->coAlgs[i]->name) == 0) {

        // a coAlgebra with this name has already been registered.
        // so return its index...
        return i;
```

```
    }
  }
}

// now check to see if this is a required coAlg
for (size_t i = 0; requiredCoAlgs[i].name; i++) {
  if (strcmp(theCoAlg->name, requiredCoAlgs[i].name) == 0) {
    size_t tag = requiredCoAlgs[i].tag;
    if (!(jInterp->coAlgs[tag])) {
      jInterp->coAlgs[tag] = theCoAlg;
    }
    theCoAlg->tag = tag;
    return tag;
  }
}

// we follow a policy of lazy management of the jInterp
// if jInterp->coAlgs is too small we expand it
if (jInterp->maxNumCoAlgs <= jInterp->numCoAlgs) {
  // we need to expand the existing coAlgs structure...
  //
  size_t oldnumCoAlgs     = jInterp->numCoAlgs;
  size_t oldmaxNumCoAlgs  = jInterp->maxNumCoAlgs;
  JClass **oldcoAlgs = jInterp->coAlgs;

  size_t newMaxNumCoAlgs =
    oldmaxNumCoAlgs + JOYLOL_COALGS_INCREMENT;

  jInterp->coAlgs =
    joyLoLCalloc(newMaxNumCoAlgs, JClass*);
  assert(jInterp->coAlgs);

  if (oldcoAlgs) {
    memcpy(jInterp->coAlgs,
      oldcoAlgs,
      sizeof(JClass*)*oldnumCoAlgs);
    free(oldcoAlgs);
  }

  jInterp->numCoAlgs    = oldnumCoAlgs;
  jInterp->maxNumCoAlgs = newMaxNumCoAlgs;
}
```

```
55    size_t newCoAlg = jInterp->numCoAlgs;

56    jInterp->coAlgs[newCoAlg] = theCoAlg;
57    theCoAlg->tag             = newCoAlg;

58    jInterp->numCoAlgs++;

59    return newCoAlg;
60  }
```

Test case

should add lots of CoAlgs to a JoyLoLInterp

We start by adding one single `CoAlgebra`.

```
JoyLoLInterp *jInterp = newJoyLoLInterp(lstate);
char*             coAlgName          = strdup("newCoAlgA");
char*             coAlgNameEnd       = coAlgName + strlen("newCoAlg");
JClassInitialize *fakeInitializeFunc = (JClassInitialize*) 0x100;
JClassInitialize *fakeRegisterFunc   = (JClassInitialize*) 0x200;
JObjEquality     *fakeEqualityFunc   = (JObjEquality*)     0x300;
JObjPrint        *fakePrintFunc      = (JObjPrint*)        0x400;

JClass* aCoAlg   = joyLoLCalloc(1, JClass);
AssertPtrNotNull(aCoAlg);
aCoAlg->name           = strdup("newCoAlgA");
aCoAlg->initializeFunc = fakeInitializeFunc;
aCoAlg->registerFunc   = fakeRegisterFunc;
aCoAlg->equalityFunc   = fakeEqualityFunc;
aCoAlg->printFunc      = fakePrintFunc;

size_t coAlgIdx = registerJClassImpl(jInterp, aCoAlg);
AssertIntEquals(coAlgIdx, NumRequiredCoAlgs);

JClass** coAlgsVec = jInterp->coAlgs;
AssertPtrNotNull(coAlgsVec);
AssertStrEquals(coAlgsVec[coAlgIdx]->name, "newCoAlgA");
AssertPtrEquals(coAlgsVec[coAlgIdx]->initializeFunc, fakeInitializeFunc);
AssertPtrEquals(coAlgsVec[coAlgIdx]->registerFunc,   fakeRegisterFunc);
AssertPtrEquals(coAlgsVec[coAlgIdx]->equalityFunc,   fakeEqualityFunc);
AssertPtrEquals(coAlgsVec[coAlgIdx]->printFunc,      fakePrintFunc);
```

Now we want to test the expansion of an existing CoAlgebras structure when the existing one runs out of 'space'.

```
AssertIntTrue(JOYLOL_COALGS_INCREMENT < 26);
for(size_t i = 1; i < 26; i++) {
  *coAlgNameEnd          = 'A' + i;
  fakeInitializeFunc    += 1;
  fakeRegisterFunc      += 1;
  fakeEqualityFunc      += 1;
  fakePrintFunc         += 1;
  aCoAlg                 = joyLoLCalloc(1, JClass);
  aCoAlg->name           = strdup(coAlgName);
  aCoAlg->initializeFunc = fakeInitializeFunc;
  aCoAlg->registerFunc   = fakeRegisterFunc;
  aCoAlg->equalityFunc   = fakeEqualityFunc;
  aCoAlg->printFunc      = fakePrintFunc;

  registerJClassImpl(jInterp, aCoAlg);

  JClass** coAlgsVec = jInterp->coAlgs;
  AssertPtrNotNull(coAlgsVec);
  size_t idx = NumRequiredCoAlgs + i;
  AssertStrEquals(coAlgsVec[idx]->name, coAlgName);
  AssertPtrEquals(coAlgsVec[idx]->initializeFunc, fakeInitializeFunc);
  AssertPtrEquals(coAlgsVec[idx]->registerFunc,   fakeRegisterFunc);
  AssertPtrEquals(coAlgsVec[idx]->equalityFunc,   fakeEqualityFunc);
  AssertPtrEquals(coAlgsVec[idx]->printFunc,      fakePrintFunc);
}
```

Now we want to test the addition of CoAlgebra with an existing name does not change the number of registered CoAlgebras but simple returns the existing CoAlgebra index.

```
size_t oldNumCoAlgs  = jInterp->numCoAlgs;

*coAlgNameEnd          = 'A' + 5;
aCoAlg                 = joyLoLCalloc(1, JClass);
aCoAlg->name           = strdup(coAlgName);
aCoAlg->initializeFunc = (JClassInitialize*) 0x100 + 5;
aCoAlg->registerFunc   = (JClassInitialize*) 0x200 + 5;
aCoAlg->equalityFunc   = (JObjEquality*)     0x300 + 5;
aCoAlg->printFunc      = (JObjPrint*)        0x400 + 5;

size_t anIndex = registerJClassImpl(jInterp, aCoAlg);
AssertIntEquals(anIndex, 5 + NumRequiredCoAlgs);
AssertIntEquals(oldNumCoAlgs, jInterp->numCoAlgs);
```

3.11.8.5 Test Suite: registerCrossCompiler

CHeader : public

```
#define registerCrossCompiler(jInterp, aTag)       \
  (                                                \
    assert(getJInterpClass(jInterp)                \
      ->registerCrossCompilerFunc),                \
    (getJInterpClass(jInterp)                      \
      ->registerCrossCompilerFunc)(jInterp, aTag) \
  )
```

CHeader : private

```
size_t registerCrossCompilerImpl(
  JoyLoLInterp     *jInterp,
  CrossCompilerObj *aCompiler
);
```

CCode : interpreter

```
size_t registerCrossCompilerImpl(
  JoyLoLInterp     *jInterp,
  CrossCompilerObj *aCompiler
) {
  assert(jInterp);
  assert(jInterp->compilers);
  assert(aCompiler);

  for(size_t i = 0; i < jInterp->numCompilers; i++ ) {
    if (jInterp->compilers[i]) {
      if (strcmp(aCompiler->type,
        jInterp->compilers[i]->type) == 0) {

        // a compiler with this name has already been registered.
        // so return its index...
        return i;
      }
    }
  }

  // now check to see if this is a required coAlg
  for (size_t i = 0; requiredCompilers[i].name; i++) {
```

```
21      if (strcmp(aCompiler->type, requiredCompilers[i].name) == 0) {
22        size_t tag = requiredCompilers[i].tag;
23        if (!(jInterp->compilers[tag])) {
24          jInterp->compilers[tag] = aCompiler;
25        }
26        return tag;
27      }
28    }
29
30    // we follow a policy of lazy management of the jInterp
31    // if jInterp->compilers is too small we expand it
32    if (jInterp->maxNumCompilers <= jInterp->numCompilers) {
33      // we need to expand the existing compilers structure...
34      //
35      size_t oldNumCompilers          = jInterp->numCompilers;
36      size_t oldMaxNumCompilers       = jInterp->maxNumCompilers;
37      CrossCompilerObj** oldCompilers = jInterp->compilers;
38      size_t newMaxNumCompilers =
39        oldMaxNumCompilers + JOYLOL_COMPILERS_INCREMENT;
40
41      jInterp->compilers =
42        joyLoLCalloc(newMaxNumCompilers, CrossCompilerObj*);
43      assert(jInterp->compilers);
44      if (oldCompilers) {
45        memcpy(jInterp->compilers,
46          oldCompilers,
47          sizeof(CrossCompilerObj*)*oldNumCompilers);
48        free(oldCompilers);
49      }
50      jInterp->numCompilers    = oldNumCompilers;
51      jInterp->maxNumCompilers = newMaxNumCompilers;
52    }
53    size_t newCompiler = jInterp->numCompilers;
54    jInterp->compilers[newCompiler] = aCompiler;
55    jInterp->numCompilers++;
56    return newCompiler;
```

```
57 }
```

Test case

should add lots of Compilers to a JoyLoLInterp

We start by adding one single `Compiler`.

```
JoyLoLInterp *jInterp = newJoyLoLInterp(lstate);
char*             compilerType       = strdup("newCompilerA");
char*             compilerTypeEnd    = compilerType + strlen("newCompiler");

CrossCompilerObj* aCompiler = joyLoLCalloc(1, CrossCompilerObj);
AssertPtrNotNull(aCompiler);
aCompiler->type               = strdup("newCompilerA");

size_t compilerIdx = registerCrossCompilerImpl(jInterp, aCompiler);
AssertIntEquals(compilerIdx, NumRequiredCrossCompilers);

CrossCompilerObj** compilers = jInterp->compilers;
AssertPtrNotNull(compilers);
AssertStrEquals(compilers[compilerIdx]->type, "newCompilerA");
```

Now we want to test the expansion of an existing CoAlgebras structure when the existing one runs out of 'space'.

```
AssertIntTrue(JOYLOL_COALGS_INCREMENT < 26);
for(size_t i = 1; i < 26; i++) {
  *compilerTypeEnd = 'A' + i;
  aCompiler        = joyLoLCalloc(1, CrossCompilerObj);
  aCompiler->type  = strdup(compilerType);

  registerCrossCompilerImpl(jInterp, aCompiler);

  CrossCompilerObj** compilers = jInterp->compilers;
  AssertPtrNotNull(compilers);
  size_t idx = NumRequiredCrossCompilers + i;
  AssertStrEquals(compilers[idx]->type, compilerType);
}
```

Now we want to test the addition of CoAlgebra with an existing name does not change the number of registered CoAlgebras but simple returns the existing CoAlgebra index.

```
size_t oldNumCompilers = jInterp->numCompilers;
```

```
  *compilerTypeEnd       = 'A' + 5;
  aCompiler              = joyLoLCalloc(1, CrossCompilerObj);
  aCompiler->type        = strdup(compilerType);

  size_t anIndex = registerCrossCompilerImpl(jInterp, aCompiler);
  AssertIntEquals(anIndex, 5 + NumRequiredCrossCompilers);
  AssertIntEquals(oldNumCompilers, jInterp->numCompilers);
```

3.11.8.6 Test Suite: initializeAllLoaded

CHeader : public

```
#define initializeAllLoaded(lstate, jInterp)      \
  (                                               \
    assert(getJInterpClass(jInterp)               \
      ->initializeAllLoadedFunc),                 \
    (getJInterpClass(jInterp)                     \
      ->initializeAllLoadedFunc(lstate, jInterp)) \
  )
```

CHeader : private

```
extern void initializeAllLoadedImpl(
  lua_State    *lstate,
  JoyLoLInterp *jInterp
);
```

CCode : interpreter

```
void initializeAllLoadedImpl(
  lua_State    *lstate,
  JoyLoLInterp *jInterp
) {
  assert(lstate);
  assert(jInterp);
  assert(jInterp->coAlgs);

  // now call the initialization function for each
  // loaded CoAlgebraic extension
  for( size_t i = 1; i < jInterp->numCoAlgs; i++) {
    if (jInterp->coAlgs[i]) {
      JClass *aCoAlg = getJClass(jInterp, i);
      if (jInterp->verbose) {
        char output[1000];
```

```
      memset(output, 0, 1000);
      snprintf(output, 999,
        "    initializing [joylol.%s]\n",
        aCoAlg->name);
      jInterp->writeStdOut(jInterp, output);
    }
    if (aCoAlg->initializeFunc)
      aCoAlg->initializeFunc(jInterp, aCoAlg);

    if (jInterp->verbose) {
      char output[1000];
      memset(output, 0, 1000);
      snprintf(output, 999,
        "    initialized [joylol.%s]\n",
        aCoAlg->name);
      jInterp->writeStdOut(jInterp, output);
    }
  }
}
```

3.11.8.7 Test Suite: registerAllLoaded

CHeader : public

```
#define registerAllLoaded(lstate, jInterp)      \
  (                                               \
    assert(getJInterpClass(jInterp)               \
      ->registerAllLoadedFunc),                   \
    (getJInterpClass(jInterp)                     \
      ->registerAllLoadedFunc(lstate, jInterp)) \
  )
```

CHeader : private

```
extern void registerAllLoadedImpl(
  lua_State    *lstate,
  JoyLoLInterp *jInterp
);
```

CCode : interpreter

```
void registerAllLoadedImpl(
  lua_State    *lstate,
```

```
3     JoyLoLInterp *jInterp
4   ) {
5     assert(lstate);
6     assert(jInterp);
7     assert(jInterp->coAlgs);

8     // now call the register function for each
9     // loaded CoAlgebraic extension
10    for( size_t i = 1; i < jInterp->numCoAlgs; i++) {
11      if (jInterp->coAlgs[i]) {
12        JClass *aCoAlg = getJClass(jInterp, i);
13        if (jInterp->verbose) {
14          char output[1000];
15          memset(output, 0, 1000);
16          snprintf(output, 999,
17            "    registering words for [joylol.%s]\n",
18            aCoAlg->name);
19          jInterp->writeStdOut(jInterp, output);
20        }
21        if (aCoAlg->registerFunc)
22          aCoAlg->registerFunc(jInterp, aCoAlg);

23        if (jInterp->verbose) {
24          char output[1000];
25          memset(output, 0, 1000);
26          snprintf(output, 999,
27            "    registered words for [joylol.%s]\n",
28            aCoAlg->name);
29          jInterp->writeStdOut(jInterp, output);
30        }
31      }
32    }
33  }
```

3.11.8.8 Test Suite: getGitVersion

We need a generic way of obtaining the git version of a given CoAlgebra. We do this in two parts. In the `bin` directory of any (collection of) CoAlgebra(s) there should be a pair of shell scripts. The `gitHook.sh` is a `bash` shell script which is run at various points in the git event cycle to capture the git version information into either a C-header file or a Lua file. These files can then be loaded as needed into either a C-implementation or required by Lua code, to provide information

about the version of a given code artefact. The `bin` directory should also contain a `setupGitHooks bash` script which copies the `gitHook.sh` script into the correct places in the `.git` directory.

For CoAlgebras implemented in ANSI-C, the following function will take the git version information in the `gitVersion.h` file which is statically linked in a given CoAlgabra's shared library and return the value associated with any valid key requested.

CHeader : public

```
1  typedef struct keyValueStruct {
2    const char *key;
3    const char *value;
4  } KeyValues;
5
6  #define getGitVersionInto(gitVersion, gitVersionKey, theValue)  \
7    Symbol* theValue = "key not found";                           \
8    for(size_t i = 0 ; gitVersion[i].key ; i++) {                 \
9      if (strcmp(gitVersionKey, gitVersion[i].key) == 0) {        \
10       theValue = gitVersion[i].value;                           \
11       break;                                                    \
12     }                                                           \
13   }
```

At the moment we can only (easily) test the `getGitVersion` function.

Test case

should get value associated with keys

```
static const KeyValues testKVs[] = {
  { "key1", "value1" },
  { "key2", "value2" },
  { NULL,   NULL }
};
getGitVersionInto(testKVs, "key1", aValue0);
AssertPtrNotNull(aValue0);
AssertStrEquals(aValue0, "value1");
getGitVersionInto(testKVs, "key2", aValue1);
AssertPtrNotNull(aValue1);
AssertStrEquals(aValue1, "value2");
getGitVersionInto(testKVs, "noKey", aValue2);
AssertPtrNotNull(aValue2);
AssertStrEquals(aValue2, "key not found");
```

CCode : interpreter

```
static Boolean initializeJInterps(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

3.11.8.9 Test Suite: regiserJInterps

CHeader : private

```
extern Boolean registerJInterps(JoyLoLInterp *jInterp);
```

CCode : interpreter

```
Boolean registerJInterps(JoyLoLInterp *jInterp) {
  assert(jInterp);

  JoyLoLInterpClass* theCoAlg =
    joyLoLCalloc(1, JoyLoLInterpClass);
  assert(theCoAlg);

  theCoAlg->super.name              = JInterpsName;
  theCoAlg->super.objectSize        = sizeof(JoyLoLInterp);
  theCoAlg->super.initializeFunc    = initializeJInterps;
  theCoAlg->super.registerFunc      = registerJInterpWords;
  theCoAlg->super.equalityFunc      = NULL;
  theCoAlg->super.printFunc         = NULL;
  theCoAlg->newObjectFunc           = newObjectImpl;
  theCoAlg->registerJClassFunc      = registerJClassImpl;
  theCoAlg->registerCrossCompilerFunc =
    registerCrossCompilerImpl;
  theCoAlg->initializeAllLoadedFunc =
    initializeAllLoadedImpl;
  theCoAlg->registerAllLoadedFunc =
    registerAllLoadedImpl;

  size_t tag =
    registerJClassImpl(jInterp, (JClass*)theCoAlg);
```

```
  // do a sanity check...
  assert(tag == JInterpsTag);
  assert(tag < jInterp->numCoAlgs);
  assert(jInterp->coAlgs);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the JInterps coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerJInterps
  //
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertIntTrue(JInterpsTag < jInterp->numCoAlgs);
  AssertPtrNotNull(getJInterpClass(jInterp));

  JoyLoLInterpClass *jInterpClass = getJInterpClass(jInterp);
  size_t result = registerJInterps(jInterp);
  AssertIntTrue(result);
  AssertPtrNotNull(getJInterpClass(jInterp));
  AssertPtrEquals(getJInterpClass(jInterp), jInterpClass);
  AssertIntEquals(
    getJInterpClass(jInterp)->super.objectSize,
    sizeof(JoyLoLInterp)
  )
```

3.11.9 JoyLoL words

CCode : interpreter

```
static void doWrite(
  ContextObj      *aCtx,
  StdOutputMethod *writeMethod,
  Boolean          newLine
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
```

```
8     assert(jInterp);
9     assert(writeMethod);

10    popCtxDataInto(aCtx, aString);
11    if (!isSymbol(aString)) {
12      raiseExceptionMsg(aCtx,
13        "write expected a symbol as aString");
14      return;
15    }
16
17    (writeMethod)(jInterp, asSymbol(aString));
18    if (newLine) (writeMethod)(jInterp, "\n");
19  }
20
21  static void writeOutAP(ContextObj *aCtx) {
22    assert(aCtx);
23    JoyLoLInterp *jInterp = aCtx->jInterp;
24    assert(jInterp);

25    doWrite(aCtx, jInterp->writeStdOut, FALSE);
26  }
27
28  static void writeOutNLAP(ContextObj *aCtx) {
29    assert(aCtx);
30    JoyLoLInterp *jInterp = aCtx->jInterp;
31    assert(jInterp);

32    doWrite(aCtx, jInterp->writeStdOut, TRUE);
33  }
34
35  static void writeErrAP(ContextObj *aCtx) {
36    assert(aCtx);
37    JoyLoLInterp *jInterp = aCtx->jInterp;
38    assert(jInterp);

39    doWrite(aCtx, jInterp->writeStdErr, FALSE);
40  }
41
42  static void writeErrNLAP(ContextObj *aCtx) {
43    assert(aCtx);
44    JoyLoLInterp *jInterp = aCtx->jInterp;
45    assert(jInterp);
```

```
  doWrite(aCtx, jInterp->writeStdErr, TRUE);
}
```

CHeader : private

```
extern Boolean registerJInterpWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : interpreter

```
Boolean registerJInterpWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  extendJoyLoLInRoot(jInterp, "writeOut",   "", writeOutAP,   "");
  extendJoyLoLInRoot(jInterp, "writeOutNL", "", writeOutNLAP, "");
  extendJoyLoLInRoot(jInterp, "writeErr",   "", writeErrAP,   "");
  extendJoyLoLInRoot(jInterp, "writeErrNL", "", writeErrNLAP, "");
  return TRUE;
}
```

3.11.10 Lua interface functions

CCode : lua

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : lua

```
static int lua_jInterps_getGitVersion (lua_State *lstate) {
  Symbol* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
```

```
6     } else {
7       lua_pushstring(lstate, "no valid key provided");
8     }
9     return 1;
10  }
```

CCode : lua

```
1   static int lua_joylol_setVerbose(lua_State *lstate) {
2     Boolean verbose =
3       (lua_toboolean(lstate, 1) ? TRUE : FALSE);
4     getJoyLoLInterpInto(lstate, jInterp);
5     assert(jInterp);
6     assert(jInterp->rootCtx);
7     jInterp->verbose = verbose;
8     jInterp->rootCtx->verbose = verbose;
9     lua_pop(lstate, 1);
10    return 0;
11  }
12
13  static int lua_joylol_setDebugging(lua_State *lstate) {
14    Boolean debugFlag =
15      (lua_toboolean(lstate, 1) ? TRUE : FALSE);
16    getJoyLoLInterpInto(lstate, jInterp);
17    assert(jInterp);
18    jInterp->debug = debugFlag;
19    lua_pop(lstate, 1);
20    return 0;
21  }
22
23  static int lua_joylol_setTracing(lua_State *lstate) {
24    Boolean tracingFlag =
25      (lua_toboolean(lstate, 1) ? TRUE : FALSE);
26    getJoyLoLInterpInto(lstate, jInterp);
27    assert(jInterp);
28    assert(jInterp->rootCtx);
29    jInterp->tracing = tracingFlag;
30    jInterp->rootCtx->tracingOn = tracingFlag;
31    lua_pop(lstate, 1);
32    return 0;
33  }
34
35  static int lua_joylol_setShowStack(lua_State *lstate) {
36    Boolean showStackFlag =
```

```
37      (lua_toboolean(lstate, 1) ? TRUE : FALSE);
38    getJoyLoLInterpInto(lstate, jInterp);
39    assert(jInterp);
40    assert(jInterp->rootCtx);
41    jInterp->rootCtx->showStack = showStackFlag;
42    lua_pop(lstate, 1);
43    return 0;
44  }
45
46  static int lua_joylol_setShowSpecifications(lua_State *lstate) {
47    Boolean showSpecsFlag =
48      (lua_toboolean(lstate, 1) ? TRUE : FALSE);
49    getJoyLoLInterpInto(lstate, jInterp);
50    assert(jInterp);
51    assert(jInterp->rootCtx);
52    jInterp->rootCtx->showSpecifications = showSpecsFlag;
53    lua_pop(lstate, 1);
54    return 0;
55  }
56
57  static int lua_joylol_setChecking(lua_State *lstate) {
58    Boolean checkingFlag =
59      (lua_toboolean(lstate, 1) ? TRUE : FALSE);
60    getJoyLoLInterpInto(lstate, jInterp);
61    assert(jInterp);
62    assert(jInterp->rootCtx);
63    jInterp->rootCtx->checkingOn = checkingFlag;
64    lua_pop(lstate, 1);
65    return 0;
66  }
67
68  static int lua_joylol_pushLoadPath(lua_State *lstate) {
69    Symbol *aPath = lua_tostring(lstate, 1);
70    getJoyLoLInterpInto(lstate, jInterp);
71    pushLoadPath(jInterp->loader, aPath);
72    lua_pop(lstate, 1);
73    return 0;
74  }
75
76  static int lua_joylol_loadFile(lua_State *lstate) {
77    Symbol *aFile = lua_tostring(lstate, 1);
78    getJoyLoLInterpInto(lstate, jInterp);
79    loadAFile(jInterp->rootCtx, aFile);
```

```
  lua_pop(lstate, 1);
  return 0;
}
```

CCode : lua

```
static void pushLuaToJoylol(
  lua_State  *lstate,
  ContextObj *rootCtx,
  Boolean     isData
) {
  int aType = lua_type(lstate, -1);
  switch(aType) {
  case LUA_TNIL : {
    if (isData) {
      pushNullCtxData(rootCtx);
    } else {
      pushNullCtxProcess(rootCtx);
    }
  } break;
  case LUA_TNUMBER : {
    lua_Number aDouble = lua_tonumber(lstate, 1);
    if (isData) {
      pushNaturalCtxData(rootCtx, (size_t)aDouble);
    } else {
      pushNaturalCtxProcess(rootCtx, (size_t)aDouble);
    }
  } break;
  case LUA_TBOOLEAN : {
    int aBool = lua_toboolean(lstate, -1);
    if (isData) {
      pushBooleanCtxData(rootCtx, aBool);
    } else {
      pushBooleanCtxProcess(rootCtx, aBool);
    }
  } break;
  case LUA_TSTRING : {
    Symbol *aString = lua_tostring(lstate, -1);
    if (isData) {
      pushSymbolCtxData(rootCtx, aString);
    } else {
      pushSymbolCtxProcess(rootCtx, aString);
    }
  } break;
```

```
  case LUA_TTABLE : {
    // we ONLY take numerically keyed items from tables
    // that is we treat all tables as ARRAYS not HashTables
    size_t tableLen = lua_rawlen(lstate, -1);
    for (size_t i = 1; i <= tableLen; i++) {
      lua_rawgeti(lstate, -1, i);
      pushLuaToJoylol(lstate, rootCtx, isData);
      lua_pop(lstate, 1);
    }
  } break;
  case LUA_TFUNCTION :
  case LUA_TUSERDATA :
  case LUA_TTHREAD :
  case LUA_TLIGHTUSERDATA :
  default:
    // ignore ....
    break;
  }
}
```

CCode : lua

```
static void peekJoylolToLua(
  lua_State    *lstate,
  JoyLoLInterp *jInterp,
  JObj         *aLoL
) {
  if (!aLoL) {
    lua_pushnil(lstate);
  } else if (isAtom(aLoL)) {
    if (isSymbol(aLoL)) {
      lua_pushstring(lstate, asSymbol(aLoL));
    } else if (isNatural(aLoL)) {
      lua_pushnumber(lstate, asNaturalDbl(jInterp, aLoL));
    } else if (isBoolean(aLoL)) {
      lua_pushboolean(lstate, (isTrue(aLoL) ? TRUE : FALSE));
    } else {
      lua_pushstring(lstate, "unknownAtom");
    }
  } else {
    lua_newtable(lstate);
    for (size_t i = 1; aLoL; i++) {
      peekJoylolToLua(lstate, jInterp, asCar(aLoL));
      lua_rawseti(lstate, -2, i);
```

```
      aLoL = asCdr(aLoL);
    }
  }
}
```

3.11.10.1 Test Suite: push/pop root context data

CCode : lua

```
static int lua_joylol_pushData(lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  pushLuaToJoylol(lstate, jInterp->rootCtx, TRUE);
  lua_pop(lstate, 1);
  return 0;
}

static int lua_joylol_pushNullData(lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  pushNullCtxData(jInterp->rootCtx);
  return 0;
}

static int lua_joylol_pushBooleanData(lua_State *lstate) {
  Boolean aBool = lua_toboolean(lstate, 1);
  getJoyLoLInterpInto(lstate, jInterp);
  pushBooleanCtxData(jInterp->rootCtx, aBool);
  lua_pop(lstate, 1);
  return 0;
}

static int lua_joylol_pushNaturalData(lua_State *lstate) {
  double aDouble = lua_tonumber(lstate, 1);
  getJoyLoLInterpInto(lstate, jInterp);
  pushNaturalCtxData(jInterp->rootCtx, (size_t)aDouble);
  lua_pop(lstate, 1);
  return 0;
}

static int lua_joylol_pushSymbolData(lua_State *lstate) {
  Symbol *aString = lua_tostring(lstate, 1);
  getJoyLoLInterpInto(lstate, jInterp);
  pushSymbolCtxData(jInterp->rootCtx, aString);
  lua_pop(lstate, 1);
```

```
35    return 0;
36  }
37
38  static int lua_joylol_pushParsedStringData(lua_State *lstate) {
39    Symbol *aString = lua_tostring(lstate, 1);
40    getJoyLoLInterpInto(lstate, jInterp);
41    pushParsedStringCtxData(jInterp->rootCtx, aString);
42    lua_pop(lstate, 1);
43    return 0;
44  }
45
46  static int lua_joylol_peekData(lua_State *lstate) {
47    getJoyLoLInterpInto(lstate, jInterp);
48    JObj *aLoL = popCtxData(jInterp->rootCtx);
49    peekJoylolToLua(lstate, jInterp, aLoL);
50    pushCtxData(jInterp->rootCtx, aLoL);
51    return 1;
52  }
53
54  static int lua_joylol_popData(lua_State *lstate) {
55    getJoyLoLInterpInto(lstate, jInterp);
56    JObj *aLoL = popCtxData(jInterp->rootCtx);
57    peekJoylolToLua(lstate, jInterp, aLoL);
58    return 1;
59  }
60
61  static int lua_joylol_clearData(lua_State *lstate) {
62    getJoyLoLInterpInto(lstate, jInterp);
63    clearCtxData(jInterp->rootCtx);
64    return 0;
65  }
66
67  static int lua_joylol_showData(lua_State *lstate) {
68    getJoyLoLInterpInto(lstate, jInterp);
69    StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
70    showCtxData(jInterp->rootCtx, aStrBuf);
71    lua_pushstring (lstate, getCString(aStrBuf));
72    strBufClose(aStrBuf);
73    return 1;
74  }
```

Test case

should be able to push and pop on data

```
local joylol = thirddata.joylol
joylol.pushBooleanData(true)
local result = joylol.popData()
assert.isBoolean(result)
assert.isTrue(result)

joylol.pushBooleanData(false)
result = joylol.popData()
assert.isBoolean(result)
assert.isFalse(result)

joylol.pushNaturalData(1234.4999)
result = joylol.popData()
assert.isNumber(result)
-- note how we round down here
assert.isEqual(result, 1234)

joylol.pushSymbolData("this is a test")
result = joylol.popData()
assert.isString(result)
assert.isEqual(result, "this is a test")

joylol.pushParsedStringData("hello there ( this is a test ) 1234")
result = joylol.peekData()
assert.isTable(result)
assert.length(result, 4)
assert.isString(result[1])
assert.isEqual(result[1], "hello")
assert.isString(result[2])
assert.isEqual(result[2], "there")
local innerResult = result[3]
assert.isTable(innerResult)
assert.length(innerResult, 4)
assert.isEqual(innerResult[1], "this")
assert.isEqual(innerResult[2], "is")
assert.isEqual(innerResult[3], "a")
assert.isEqual(innerResult[4], "test")
assert.isNumber(result[4])
assert.isEqual(result[4], 1234)
joylol.clearData()
```

```
result = joylol.popData()
assert.isNil(result)
```

PASSED

3.11.10.2 Test Suite: push/pop root context process

CCode : lua

```
1  static int lua_joylol_pushProcess(lua_State *lstate) {
2    getJoyLoLInterpInto(lstate, jInterp);
3    pushLuaToJoylol(lstate, jInterp->rootCtx, FALSE);
4    lua_pop(lstate, 1);
5    return 0;
6  }
7
8  static int lua_joylol_pushNullProcess(lua_State *lstate) {
9    getJoyLoLInterpInto(lstate, jInterp);
10   pushNullCtxProcess(jInterp->rootCtx);
11   return 0;
12 }
13
14 static int lua_joylol_pushBooleanProcess(lua_State *lstate) {
15   Boolean aBool = lua_toboolean(lstate, 1);
16   getJoyLoLInterpInto(lstate, jInterp);
17   pushBooleanCtxProcess(jInterp->rootCtx, aBool);
18   lua_pop(lstate, 1);
19   return 0;
20 }
21
22 static int lua_joylol_pushNaturalProcess(lua_State *lstate) {
23   double aDouble = lua_tonumber(lstate, 1);
24   getJoyLoLInterpInto(lstate, jInterp);
25   pushNaturalCtxProcess(jInterp->rootCtx, (size_t)aDouble);
26   lua_pop(lstate, 1);
27   return 0;
28 }
29
30 static int lua_joylol_pushSymbolProcess(lua_State *lstate) {
31   Symbol *aString = lua_tostring(lstate, 1);
32   getJoyLoLInterpInto(lstate, jInterp);
33   pushSymbolCtxProcess(jInterp->rootCtx, aString);
34   lua_pop(lstate, 1);
35   return 0;
```

```
36  }
37
38  static int lua_joylol_pushParsedStringProcess(lua_State *lstate) {
39    Symbol *aString = lua_tostring(lstate, 1);
40    getJoyLoLInterpInto(lstate, jInterp);
41    pushParsedStringCtxProcess(jInterp->rootCtx, aString);
42    lua_pop(lstate, 1);
43    return 0;
44  }
45
46  static int lua_joylol_peekProcess(lua_State *lstate) {
47    getJoyLoLInterpInto(lstate, jInterp);
48    JObj *aLoL = popCtxProcess(jInterp->rootCtx);
49    peekJoylolToLua(lstate, jInterp, aLoL);
50    pushCtxProcess(jInterp->rootCtx, aLoL);
51    return 1;
52  }
53
54  static int lua_joylol_popProcess(lua_State *lstate) {
55    getJoyLoLInterpInto(lstate, jInterp);
56    JObj *aLoL = popCtxProcess(jInterp->rootCtx);
57    peekJoylolToLua(lstate, jInterp, aLoL);
58    return 1;
59  }
60
61  static int lua_joylol_clearProcess(lua_State *lstate) {
62    getJoyLoLInterpInto(lstate, jInterp);
63    clearCtxProcess(jInterp->rootCtx);
64    return 0;
65  }
66
67  static int lua_joylol_showProcess(lua_State *lstate) {
68    getJoyLoLInterpInto(lstate, jInterp);
69    StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
70    showCtxProcess(jInterp->rootCtx, aStrBuf);
71    lua_pushstring (lstate, getCString(aStrBuf));
72    strBufClose(aStrBuf);
73    return 1;
74  }
```

Test case

should be able to push and pop on process

```
local joylol = thirddata.joylol
joylol.pushBooleanProcess(true)
local result = joylol.popProcess()
assert.isBoolean(result)
assert.isTrue(result)

joylol.pushBooleanProcess(false)
result = joylol.popProcess()
assert.isBoolean(result)
assert.isFalse(result)

joylol.pushNaturalProcess(1234.56)
result = joylol.popProcess()
assert.isNumber(result)
-- note how we round up here
assert.isEqual(result, 1235)

joylol.pushSymbolProcess("this is a test")
result = joylol.popProcess()
assert.isString(result)
assert.isEqual(result, "this is a test")

joylol.pushParsedStringProcess("hello there ( this is a test ) 1234")
result = joylol.peekProcess()
assert.isTable(result)
assert.length(result, 4)
assert.isString(result[1])
assert.isEqual(result[1], "hello")
assert.isString(result[2])
assert.isEqual(result[2], "there")
local innerResult = result[3]
assert.isTable(innerResult)
assert.length(innerResult, 4)
assert.isEqual(innerResult[1], "this")
assert.isEqual(innerResult[2], "is")
assert.isEqual(innerResult[3], "a")
assert.isEqual(innerResult[4], "test")
assert.isNumber(result[4])
assert.isEqual(result[4], 1234)
joylol.clearProcess()
```

```
result = joylol.popProcess()
assert.isNil(result)
```

LuaTest FAILED:
Could not execute the LuaTest.
Expected 1234.0 to equal 1235.
in file: /home/stg/ExpositionGit/tools/conTeXt/joylol-c/base/jInterps/doc/luaInterface.tex between lines 401 and 446

CCode : lua

```
1  static int lua_joylol_evalString(lua_State *lstate) {
2    Symbol *aString = lua_tostring(lstate, 1);
3    getJoyLoLInterpInto(lstate, jInterp);
4    evalStringInContext(jInterp->rootCtx, aString);
5    lua_pop(lstate, 1);
6    return 0;
7  }
8
9  static int lua_joylol_eval(lua_State *lstate) {
10   getJoyLoLInterpInto(lstate, jInterp);
11   evalContext(jInterp->rootCtx);
12   return 0;
13 }
```

CCode : lua

```
1  #define NumJoyLoLInterfaceFunctions 15
2  static const struct luaL_Reg lua_joylol_interface [] = {
3    { "gitVersion",              lua_jInterps_getGitVersion         },
4    { "setVerbose",              lua_joylol_setVerbose              },
5    { "setDebugging",            lua_joylol_setDebugging            },
6    { "setTracing",              lua_joylol_setTracing              },
7    { "setShowStack",            lua_joylol_setShowStack            },
8    { "setShowSpecifications",   lua_joylol_setShowSpecifications   },
9    { "setChecking",             lua_joylol_setChecking             },
10   { "pushLoadPath",            lua_joylol_pushLoadPath            },
11   { "loadFile",                lua_joylol_loadFile                },
12   { "pushData",                lua_joylol_pushData                },
13   { "pushNullData",            lua_joylol_pushNullData            },
14   { "pushBooleanData",         lua_joylol_pushBooleanData         },
15   { "pushNaturalData",         lua_joylol_pushNaturalData         },
16   { "pushSymbolData",          lua_joylol_pushSymbolData          },
17   { "pushParsedStringData",    lua_joylol_pushParsedStringData    },
18   { "peekData",                lua_joylol_peekData                },
```

```
  { "popData",                lua_joylol_popData                },
  { "clearData",              lua_joylol_clearData              },
  { "showData",               lua_joylol_showData               },
  { "pushProcess",            lua_joylol_pushProcess            },
  { "pushNullProcess",        lua_joylol_pushNullProcess        },
  { "pushBooleanProcess",     lua_joylol_pushBooleanProcess     },
  { "pushNaturalProcess",     lua_joylol_pushNaturalProcess     },
  { "pushSymbolProcess",      lua_joylol_pushSymbolProcess      },
  { "pushParsedStringProcess", lua_joylol_pushParsedStringProcess },
  { "peekProcess",            lua_joylol_peekProcess            },
  { "popProcess",             lua_joylol_popProcess             },
  { "clearProcess",           lua_joylol_clearProcess           },
  { "showProcess",            lua_joylol_showProcess            },
  { "evalString",             lua_joylol_evalString             },
  { "eval",                   lua_joylol_eval                   },
  {NULL, NULL}
};
```

3.11.11 Lua initialization functions

3.11.11.1 Test Suite: set

CHeader : public

```
#define setLuaPath(lstate, aLuaPath, addToCPath)  \
  lua_getglobal(lstate, "package");               \
  lua_pushstring(lstate, aLuaPath);               \
  lua_setfield(lstate, -2,                        \
    (addToCPath ? "cpath" : "path"));             \
  lua_pop(lstate, 1)
#define addLuaPath(lstate, aLuaPath, addToCPath)              \
for(size_t iAddLuaPath = 0; iAddLuaPath < 1; iAddLuaPath++) { \
  luaL_Buffer lBuf;                                           \
  luaL_buffinit(lstate, &lBuf);                               \
  luaL_addstring(&lBuf, aLuaPath);                            \
  luaL_addstring(&lBuf, ";");                                 \
  lua_getglobal(lstate, "package");                           \
  lua_getfield(lstate, -1,                                    \
    (addToCPath ? "cpath" : "path"));                         \
  luaL_addvalue (&lBuf);                                      \
  luaL_pushresult(&lBuf);                                     \
  lua_setfield(lstate, -2,                                    \
```

```
19      (addToCPath ? "cpath" : "path"));                              \
20    lua_pop(lstate, 1);                                              \
21  }
22  #define getLuaPathInto(lstate, result, getCPath)  \
23    lua_getglobal(lstate, "package");               \
24    lua_getfield(lstate, -1,                        \
25      (getCPath ? "cpath" : "path"));               \
26    Symbol *result =                                \
27      strdup(lua_tostring(lstate, -1));             \
28    lua_pop(lstate, 2)
29  #define PATH_BUFFER_SIZE  8000
30  #define buildCWDPathInto(buffer, aDir)                  \
31    char *buffer = (char*)calloc(1, PATH_BUFFER_SIZE);    \
32    if (!getcwd(buffer, PATH_BUFFER_SIZE)) {              \
33      /*return*/ luaL_error(lstate, "%s%s%s",             \
34        "\nERROR:\n",                                     \
35        "  Path buffer size too small\n",                 \
36        "  while building CWD path\n\n");                 \
37    }                                                     \
38    strncat(buffer, "/",                                  \
39      PATH_BUFFER_SIZE - strlen(buffer) - 1);             \
40    strncat(buffer, aDir,                                 \
41      PATH_BUFFER_SIZE - strlen(buffer) - 1)
42  #define addBuildLuaPath(lstate)             \
43    buildCWDPathInto(buildLuaPath,            \
44      "buildDir/?.lua");                      \
45    addLuaPath(lstate, buildLuaPath, FALSE); \
46    free(buildLuaPath);                       \
47    buildCWDPathInto(buildCPath,              \
48      "buildDir/?.so");                       \
49    addLuaPath(lstate, buildCPath, TRUE);     \
50    free(buildCPath)

51  #define buildHomePathInto(buffer, homeBuffer, aDir)     \
52    char *buffer = (char*)calloc(1, PATH_BUFFER_SIZE);    \
53    char *homeBuffer = getenv("HOME");                    \
54    if (!homeBuffer) {                                    \
55      /*return*/ luaL_error(lstate, "%s",                 \
56        "\nERROR:\n",                                     \
57        "  Environment variable 'HOME'\n",                \
58        "  is empty!\n\n");                               \
59    }                                                     \
60    strncat(buffer, homeBuffer,                           \
```

```
61      PATH_BUFFER_SIZE - strlen(buffer) - 1);              \
62    strncat(buffer, "/",                                  \
63      PATH_BUFFER_SIZE - strlen(buffer) - 1);              \
64    strncat(buffer, aDir,                                 \
65      PATH_BUFFER_SIZE - strlen(buffer) - 1)
66  #define addJoyLoLLuaPath(lstate)                  \
67    buildHomePathInto(joyLoLLuaPath, homeLuaPath,   \
68      ".joylol/?.lua");                             \
69    addLuaPath(lstate, joyLoLLuaPath, FALSE);       \
70    free(joyLoLLuaPath);                            \
71    buildHomePathInto(joyLoLCPath, homeCPath,       \
72      ".joylol/?.so");                              \
73    addLuaPath(lstate, joyLoLCPath, TRUE);          \
74    free(joyLoLCPath)
```

Test case

should set, add and get package.path

```
  getLuaPathInto(lstate, origPath, FALSE);
  AssertPtrNotNull(origPath);
  AssertPtrNotNull(strstr(origPath, "?.lua"));

  addLuaPath(lstate, "tests", FALSE);
  getLuaPathInto(lstate, testPath, FALSE);
  AssertPtrNotNull(testPath);

  AssertIntZero(strncmp(testPath, "tests;", strlen("tests;")));

  setLuaPath(lstate, origPath, FALSE);
  getLuaPathInto(lstate, newTestPath, FALSE);
  AssertPtrNotNull(newTestPath);
  AssertIntZero(strcmp(origPath, newTestPath));

  free((void*)testPath);
  free((void*)newTestPath);
  free((void*)origPath);
```

Test case

should set, add and get package.cpath

```
  getLuaPathInto(lstate, origPath, TRUE);
```

```
AssertPtrNotNull(origPath);
AssertPtrNotNull(strstr(origPath, "?.so"));

addLuaPath(lstate, "tests", TRUE);
getLuaPathInto(lstate, testPath, TRUE);
AssertPtrNotNull(testPath);

AssertIntZero(strncmp(testPath, "tests;", strlen("tests;")));

setLuaPath(lstate, origPath, TRUE);
getLuaPathInto(lstate, newTestPath, TRUE);
AssertPtrNotNull(newTestPath);
AssertIntZero(strcmp(origPath, newTestPath));

free((void*)testPath);
free((void*)newTestPath);
free((void*)origPath);
```

3.11.11.2 Test Suite: require lua modules

CHeader : public

```
#define requireLuaModule(lstate, aLuaModule)  \
  lua_getglobal(lstate, "require");           \
  lua_pushstring(lstate, aLuaModule);         \
  if (lua_pcall(lstate, 1, 1, 0)) {           \
    /* there was an error...                  \
     * so return a copy of the error message  \
     */                                       \
    /*return*/ luaL_error(lstate,             \
      "Failed to load [%s]\n%s%s\n",          \
      aLuaModule,                             \
      "ERROR:\n",                             \
      lua_tostring(lstate, -1));              \
  }                                           \
  lua_pop(lstate, 1)
```

CHeader : public

```
#define requireLuaModuleInto(lstate, aLuaModule, result)  \
  Boolean result = TRUE;                                  \
  lua_getglobal(lstate, "require");                       \
  lua_pushstring(lstate, aLuaModule);                     \
```

```
  if (lua_pcall(lstate, 1, 1, 0)) {                          \
    /* there was an error... */                              \
    result = FALSE;                                          \
  }                                                          \
  lua_pop(lstate, 1)
```

CHeader : public

```
// modified from: Joe Rossi's initial question
// http://lua-users.org/lists/lua-l/2006-03/msg00335.html
#define stackDump(lstate, message)                           \
  {                                                          \
    int i = lua_gettop(lstate);                              \
    printf("%s ---- Stack Dump Started  ----\n",  \
      message);                                              \
    while( i ) {                                             \
      int aType = lua_type(lstate, i);                       \
      switch (aType) {                                       \
      case LUA_TSTRING:                                      \
        printf("%d: [%s] (string)\n",                        \
          i, lua_tostring(lstate, i));                       \
        break;                                               \
      case LUA_TBOOLEAN:                                     \
        printf("%d: %s (boolean)\n",                         \
          i, (lua_toboolean(lstate, i) ?                     \
            "true" : "false"));                              \
        break;                                               \
      case LUA_TNUMBER:                                      \
        printf("%d: %g (number)\n",                          \
          i, lua_tonumber(lstate, i));                       \
        break;                                               \
      case LUA_TTABLE:                                       \
        printf("%d: (%s)\n",                                 \
          i, lua_typename(lstate, aType));                   \
        lua_pushnil(lstate); /* first key */                 \
        while (lua_next(lstate, i) != 0) {                   \
          /* duplicate the key */                            \
          lua_pushvalue (lstate, -2);                        \
          /* print:                              */          \
          /*  tostring of duplicate key at -1 */            \
          /*  type of value             at -2 */            \
          /*  type of key               at -3 */            \
          printf("   %s (%s) - %s\n",                       \
              lua_tostring(lstate, -1),                      \
```

```
            lua_typename(lstate,                \
              lua_type(lstate, -3)),            \
            lua_typename(lstate,                \
              lua_type(lstate, -2)));           \
        /* removes dupliate 'key' & 'value'; */ \
        /* keeps 'key' for next iteration    */ \
        lua_pop(lstate, 2);                     \
      }                                         \
      break;                                    \
    default:                                    \
      printf("%d: (%s)\n",                      \
        i, lua_typename(lstate, aType));        \
      break;                                    \
    }                                           \
    i--;                                        \
  }                                             \
  printf("%s ---- Stack Dump Finished ----\n",  \
    message);                                   \
}
```

CCode : lua

```
static void checkAllCoAlgs(
  lua_State    *lstate,
  JoyLoLInterp *jInterp
) {
  // first check that core stdout and stderr are defined
  if (!(jInterp->writeStdOut) || !(jInterp->writeStdErr)) {
    /*return*/ luaL_error(lstate, "%s%s",
      "\nERROR:\n",
      "  Core output methods are missing\n");
  }

  // now check that all required CoAlg extensions
  // have been registered
  //
  RequiredObjects *curCoAlg = requiredCoAlgs;
  for ( ; curCoAlg->name ; curCoAlg++) {
    size_t curTag = curCoAlg->tag;
    if (!getJClass(jInterp, curTag)) {
      /*return*/ luaL_error(lstate, "%s%s%s%s\n\n",
        "\nERROR:\n",
       "  Missing a required CoAlgebraic extension\n",
       "  need to require joylol.",
```

```
22        curCoAlg->name);
23      }
24    }

25    // Now check that all CoAlg extensions
26    // have the required callbacks
27    //
28    for (size_t i = 1; i < jInterp->numCoAlgs; i++ ) {
29      if (jInterp->coAlgs[i]) {
30        JClass *aCoAlg = getJClass(jInterp, i);
31        assert(0 < aCoAlg->objectSize);
32        assert(aCoAlg->initializeFunc);
33        assert(aCoAlg->registerFunc);
34      }
35    }

36    // finally check that all important interpreter
37    // objects have been assigned
38    if (
39      !(jInterp->loader) ||
40      !(jInterp->rootCtx)
41    ) {
42        /*return*/ luaL_error(lstate, "%s%s",
43          "\nERROR:\n",
44          "  some required objects are missing\n\n");
45    }
46  }
47
48  static int lua_jInterps_checkAllCoAlgs(lua_State *lstate) {
49    getJoyLoLInterpInto(lstate, jInterp);
50    assert(jInterp);
51    checkAllCoAlgs(lstate, jInterp);
52    return 0;
53  }
54
55  static int lua_jInterps_initializeAllLoaded(lua_State *lstate) {
56    getJoyLoLInterpInto(lstate, jInterp);
57    assert(jInterp);
58    initializeAllLoadedImpl(lstate, jInterp);
59    return 0;
60  }
```

CCode : lua

```
static const struct luaL_Reg lua_jInterps [] = {
  {"gitVersion",          lua_jInterps_getGitVersion},
  {"initializeAllLoaded", lua_jInterps_initializeAllLoaded},
  {"checkAllCoAlgs",      lua_jInterps_checkAllCoAlgs},
  {NULL, NULL}
};
```

CCode : lua

```
static void requireAllRequiredCoAlgs(
  lua_State    *lstate,
  JoyLoLInterp *jInterp
) {
  assert(lstate);
  assert(jInterp);
  assert(jInterp->coAlgs);

  // start by creating a master table
  lua_createtable(lstate, 0,
    NumRequiredCoAlgs + NumJoyLoLInterfaceFunctions + 5
  );                                                    // -3
  // add the joylol interface into the master (joylol) table
  luaL_setfuncs(lstate, lua_joylol_interface, 0);

  // now place jInterps into the master table
  lua_pushstring(lstate, "jInterps");                   // -2
  luaL_newlib(lstate, lua_jInterps);                    // -1
  lua_settable(lstate, -3);

  // now walk through the required coAlgs
  // requiring each in turn and
  // placing it into the master table
  RequiredObjects *curCoAlg = requiredCoAlgs;
  for ( ; curCoAlg->name ; curCoAlg++) {
    size_t curTag = curCoAlg->tag;
    if (!(jInterp->coAlgs[curTag])) {
      if (jInterp->verbose) {
        char output[1000];
        memset(output, 0, 1000);
        snprintf(output, 999,
          "    loading [joylol.%s]\n",
          curCoAlg->name);
        jInterp->writeStdOut(jInterp, output);
      }
```

```
33      lua_pushstring(lstate, curCoAlg->name); // -2
34      lua_getglobal(lstate, "require");
35      luaL_Buffer modName;
36      luaL_buffinit(lstate, &modName);
37      luaL_addstring(&modName, "joylol.");
38      luaL_addstring(&modName, curCoAlg->name);
39      luaL_pushresult(&modName);
40      if (lua_pcall(lstate, 1, 1, 0)) {       // -> -1
41        /* return */ luaL_error(lstate,
42          "Failed to load [%s]\n%s%s\n",
43          curCoAlg->name,
44          "ERROR:\n",
45          lua_tostring(lstate, -1)
46        );
47      }
48      lua_settable(lstate, -3);
49      if (jInterp->verbose) {
50        char output[1000];
51        memset(output, 0, 1000);
52        snprintf(output, 999,
53          "    loaded [joylol.%s]\n",
54          curCoAlg->name);
55        jInterp->writeStdOut(jInterp, output);
56      }
57    }
58  }
59 }
```

Now each shared library needs to implement the following code to provide an interface between the generic C-implementation and the Lua module for the shared library.

CCode : lua

```
1  int luaopen_joylol_jInterps (lua_State *lstate) {
2    setJoyLoLInterpInto(lstate, jInterp);
3    if (jInterp->verbose) {
4      jInterp->writeStdOut(jInterp,
5        "  loading [joylol.jInterps]\n"
6      );
7    }
8    registerJInterps(jInterp);
9    requireAllRequiredCoAlgs(lstate, jInterp);
10   initializeAllLoaded(lstate, jInterp);
11   registerAllLoaded(lstate, jInterp);
```

```
  checkAllCoAlgs(lstate, jInterp);
  if (jInterp->verbose) {
    jInterp->writeStdOut(jInterp,
      "  loaded [joylol.jInterps]\n"
    );
  }
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedJInterps` does just this.

CHeader : public

```
extern Boolean requireStaticallyLinkedJInterps(
  lua_State *lstate
);
```

CCode : lua

```
Boolean requireStaticallyLinkedJInterps(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");

  // fake a luaopen_joylol_jInterps
  // BUT without the requireAllRequiredCoAlgs
  // AND without the checkAllRequired
  // AND without the initializeAllLoaded
  setJoyLoLInterpInto(lstate, jInterp);
  registerJInterps(jInterp);
  luaL_newlib(lstate, lua_jInterps);

  lua_setfield(lstate, -2, "joylol.jInterps");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.11.12 JoyLoL's Lua script

LuaCode : default

```
-- some code
```

3.11.13 Conclusions

LuaCode : default

```
return joylol
```

Before we actually we write out the `CoAlgs` C-header file, provide some standard definitions, the most important of which deal with memory allocation and alignment as well as a simple `DEBUG` system.

CHeader : public

```
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include <assert.h>
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>
#ifndef JOYLOL_SYSTEM_CONFIG_PATH
#define JOYLOL_SYSTEM_CONFIG_PATH "/usr/local/etc/joyLoL"
#endif
#ifndef JOYLOL_SYSTEM_COALG_PATH
#define JOYLOL_SYSTEM_COALG_PATH "/usr/local/lib/joyLoL"
#endif
#ifndef JOYLOL_COALGS_INCREMENT
#define JOYLOL_COALGS_INCREMENT 10
#endif
#ifndef JOYLOL_COMPILERS_INCREMENT
#define JOYLOL_COMPILERS_INCREMENT 10
#endif

#ifndef NULL
#define NULL (void*)0
#endif
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifdef __LP64__
```

```
30 #define MEM_ALIGNMENT ((size_t)0x7)
31 #else
32 #define MEM_ALIGNMENT ((size_t)0x3)
33 #endif
34 #define IS_MEM_ALIGNED(someMem) \
35   (!( ((size_t)(someMem)) & (MEM_ALIGNMENT) ))
36 #define joyLoLCalloc(numItems, itemType) \
37   (itemType*)calloc((numItems), sizeof(itemType))
38 #ifndef NDEBUG
39 #include <stdio.h>
40 #define setDebugging(jInterp, aBool)  \
41   assert(jInterp);                    \
42   jInterp->debug = aBool
43 #define getDebuggingInto(jInterp, aVar) \
44   assert(jInterp);                      \
45   Boolean aVar = jInterp->debug
46 #define DEBUG(jInterp, format, ... )              \
47   assert(jInterp);                                \
48   if (jInterp->debug) {                           \
49     char output[1000];                            \
50     memset(output, 0, 1000);                      \
51     snprintf(output, 999, format, __VA_ARGS__ );  \
52     jInterp->writeStdOut(jInterp, output);        \
53   }
54 #else
55 #define startDebugging(jInterp)
56 #define stopDebuggging(jInterp)
57 #define DEBUG(jInterp, format, ...)
58 #endif
```

CHeader : public

CHeader : private

CCode : objects

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include <assert.h>
4 #include "joylol/jInterps.h"
5 #include "joylol/jInterps-private.h"
```

CCode : interpreter

```
1 #include <stdlib.h>
2 #include <assert.h>
```

```
3  #include <joylol/jInterps.h>
4  #include <joylol/cFunctions.h>
5  #include <joylol/stringBuffers.h>
6  #include <joylol/symbols.h>
7  #include <joylol/texts.h>
8  #include <joylol/assertions.h>
9  #include <joylol/contexts.h>
10 #include "joylol/crossCompilers.h"
11 #include "joylol/jInterps-private.h"
```

CCode : words

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include "joylol/jInterps.h"
5  #include "joylol/jInterps-private.h"
```

CCode : lua

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include "joylol/jInterps.h"
5  #include "joylol/booleans.h"
6  #include "joylol/naturals.h"
7  #include "joylol/symbols.h"
8  #include "joylol/stringBuffers.h"
9  #include "joylol/pairs.h"
10 #include "joylol/cFunctions.h"
11 #include <joylol/assertions.h>
12 #include "joylol/texts.h"
13 #include "joylol/parsers.h"
14 #include "joylol/contexts.h"
15 #include "joylol/loaders.h"
16 #include "joylol/jInterps-private.h"
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.stringBuffers");

getJoyLoLInterpInto(lstate, jInterp);
```

Lmsfile : default
Lmsfile : default
Lmsfile : default

3.12 Loaders

3.12.1 Goals

A Loader loads JoyLoL and Lua files into a given Context.

3.12.2 Loader functions

CHeader : public

```c
typedef struct loader_object_struct {
  JObj      super;
  JoyLoLInterp *jInterp;
  JObj     *paths;
  JObj     *extensions;
} LoaderObj;
```

CHeader : public

```c
typedef LoaderObj *(NewLoader)(
  JoyLoLInterp *jInterp
);

#define newLoader(jInterp)          \
  (                                 \
    assert(getLoadersClass(jInterp) \
      ->newLoaderFunc),             \
    (getLoadersClass(jInterp)       \
      ->newLoaderFunc(jInterp))     \
  )
```

CHeader : private

```c
extern LoaderObj *newLoaderImpl(
  JoyLoLInterp *jInterp
);
```

CCode : lua

```c
LoaderObj *newLoaderImpl(
  JoyLoLInterp *jInterp
) {
  assert(jInterp);
  LoaderObj *loader =
    (LoaderObj*)newObject(jInterp, LoadersTag);
```

```
7   assert(loader);
8   loader->jInterp    = jInterp;
9   loader->paths      = NULL;
10  loader->extensions = NULL;
11
12  // get the lua state associated with this jInterp
13  lua_State *lstate = jInterp->lstate;
14  assert(lstate);
15
16  // get the method to get the core resources (XXXpaths)
17  getJoyLoLCallbackInto(lstate, getCoreResources);
18  assert(getCoreResources);

19  // init standard load paths
20  // allow absolute paths as a last resort
21  pushLoadPath(loader, "");
22  pushLoadPath(loader,
23    getCoreResources(lstate, JoyLoLCallback_SystemPath));
24  pushLoadPath(loader,
25    getCoreResources(lstate, JoyLoLCallback_LocalPath));
26  pushLoadPath(loader,
27    getCoreResources(lstate, JoyLoLCallback_UserPath));
28  pushLoadPath(loader, ".");
29
30  // init standard load extensions
31  // allow absolute extensions as a last resort
32  pushLoadExtension(loader, "");
33  pushLoadExtension(loader, ".joy");

34  return loader;
35 }
```

CHeader : public

```
1  typedef void (PushLoadSymbol)(
2    LoaderObj *loader,
3    Symbol    *aLoadSym
4  );
5
6  #define pushLoadPath(loader, aLoadPath)       \
7    (                                           \
8      assert(getLoadersClass(loader->jInterp)   \
9        ->pushLoadPathFunc),                    \
10     (getLoadersClass(loader->jInterp)         \
```

```
11      ->pushLoadPathFunc(loader, aLoadPath))  \
12    )
```

CHeader : private

```
1  extern void pushLoadPathImpl(
2    LoaderObj *loader,
3    Symbol    *aLoadPath
4  );
```

CCode : lua

```
1  void pushLoadPathImpl(
2    LoaderObj *loader,
3    Symbol    *aLoadPath
4  ) {
5    assert(aLoadPath);
6    assert(loader);
7    loader->paths =
8      newPair(
9        loader->jInterp,
10       newSymbol(loader->jInterp, aLoadPath, "loadPath", 0),
11       loader->paths
12     );
13 }
```

CHeader : public

```
1  typedef void (ListLoader)(
2    LoaderObj       *loader,
3    StringBufferObj *aStrBuf
4  );
5
6  #define listLoadPaths(loader, aStrBuf)      \
7    (                                         \
8      assert(getLoadersClass(loader->jInterp) \
9        ->listLoadPathsFunc),                 \
10     (getLoadersClass(loader->jInterp)       \
11       ->listLoadPathsFunc(loader, aStrBuf)) \
12   )
```

CHeader : private

```
1  extern void listLoadPathsImpl(
2    LoaderObj       *loader,
3    StringBufferObj *aStrBuf
```

```
);
```

CCode : lua

```
void listLoadPathsImpl(
  LoaderObj       *loader,
  StringBufferObj *aStrBuf
) {
  assert(loader);
  assert(aStrBuf);
  JObj* pathList = loader->paths;
  while(pathList && isPair(pathList)) {
    JObj* aPath = asCar(pathList);
    if (isSymbol(aPath)) {
      strBufPrintf(aStrBuf,
        "%s\n", asSymbol(aPath)
      );
    }
    pathList = asCdr(pathList);
  }
}
```

CHeader : public

```
#define pushLoadExtension(loader, aLoadExt)       \
  (                                               \
    assert(getLoadersClass(loader->jInterp)       \
      ->pushLoadExtensionFunc),                   \
    (getLoadersClass(loader->jInterp)             \
      ->pushLoadExtensionFunc(loader, aLoadExt))  \
  )
```

CHeader : private

```
extern void pushLoadExtensionImpl(
  LoaderObj *loader,
  Symbol    *aLoadExtension
);
```

CCode : lua

```
void pushLoadExtensionImpl(
  LoaderObj *loader,
  Symbol    *aLoadExtension
) {
  assert(aLoadExtension);
```

```
  assert(loader);
  loader->extensions =
    newPair(
      loader->jInterp,
      newSymbol(loader->jInterp, aLoadExtension, "loadExtension", 0),
      loader->extensions
    );
}
```

CHeader : public

```
#define listLoadExtensions(loader, aStrBuf)      \
  (                                              \
    assert(getLoadersClass(loader->jInterp)      \
      ->listLoadExtensionsFunc),                 \
    (getLoadersClass(loader->jInterp)            \
      ->listLoadExtensionsFunc(loader, aStrBuf)) \
  )
```

CHeader : private

```
extern void listLoadExtensionsImpl(
  LoaderObj       *loader,
  StringBufferObj *aStrBuf
);
```

CCode : lua

```
void listLoadExtensionsImpl(
  LoaderObj       *loader,
  StringBufferObj *aStrBuf
) {
  assert(loader);
  assert(aStrBuf);
  JObj* extensionList = loader->extensions;
  while(extensionList && isPair(extensionList)) {
    JObj* anExtension = asCar(extensionList);
    if (isSymbol(anExtension)) {
      strBufPrintf(aStrBuf,
        "%s\n", asSymbol(anExtension)
      );
    }
    extensionList = asCdr(extensionList);
  }
```

```
}
```

CHeader : private

```
extern Boolean loadAJoyLoLFile(
  ContextObj *aCtx,
  Symbol     *filePath
);
```

CCode : lua

```
Boolean loadAJoyLoLFile(
  ContextObj *aCtx,
  Symbol     *filePath
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  DEBUG(jInterp, "loadAJoyLoLFile [%s]\n", filePath);
  FILE* inputFile = fopen(filePath, "r");
  if (inputFile) {
    if(aCtx->verbose) {
      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
      strBufPrintf(aStrBuf, "(%s)\n", filePath);
      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
      strBufClose(aStrBuf);
    }
    TextObj* aText =
      createTextFromInputFile(jInterp, inputFile, filePath);
    evalTextInContext(aCtx, aText);
    freeText(aText);
    fclose(inputFile);
    if (reportException(aCtx)) {
      DEBUG(jInterp, "loadAJoyLoLFile [%s] RAISED EXCEPTION\n", filePath);
      return FALSE;
    }
    DEBUG(jInterp, "loadAJoyLoLFile [%s] OK\n", filePath);
    return TRUE; // we have loaded this file so return TRUE;
  }
  DEBUG(jInterp, "loadAJoyLoLFile [%s] FAILED\n", filePath);
  return FALSE;
}
```

CHeader : public

```
typedef Boolean (LoadAFile)(
  ContextObj *aCtx,
  Symbol     *aFileToLoad
);

#define loadAFile(aCtx, aFileToLoad)       \
  (                                        \
    assert(aCtx),                          \
    assert(getLoadersClass(aCtx->jInterp)  \
      ->loadAFileFunc),                    \
    (getLoadersClass(aCtx->jInterp)        \
      ->loadAFileFunc(aCtx, aFileToLoad))  \
  )
```

CHeader : private

```
extern Boolean loadAFileImpl(
  ContextObj *aCtx,
  Symbol     *aFileToLoad
);
```

CCode : lua

```
Boolean loadAFileImpl(
  ContextObj *aCtx,
  Symbol     *aFileToLoad
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  LoaderObj* loader = aCtx->jInterp->loader;
  assert(loader);
  if (!aFileToLoad) return FALSE;
  if (!*aFileToLoad) return FALSE;

  if (aCtx->verbose) {
    StringBufferObj *aStrBuf = newStringBuffer(aCtx);
    strBufPrintf(aStrBuf, "\t%s ", aFileToLoad);
    jInterp->writeStdOut(jInterp, getCString(aStrBuf));
    strBufClose(aStrBuf);
  }

  JObj* aLoadPathList = loader->paths;
  while(aLoadPathList) {
```

```
22    assert(isPair(aLoadPathList));
23    JObj* aLoadPath = asCar(aLoadPathList);
24    if (!isSymbol(aLoadPath)) continue;
25
26    JObj* aLoadExtensionList = loader->extensions;
27    while(aLoadExtensionList) {
28      assert(isPair(aLoadExtensionList));
29      JObj* aLoadExtension = asCar(aLoadExtensionList);
30      if (!isSymbol(aLoadExtension)) continue;
31
32      char buffer[8000];
33      buffer[0] = 0;
34      assert(isSymbol(aLoadPath));
35      strcat(buffer, asSymbol(aLoadPath));
36      strcat(buffer, "/");
37      strcat(buffer, aFileToLoad);
38      assert(isSymbol(aLoadExtension));
39      strcat(buffer, asSymbol(aLoadExtension));
40      DEBUG(jInterp, "trying to load: [%s]\n", buffer);
41
42      if (!access(buffer, R_OK)) {
43        if (strcmp(asSymbol(aLoadExtension), ".joy") == 0) {
44          if (loadAJoyLoLFile(aCtx, buffer)) {
45            DEBUG(jInterp, "loaded: [%s]\n", buffer);
46            return TRUE;
47          }
48        } else {
49          assert(aCtx);
50          lua_State *lstate = aCtx->jInterp->lstate;
51          requireLuaModuleInto(lstate, aFileToLoad, loadedOk);
52          if (loadedOk) {
53            DEBUG(jInterp, "loaded: [%s]\n", buffer);
54            return TRUE;
55          }
56        }
57        // keep trying other paths/extensions ...
58      }
59      // try the next ext
60      aLoadExtensionList = asCdr(aLoadExtensionList);
61    }
62    // try the next load path
63    aLoadPathList = asCdr(aLoadPathList);
64  }
```

```
65    if(aCtx->verbose) {
66      jInterp->writeStdOut(jInterp, "FAILED\n");
67    }
68    DEBUG(jInterp, "loadAFile: FAILED%s\n", "");
69    return FALSE; // we could not find this file in any of the load paths
70  }
```

```
Boolean loadFiles(
  ContextObj *aCtx,
  JObj   *filesToLoad
) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  Loader* loader = jInterp->loader;
  assert(loader);
  if (!filesToLoad) return TRUE; // nothing to do...

  DEBUG(jInterp, "loadFiles > %s:%p\n", filesToLoad->coAlg->name, filesToLoad);

  if (isSymbol(filesToLoad)) { // try to load this file
    return loadAFile(aCtx, filesToLoad->symbol);
  }

  // this is a Pair in the list of load files...
  // ... walk down the list
  if (loadFiles(aCtx, filesToLoad->pair.cdr)) {
    // if all previous files have been loaded... load this one as well
    return loadFiles(aCtx, filesToLoad->pair.car);
  }
  DEBUG(jInterp, "loadFiles < %p FAILED\n", filesToLoad);
  return FALSE; // some previous file has failed to load...
}
```

3.12.2.1 Test Suite: regiserLoaders

CHeader : public

```
1  typedef struct loaders_class_struct {
2    JClass      super;
3    NewLoader       *newLoaderFunc;
4    PushLoadSymbol *pushLoadPathFunc;
5    ListLoader      *listLoadPathsFunc;
```

```
  PushLoadSymbol *pushLoadExtensionFunc;
  ListLoader     *listLoadExtensionsFunc;
  LoadAFile      *loadAFileFunc;
} LoadersClass;
```

CCode : lua

```
static Boolean initializeLoaders(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  if (!jInterp->loader) {
    jInterp->loader = newLoader(jInterp);
  }
  return TRUE;
}
```

CHeader : private

```
Boolean registerLoaders(
  JoyLoLInterp *jInterp
);
```

CCode : lua

```
Boolean registerLoaders(
  JoyLoLInterp *jInterp
) {
  assert(jInterp);

  LoadersClass* theCoAlg =
    joyLoLCalloc(1, LoadersClass);
  assert(theCoAlg);

  theCoAlg->super.name             = LoadersName;
  theCoAlg->super.objectSize       = sizeof(LoaderObj);
  theCoAlg->super.initializeFunc   = initializeLoaders;
  theCoAlg->super.registerFunc     = registerLoaderWords;
  theCoAlg->super.equalityFunc     = NULL;
  theCoAlg->super.printFunc        = NULL;
  theCoAlg->newLoaderFunc          = newLoaderImpl;
  theCoAlg->pushLoadPathFunc       = pushLoadPathImpl;
  theCoAlg->listLoadPathsFunc      = listLoadPathsImpl;
```

```
17    theCoAlg->pushLoadExtensionFunc  = pushLoadExtensionImpl;
18    theCoAlg->listLoadExtensionsFunc = listLoadExtensionsImpl;
19    theCoAlg->loadAFileFunc          = loadAFileImpl;
20
21    size_t tag =
22      registerJClass(jInterp, (JClass*)theCoAlg);
23    // do a sanity check...
24    assert(tag == LoadersTag);
25    assert(tag < jInterp->numCoAlgs);
26    assert(getLoadersClass(jInterp));
27
28    return TRUE;
29  }
```

Test case

should register the Loaders coAlg

```
// CTestsSetup has already created a jInterp
// and run registerJInterps
//
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertIntTrue(LoadersTag < jInterp->numCoAlgs);
AssertPtrNotNull(getLoadersClass(jInterp));

LoadersClass *loadersClass = getLoadersClass(jInterp);
size_t result = registerLoaders(jInterp);
AssertIntTrue(result);
AssertPtrNotNull(getLoadersClass(jInterp));
AssertPtrEquals(getLoadersClass(jInterp), loadersClass);
AssertIntEquals(
  getLoadersClass(jInterp)->super.objectSize,
  sizeof(LoaderObj)
)
```

3.12.3 Words

CHeader : private

```
1  extern Boolean registerLoaderWords(
```

```
  JoyLoLInterp *jInterp,
  JClass       *theCoALg
);
```

CCode : lua

```
Boolean registerLoaderWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  return TRUE;
}
```

3.12.4 Lua functions

CCode : lua

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : lua

```
static int lua_loaders_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_loaders [] = {
  {"gitVersion", lua_loaders_getGitVersion},
  {NULL, NULL}
};
```

```

int luaopen_joylol_loaders (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerLoaders(jInterp);
  luaL_newlib(lstate, lua_loaders);
  return 1;
}

```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedLoaders` does just this.

CHeader : public

```
extern Boolean requireStaticallyLinkedLoaders(
  lua_State *lstate
);
```

CCode : lua

```
Boolean requireStaticallyLinkedLoaders(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_loaders(lstate);
  lua_setfield(lstate, -2, "joylol.loaders");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.12.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_loaders(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
```

```
3  #include <unistd.h>
4  #include <assert.h>
5  #include <joylol/jInterps.h>
6  #include <joylol/stringBuffers.h>
7  #include <joylol/symbols.h>
8  #include <joylol/cFunctions.h>
9  #include <joylol/pairs.h>
10 #include <joylol/texts.h>
11 #include <joylol/assertions.h>
12 #include <joylol/contexts.h>
13 #include <joylol/loaders.h>
14 #include <joylol/loaders-private.h>
15 // dictionary
16 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedLoaders(lstate);
getJoyLoLInterpInto(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.13 Lua-Functions

3.13.1 Goals

A Lua-Function represents a JoyLoL word implemented as an Lua function which directly manipulates a Context.

34

3.13.2 Code

```
\startCHeader

typedef void (CFunction)(Context*);

typedef struct luaFunction_struct {
  CoAlgebras super;
  CFunction  theFunction;
} CFunction;

\stopCHeader

\startJoyLoLWord[isCFunction]

\preProcessStack[aCFunc][]
\postDataStack[isBoolean]

\startCCode

if (!aCFunc) then
  pushDataFalse(aCtx);
if (aCFunc->isA == CFUNCTIONS_COALG) then
  pushDataTrue(aCtx);
pushDataFalse(aCtx);

\stopCCode

\stopJoyLoLWord

\startJoyLoLWord[executeCFunc]

\preProcessStack[aCFunc][]
```

Code 3.13.2

```
\startCCode

if (!aCFunc) then
  raiseExceptionMsg(aCtx, "no atom (CFunction)");
if (aCFunc->super.isA != CFUNCTIONS_COALG) then
  raiseExceptionMsg(aCtx, "not a CFunction");
if (!aCFunc->theFunction) then
  raiseExceptionMsg(aCtx, "no CFunction found");

(aCFunc->theFunction)(aCtx);

\stopCCode

\stopJoyLoLWord
```

```
#ifndef JOYLOL_COALG_FUNCTIONS_H
#define JOYLOL_COALG_FUNCTIONS_H

typedef struct context_struct Context;
typedef void (JoyLoLFunction)(Context*);

typedef struct functions_struct {
  CoAlgebra super;
  // other things
} Functions;

extern Functions* createFunctionsCoAlgebra(void);
extern void initFunctionsCoAlgebra(CoAlgebras* coAlgs);

extern PairAtom* newJoyLoLFunc(CoAlgebras* coAlgs, JoyLoLFunction *aJoyLoLFunc);

extern size_t isFunction(PairAtom* aLoL);

#endif
```

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "joyLoL/macros.h"
#include "joyLoL/coAlg/coAlgs.h"
#include "joyLoL/lists.h"
#include "joyLoL/printer.h"
```

```
static size_t equalityFuncCoAlg(CoAlgebra* klass,
                                PairAtom* lolA, PairAtom* lolB,
                                size_t debugFlag) {
  DEBUG(debugFlag, "funcCoAlg-equal klass:%p a:%p b:%p\n", klass, lolA,
lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (lolA->coAlg != klass) return FALSE;
  if (lolB->coAlg != klass) return FALSE;
  if (lolA->func != lolB->func) return FALSE;
  return TRUE;
}

static size_t printSizeFuncCoAlg(PairAtom* aLoL, size_t debugFlag) {
  DEBUG(debugFlag, "funcCoAlg-printSize: %p\n", aLoL);
  assert(aLoL);
  assert(aLoL->coAlg);
  assert(aLoL->coAlg->isA == FUNCTION_COALG);
  DEBUG(debugFlag, "funcCoAlg-printSize: func<%p> %p\n", aLoL->func, aLoL);
  return 15;
}

static size_t printStrFuncCoAlg(PairAtom* aLoL,
                                char* buffer, size_t bufferSize) {
  assert(aLoL);
  assert(aLoL->coAlg);
  assert(aLoL->coAlg->isA == FUNCTION_COALG);

  char ptoa[100];
  sprintf(ptoa, "<%p> ", aLoL->func);
  strcat(buffer, ptoa);
  return TRUE;
}

Functions* createFunctionsCoAlgebra(void) {
  Functions* funcs = (Functions*) calloc(1, sizeof(Functions));
  initACoAlgebra((CoAlgebra*)funcs);

  funcs->super.isA       = FUNCTION_COALG;
  funcs->super.name      = LuaFunctionsName;
  funcs->super.equality  = equalityFuncCoAlg;
  funcs->super.printSize = printSizeFuncCoAlg;
```

```
  funcs->super.printStr  = printStrFuncCoAlg;
  return funcs;
}

PairAtom* newJoyLoLFunc(CoAlgebras* coAlgs, JoyLoLFunction *aFunc) {
  assert(coAlgs);
  PairAtom* aNewFunc = newPairAtom(coAlgs);
  assert(aNewFunc);
  aNewFunc->coAlg = (CoAlgebra*)coAlgs->functions;
  aNewFunc->tag   = 0;
  aNewFunc->func  = aFunc;
  return aNewFunc;
}

size_t isFunction(PairAtom* aLoL) {
  if (!aLoL) return FALSE;
  if (aLoL->coAlg && (aLoL->coAlg->isA == FUNCTION_COALG)) return TRUE;
  return FALSE;
}

static void isFunctionAP(Context* aCtx) {
  assert(aCtx);
  assert(aCtx->coAlgebras);
  popCtxDataInto(aCtx, top);
  PairAtom* result = NULL;
  if (isFunction(top)) result = newBoolean(aCtx->coAlgebras, TRUE);
  else                 result = newBoolean(aCtx->coAlgebras, FALSE);
  pushCtxData(aCtx, result);
}

void initFunctionsCoAlgebra(CoAlgebras* coAlgs) {
  extendJoyLoL(coAlgs, "isFunction", isFunctionAP);
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "CuTest.h"

#include "joyLoL/macros.h"
#include "joyLoL/coAlg/coAlgs.h"
#include "joyLoL/dictionary.h"
```

```
#include "joyLoL/lists.h"
#include "joyLoL/printer.h"

// suiteName: - Functions CoAlgebra tests -

void Test_createFunctionsCoAlgebra(CuTest* tc) {
  CoAlgebras *coAlgs = createCoAlgebras();
  CuAssertPtrNotNull(tc, coAlgs);
  CuAssertPtrNotNull(tc, coAlgs->functions);
}

void testJoyLoLFunction(Context* aCtx) { }

void Test_newJoyLoLFunction(CuTest* tc) {
  CoAlgebras* coAlgs = createCoAlgebras();
  CuAssertPtrNotNull(tc, coAlgs);
  CuAssertPtrNotNull(tc, coAlgs->functions);

  PairAtom* aNewFunc = newJoyLoLFunc(coAlgs, testJoyLoLFunction);
  CuAssertPtrNotNull(tc, aNewFunc);
  CuAssertPtrNotEquals(tc, aNewFunc, testJoyLoLFunction);
  CuAssertPtrNotNull(tc, aNewFunc->coAlg);
  CuAssertIntEquals(tc, aNewFunc->coAlg->isA, FUNCTION_COALG);
  CuAssertPtrEquals(tc, aNewFunc->func, testJoyLoLFunction);
  CuAssertTrue(tc, isFunction(aNewFunc));
  CuAssertTrue(tc, isAtom(aNewFunc));
  CuAssertFalse(tc, isBoolean(aNewFunc));
  CuAssertFalse(tc, isContext(aNewFunc));
  CuAssertFalse(tc, isNatural(aNewFunc));
  CuAssertFalse(tc, isPair(aNewFunc));
  CuAssertFalse(tc, isSymbol(aNewFunc));
}

void Test_printJoyLoLFuncion(CuTest* tc) {
  CoAlgebras* coAlgs = createCoAlgebras();
  CuAssertPtrNotNull(tc, coAlgs);
  CuAssertPtrNotNull(tc, coAlgs->functions);

  char buffer[100];
  buffer[0] = 0;
  sprintf((char*)&buffer, "<%p>", testJoyLoLFunction);

  PairAtom* aNewFunc = newJoyLoLFunc(coAlgs, testJoyLoLFunction);
```

```
  CuAssertPtrNotNull(tc, aNewFunc);
  CuAssertIntEquals(tc, printSizeDebug(aNewFunc, FALSE), 15);
  CuAssertStrEquals(tc, printLoLDebug(aNewFunc, FALSE), buffer);
}
```

3.13.2.1 Test Suite: registerLuaFunctions

CHeader : public

```
typedef struct luaFunctions_class_struct {
  JClass super;
} LuaFunctionsClass;
```

CCode : default

```
static Boolean initializeLuaFunctions(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerLuaFunctions(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerLuaFunctions(JoyLoLInterp *jInterp) {
  assert(jInterp);

  LuaFunctionsClass* theCoAlg =
    joyLoLCalloc(1, LuaFunctionsClass);
  theCoAlg->super.name           = LuaFunctionsName;
  theCoAlg->super.objectSize     = sizeof(JObj);
  theCoAlg->super.initializeFunc = initializeLuaFunctions;
  theCoAlg->super.registerFunc   = registerLuaFunctionWords;
  theCoAlg->super.equalityFunc   = NULL;
  theCoAlg->super.printFunc      = NULL;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

```

```
  // do a sanity check...
  assert(tag == LuaFunctionsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the LuaFunctions coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerLuaFunctions

  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getLuaFunctionsClass(jInterp));
  LuaFunctionsClass *coAlg =
    getLuaFunctionsClass(jInterp);
  AssertIntTrue(registerLuaFunctions(jInterp));
  AssertPtrNotNull(getLuaFunctionsClass(jInterp));
  AssertPtrEquals(getLuaFunctionsClass(jInterp), coAlg);
  AssertIntEquals(
    getLuaFunctionsClass(jInterp)->super.objectSize,
    sizeof(JObj)
  )
```

3.13.3 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_luaFunctions_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_luaFunctions [] = {
  {"gitVersion", lua_luaFunctions_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_luaFunctions (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerLuaFunctions(jInterp);
  luaL_newlib(lstate, lua_luaFunctions);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedLuaFunctions` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedLuaFunctions(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedLuaFunctions(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_luaFunctions(lstate);
  lua_setfield(lstate, -2, "joylol.luaFunctions");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
```

```
10    return TRUE;
11  }
```

3.13.4 JoyLoL words

CHeader : private

```
1  extern Boolean registerLuaFunctionWords(
2    JoyLoLInterp *jInterp,
3    JClass       *theCoAlg
4  );
```

CCode : default

```
1  Boolean registerLuaFunctionWords(
2    JoyLoLInterp *jInterp,
3    JClass       *theCoAlg
4  ) {
5    return TRUE;
6  }
```

3.13.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/luaFunctions.h>
6  #include <joylol/luaFunctions-private.h>
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireStaticallyLinkedLuaFunctions(lstate);
  getJoyLoLInterpInto(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.14 Naturals

3.14.1 Goals

The symbols extension extends the core JoyLoL by providing support for constant strings, aka 'symbols'.

3.14.2 Code

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```c
#include <gmp.h>
typedef mpz_t Natural;

typedef struct natural_object_struct {
  JObj super;
  Natural  natural;
} NaturalObj;

#define asNatural(aObj) (((NaturalObj*)(aObj))->natural)
```

3.14.2.1 Test Suite: newNatural

CHeader : public

```c
typedef JObj *(NewNatural)(
  JoyLoLInterp *jInterp,
  Symbol       *aNatural
);

#define newNatural(jInterp, aNatural)       \
```

```
  (                                          \
    assert(getNaturalsClass(jInterp)         \
      ->newNaturalFunc),                     \
    (getNaturalsClass(jInterp)               \
      ->newNaturalFunc(jInterp, aNatural))   \
  )
```

CHeader : private

```
extern JObj* newNaturalImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aNatural
);
```

CCode : default

```
JObj* newNaturalImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aNatural
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj* result = newObject(jInterp, NaturalsTag);
  assert(result);
  if (aNatural) {
    long success = mpz_set_str(asNatural(result), aNatural, 0);
    assert(success == 0);
  } else mpz_init(asNatural(result));
  return result;
}
```

Test case

should create some new naturals

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);

  const char* testNum = "12345";
  JObj* aNewNat = newNatural(jInterp, testNum);
  AssertPtrNotNull(aNewNat);
  AssertPtrNotNull(asType(aNewNat));
  AssertIntEquals(asTag(aNewNat), NaturalsTag);
  AssertIntTrue(isAtom(aNewNat));
```

```
  AssertIntFalse(isPair(aNewNat));
```

3.14.2.2 Test Suite: isNatural

CHeader : public

```
#define isNatural(aLoL)              \
  (                                  \
    (                                \
      (aLoL) &&                      \
      asType(aLoL) &&                \
      (asTag(aLoL) == NaturalsTag)   \
    ) ?                              \
      TRUE :                         \
      FALSE                          \
  )
```

Test case

should return true if a natural

```
  JObj *aNat = newNatural(jInterp, "12345");
  AssertIntTrue(isNatural(aNat));
```

Test case

should return false if not a symbol

```
  AssertIntFalse(isNatural(NULL));
  JObj *aObj = newObject(jInterp, BooleansTag);
  AssertIntFalse(isNatural(aObj));
```

3.14.2.3 Test Suite: asNaturalDbl

CHeader : public

```
typedef double (AsNaturalDbl)(
  JObj* aNatural
);

#define asNaturalDbl(jInterp, aNatural) \
  (                                     \
```

```
    assert(getNaturalsClass(jInterp)     \
      ->asNaturalDblFunc),               \
    (getNaturalsClass(jInterp)           \
      ->asNaturalDblFunc(aNatural))       \
  )
```

CHeader : private

```
extern double asNaturalDblImpl(
  JObj* aNatural
);
```

CCode : default

```
double asNaturalDblImpl(
  JObj* aNatural
) {
  double result = 0.0;
  if (isNatural(aNatural)) {
    result = mpz_get_d(asNatural(aNatural));
  }
  return result;
}
```

Test case

should convert naturals to doubles

```
  JObj *aNat = newNatural(jInterp, "12345");
  AssertDblEquals(asNaturalDbl(jInterp, aNat), 12345.0, 0.0001);
```

3.14.2.4 Test Suite: natural equality

CHeader : private

```
Boolean equalityNatCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityNatCoAlg(
```

```
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "natCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA) != NaturalsTag) return FALSE;
  if (mpz_cmp(asNatural(lolA), asNatural(lolB)) != 0) return FALSE;
  return TRUE;
}
```

Test case

should return true if naturals are equal

```
  AssertIntTrue(equalityNatCoAlg(jInterp, NULL, NULL, 10));
  JObj *natA = newNatural(jInterp, "12345");
  JObj *natB = newNatural(jInterp, "12345");
  AssertIntTrue(equalityNatCoAlg(jInterp, natA, natB, 10));
```

Test case

should return false if symbols are not equal

```
  JObj *natA = newNatural(jInterp, "12346");
  JObj *natB = newNatural(jInterp, "12345");
  AssertIntFalse(equalityNatCoAlg(jInterp, NULL, natB, 10));
  AssertIntFalse(equalityNatCoAlg(jInterp, natA, NULL, 10));
  AssertIntFalse(equalityNatCoAlg(jInterp, natA, natB, 10));
```

3.14.2.5 Test Suite: printing symbols

CHeader : private

```
extern Boolean printNatCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
```

```
  size_t           timeToLive
);
```

CCode : default

```
Boolean printNatCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == NaturalsTag);

  char* mpztoa;
  mpztoa = NULL;
  long numChars = gmp_asprintf(&mpztoa, "%Zd ", asNatural(aLoL));
  assert(0 < numChars);
  strBufPrintf(aStrBuf, mpztoa);
  free(mpztoa);
  return TRUE;
}
```

Test case

should print symbols

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[NaturalsTag]);

  const char* testNum = "12345 ";
  JObj* aNewNat = newNatural(jInterp, testNum);
  AssertPtrNotNull(aNewNat);
  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  printNatCoAlg(aStrBuf, aNewNat, 10);
  AssertStrEquals(getCString(aStrBuf), testNum);
  strBufClose(aStrBuf);
```

3.14.2.6 Test Suite: registerSymbols

CHeader : public

```
typedef struct naturals_class_struct {
  JClass super;
```

```
  NewNatural   *newNaturalFunc;
  AsNaturalDbl *asNaturalDblFunc;
} NaturalsClass;
```

CCode : default

```
static Boolean initializeNaturals(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerNaturals(JoyLoLInterp* jInterp);
```

CCode : default

```
Boolean registerNaturals(JoyLoLInterp* jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  NaturalsClass* theCoAlg        = joyLoLCalloc(1, NaturalsClass);
  theCoAlg->super.name           = NaturalsName;
  theCoAlg->super.objectSize     = sizeof(NaturalObj);
  theCoAlg->super.initializeFunc = initializeNaturals;
  theCoAlg->super.registerFunc   = registerNaturalWords;
  theCoAlg->super.equalityFunc   = equalityNatCoAlg;
  theCoAlg->super.printFunc      = printNatCoAlg;
  theCoAlg->newNaturalFunc       = newNaturalImpl;
  theCoAlg->asNaturalDblFunc     = asNaturalDblImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // sanity check...
  assert(tag == NaturalsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Naturals coAlg

```
// CTestsSetup has already created a jInterp
// and run registerSymbols
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getNaturalsClass(jInterp));
NaturalsClass *coAlg = getNaturalsClass(jInterp);
AssertIntTrue(registerNaturals(jInterp));
AssertPtrNotNull(getNaturalsClass(jInterp));
AssertPtrEquals(getNaturalsClass(jInterp), coAlg);
AssertIntEquals(
  getNaturalsClass(jInterp)->super.objectSize,
  sizeof(NaturalObj)
)
```

3.14.3 Words

CCode : default

```
static  mpz_t zero;
```

CCode : default

```
static void addAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  if (!isNatural(top1) || !isNatural(top2)) {
    raiseExceptionMsg(aCtx,
      "addition requires that the top two stack elements are naturals"
    );
    return;
  }
  JObj* sum = newNatural(jInterp, NULL);
  mpz_add(asNatural(sum), asNatural(top1), asNatural(top2));
  pushCtxData(aCtx, sum);
}
```

```
\startWord[add]

\preDataStack
  (
    top1 : natural
    top2 : natural
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreStack

\postDataStack
  (
    top1 + top2 : natural ??
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void subtractAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  if (!isNatural(top1) || !isNatural(top2)) {
    raiseExceptionMsg(aCtx,
      "subtraction requires that the top two stack elements are naturals"
    );
    return;
  }
  JObj* difference = newNatural(jInterp, NULL);
  if (mpz_cmp(asNatural(top2), asNatural(top1)) < 0) {
    mpz_sub(asNatural(difference), asNatural(top1), asNatural(top2));
  }
  pushCtxData(aCtx, difference);
```

```
18  }
```

```
\startWord[subtract]

\preDataStack
  (
    top1 : natural
    top2 : natural
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreStack

\postDataStack
  (top2 < top1) ->(
    top1 - top2 : natural ??
    dataStack
  )
  OR
  (else) -> (
    0 : natural
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
1   static void subtractReverseAP(ContextObj* aCtx) {
2     assert(aCtx);
3     JoyLoLInterp *jInterp = aCtx->jInterp;
4     assert(jInterp);
5     popCtxDataInto(aCtx, top1);
6     popCtxDataInto(aCtx, top2);
7     if (!isNatural(top1) || !isNatural(top2)) {
8       raiseExceptionMsg(aCtx,
9         "(reverse) subtraction requires that the top two stack elements are naturals"
10      );
```

```
    return;
  }
  JObj* difference = newNatural(jInterp, NULL);
  if (mpz_cmp(asNatural(top1), asNatural(top2)) < 0) {
    mpz_sub(asNatural(difference), asNatural(top2), asNatural(top1));
  }
  pushCtxData(aCtx, difference);
}
```

```
\startWord[subtractReverse]

\preDataStack
  (
    top1 : natural
    top2 : natural
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreStack

\postDataStack
  ( top1 < top2 ) -> (
    top2 - top1 : natural ?
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void multiplyAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  if (!isNatural(top1) || !isNatural(top2)) {
    raiseExceptionMsg(aCtx,
```

```
      "multiplication requires that the top two stack elements are naturals"
    );
    return;
  }
  JObj* product = newNatural(jInterp, NULL);
  mpz_mul(asNatural(product), asNatural(top1), asNatural(top2));
  pushCtxData(aCtx, product);
}
```

```
\startWord[mulitply]

\preDataStack
  (
    top1 : natural
    top2 : natural
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreStack

\postDataStack
  (
    top1 * top2 : natural ?
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostStack

\stopWord
```

CCode : default

```
static void quotientAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  if (!isNatural(top1) || !isNatural(top2)) {
    raiseExceptionMsg(aCtx,
```

```
9         "integer quotient requires that the top two stack elements are naturals"
10      );
11      return;
12    }
13    if (mpz_cmp(asNatural(top2), zero) == 0) {
14      raiseExceptionMsg(aCtx,
15        "integer quotient requires that the denominator is not zero"
16      );
17      return;
18    }
19    JObj* product = newNatural(jInterp, NULL);
20    mpz_tdiv_q(asNatural(product), asNatural(top1), asNatural(top2));
21    pushCtxData(aCtx, product);
22  }
23
24  static void quotientReverseAP(ContextObj* aCtx) {
25    assert(aCtx);
26    JoyLoLInterp *jInterp = aCtx->jInterp;
27    assert(jInterp);
28    popCtxDataInto(aCtx, top1);
29    popCtxDataInto(aCtx, top2);
30    if (!isNatural(top1) || !isNatural(top2)) {
31      raiseExceptionMsg(aCtx,
32        "integer quotient requires that the top two stack elements are naturals"
33      );
34      return;
35    }
36    if (mpz_cmp(asNatural(top1), zero) == 0) {
37      raiseExceptionMsg(aCtx,
38        "integer quotient requires that the denominator is not zero"
39      );
40      return;
41    }
42    JObj* product = newNatural(jInterp, NULL);
43    mpz_tdiv_q(asNatural(product), asNatural(top2), asNatural(top1));
44    pushCtxData(aCtx, product);
45  }
46
47  static void remainderAP(ContextObj* aCtx) {
48    assert(aCtx);
49    JoyLoLInterp *jInterp = aCtx->jInterp;
50    assert(jInterp);
51    popCtxDataInto(aCtx, top1);
```

```
52    popCtxDataInto(aCtx, top2);
53    if (!isNatural(top1) || !isNatural(top2)) {
54      raiseExceptionMsg(aCtx,
55        "integer remainder requires that the top two stack elements are naturals"
56      );
57      return;
58    }
59    if (mpz_cmp(asNatural(top2), zero) == 0) {
60      raiseExceptionMsg(aCtx,
61        "integer remainder requires that the denominator is not zero"
62      );
63      return;
64    }
65    JObj* product = newNatural(jInterp, NULL);
66    mpz_tdiv_r(asNatural(product), asNatural(top1), asNatural(top2));
67    pushCtxData(aCtx, product);
68  }
69
70  static void remainderReverseAP(ContextObj* aCtx) {
71    assert(aCtx);
72    JoyLoLInterp *jInterp = aCtx->jInterp;
73    assert(jInterp);
74    popCtxDataInto(aCtx, top1);
75    popCtxDataInto(aCtx, top2);
76    if (!isNatural(top1) || !isNatural(top2)) {
77      raiseExceptionMsg(aCtx,
78        "integer remainder requires that the top two stack elements are naturals"
79      );
80      return;
81    }
82    if (mpz_cmp(asNatural(top1), zero) == 0) {
83      raiseExceptionMsg(aCtx,
84        "integer remainder requires that the denominator is not zero"
85      );
86      return;
87    }
88    JObj* product = newNatural(jInterp, NULL);
89    mpz_tdiv_r(asNatural(product), asNatural(top2), asNatural(top1));
90    pushCtxData(aCtx, product);
91  }
```

CCode : default

```
1   static void lessThanNatAP(ContextObj* aCtx) {
```

```
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  Boolean result = FALSE;
  if (isNatural(top1) &&
      isNatural(top2) &&
      (mpz_cmp(asNatural(top1), asNatural(top2)) < 0)) result = TRUE;
  DEBUG(jInterp, "equalNat: %zu\n", result);
  pushCtxData(aCtx, top2);
  pushCtxData(aCtx, top1);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

CCode : default

```
static void lessThanEqualNatAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  Boolean result = FALSE;
  if (isNatural(top1) &&
      isNatural(top2) &&
      (mpz_cmp(asNatural(top1), asNatural(top2)) <= 0)) result = TRUE;
  DEBUG(jInterp, "equalNat: %zu\n", result);
  pushCtxData(aCtx, top2);
  pushCtxData(aCtx, top1);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

CCode : default

```
static void greaterThanNatAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  Boolean result = FALSE;
  if (isNatural(top1) &&
      isNatural(top2) &&
```

```
      (mpz_cmp(asNatural(top1), asNatural(top2)) > 0)) result = TRUE;
  DEBUG(jInterp, "equalNat: %zu\n", result);
  pushCtxData(aCtx, top2);
  pushCtxData(aCtx, top1);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

CCode : default

```
static void greaterThanEqualNatAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  Boolean result = FALSE;
  if (isNatural(top1) &&
      isNatural(top2) &&
      (mpz_cmp(asNatural(top1), asNatural(top2)) >= 0)) result = TRUE;
  DEBUG(jInterp, "equalNat: %zu\n", result);
  pushCtxData(aCtx, top2);
  pushCtxData(aCtx, top1);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

CCode : default

```
static void equalNatAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top1);
  popCtxDataInto(aCtx, top2);
  Boolean result = FALSE;
  if (isNatural(top1) &&
      isNatural(top2) &&
      (mpz_cmp(asNatural(top1), asNatural(top2)) == 0)) result = TRUE;
  DEBUG(jInterp, "equalNat: %zu\n", result);
  pushCtxData(aCtx, top2);
  pushCtxData(aCtx, top1);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

```
\startWord[equal]
```

```
\preDataStack
  (
    top1 : natural
    top2 : natural
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreConditions

\postDataStack
  (
    top1 == top2 : boolean
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostConditions

\stopWord
```

CCode : default

```
static void isZeroAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  size_t result = FALSE;
  if (isNatural(top) &&
      (mpz_cmp(asNatural(top), zero) == 0)) result = TRUE;
  pushCtxData(aCtx, top);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

```
\startWord[isZero]

\preDataStack
  (
    top : natural
    dataStack
  )
```

```
\preProcessStack
  ( processStack )
\preConditions
\stopPreConditions

\postDataStack
  (
    top == 0 : Boolean
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostConditions

\stopWord
```

CCode : default

```
1  static void isNaturalAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isNatural(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, top);
12   pushCtxData(aCtx, result);
13 }
```

```
\startWord[isNatural]

\preDataStack
  (
    top : aType
    dataStack
  )
\preProcessStack
  ( processStack )
\preConditions
\stopPreConditions
```

```
\postDataStack
  (
    (top isNatural) : Boolean ????
    dataStack
  )
\postProcessStack
  ( processStack )
\postConditions
\stopPostConditions

\stopWord
```

CHeader : private

```
extern Boolean registerNaturalWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerNaturalWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  mpz_init(zero);
  extendJoyLoLInRoot(jInterp, "isNatural", "", isNaturalAP,           "");
  extendJoyLoLInRoot(jInterp, "+",         "", addAP,                 "");
  extendJoyLoLInRoot(jInterp, "-",         "", subtractAP,            "");
  extendJoyLoLInRoot(jInterp, "-rev",      "", subtractReverseAP,     "");
  extendJoyLoLInRoot(jInterp, "*",         "", multiplyAP,            "");
  extendJoyLoLInRoot(jInterp, "/",         "", quotientAP,            "");
  extendJoyLoLInRoot(jInterp, "/rev",      "", quotientReverseAP,     "");
  extendJoyLoLInRoot(jInterp, "%",         "", remainderAP,           "");
  extendJoyLoLInRoot(jInterp, "%rev",      "", remainderReverseAP,    "");
  extendJoyLoLInRoot(jInterp, "<nat",      "", lessThanNatAP,         "");
  extendJoyLoLInRoot(jInterp, "<=nat",     "", lessThanEqualNatAP,    "");
  extendJoyLoLInRoot(jInterp, ">nat",      "", greaterThanNatAP,      "");
  extendJoyLoLInRoot(jInterp, ">=nat",     "", greaterThanEqualNatAP,
"");
  extendJoyLoLInRoot(jInterp, "=nat",      "", equalNatAP,            "");
  extendJoyLoLInRoot(jInterp, "isZero",    "", isZeroAP,              "");

  return TRUE;
```

```
23 }
```

3.14.4 Lua functions

CCode : default

```
1  static int lua_naturals_getGitVersion (lua_State *lstate) {
2    const char* aKey   = lua_tostring(lstate, 1);
3    if (aKey) {
4      getGitVersionInto(gitVersionKeyValues, aKey, aValue);
5      lua_pushstring(lstate, aValue);
6    } else {
7      lua_pushstring(lstate, "no valid key provided");
8    }
9    return 1;
10 }
11
12 static const struct luaL_Reg lua_naturals [] = {
13   {"gitVersion", lua_naturals_getGitVersion},
14   {NULL, NULL}
15 };
16
17 int luaopen_joylol_naturals (lua_State *lstate) {
18   getJoyLoLInterpInto(lstate, jInterp);
19   registerNaturals(jInterp);
20   luaL_newlib(lstate, lua_naturals);
21   return 1;
22 }
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedNaturals` does just this.

CHeader : public

```
1 Boolean requireStaticallyLinkedNaturals(
2   lua_State *lstate
3 );
```

CCode : default

```
1 Boolean requireStaticallyLinkedNaturals(
2   lua_State *lstate
3 ) {
```

```
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_naturals(lstate);
  lua_setfield(lstate, -2, "joylol.naturals");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.14.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/booleans.h>
#include <joylol/stringBuffers.h>
#include <joylol/cFunctions.h>
#include <joylol/naturals.h>
#include <joylol/texts.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/naturals-private.h>
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
  requireLuaModule(lstate, "joylol.pairs");
  requireLuaModule(lstate, "joylol.cFunctions");
  requireLuaModule(lstate, "joylol.texts");
  requireLuaModule(lstate, "joylol.contexts");
  requireLuaModule(lstate, "joylol.dictionaries");
  requireLuaModule(lstate, "joylol.dictNodes");
  requireLuaModule(lstate, "joylol.stringBuffers");
  requireLuaModule(lstate, "joylol.booleans");
  requireStaticallyLinkedNaturals(lstate);
```

```
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.15 Pairs

3.15.1 Goals

A pair has two pointers, a `car` and a `cdr`.

3.15.2 Code

CHeader : public

```
typedef struct pairs_object_struct {
  JObj  super;
  JObj* car;
  JObj* cdr;
} PairsObj;
```

3.15.2.1 Test Suite: newPair

CHeader : public

```
typedef JObj* (NewPair)(
  JoyLoLInterp* jInterp,
  JObj* car,
  JObj* cdr
);

#define newPair(jInterp, aCar, aCdr)      \
  (                                       \
    assert(getPairsClass(jInterp)         \
      ->newPairFunc),                     \
    (getPairsClass(jInterp)               \
      ->newPairFunc(jInterp, aCar, aCdr)) \
  )
#define asCar(aLoL) (((PairsObj*)(aLoL))->car)
#define asCdr(aLoL) (((PairsObj*)(aLoL))->cdr)
```

CHeader : private

```
extern JObj* newPairImpl(
  JoyLoLInterp* jInterp,
  JObj* car,
  JObj* cdr
```

```
);
```

CCode : default

```
JObj* newPairImpl(
  JoyLoLInterp* jInterp,
  JObj* car,
  JObj* cdr
) {
  assert(jInterp);
  JObj* result = newObject(jInterp, PairsTag);
  assert(result);
  asCar(result) = car;
  asCdr(result) = cdr;
  DEBUG(jInterp, "newPair %p %p %p\n",
    result, car, cdr);
  return result;
}
```

Test case

should create a new pair

```
  AssertPtrNotNull(jInterp);

  JObj *lolA = newPair(jInterp, NULL, NULL);
  JObj *lolB = newPair(jInterp, NULL, NULL);
  JObj *lol  = newPair(jInterp, lolA, lolB);
  AssertPtrEquals(asType(lol), jInterp->coAlgs[PairsTag]);
  AssertIntEquals(asTag(lol), PairsTag);
  AssertIntZero(asFlags(lol));
  AssertPtrEquals(asCar(lol), lolA);
  AssertPtrEquals(asCdr(lol), lolB);
```

3.15.2.2 Test Suite: car and cdr

CHeader : public

```
typedef JObj *(CarCdr)(JObj*);

#define getCar(jInterp, aLoL)                \
  (                                          \
    assert(getPairsClass(jInterp)->carFunc),\
```

```
    (getPairsClass(jInterp)->carFunc(aLoL)) \
  )
#define getCdr(jInterp, aLoL)                \
  (                                          \
    assert(getPairsClass(jInterp)->cdrFunc),\
    (getPairsClass(jInterp)->cdrFunc(aLoL)) \
  )
```

CHeader : private

```
extern JObj* carImpl(JObj* aLoL);
extern JObj* cdrImpl(JObj* aLoL);
```

CCode : default

```
JObj* carImpl(JObj* aLoL) {
  if (!aLoL) return NULL;
  if (asTag(aLoL) != PairsTag) return aLoL;
  return asCar(aLoL);
}

JObj* cdrImpl(JObj* aLoL) {
  if (!aLoL) return NULL;
  if (asTag(aLoL) != PairsTag) return NULL;
  return asCdr(aLoL);
}
```

Test case

should return the car and cdr

```
  JObj *pairA = newPair(jInterp, NULL, NULL);
  JObj *pairB = newPair(jInterp, NULL, NULL);
  JObj *aPair = newPair(jInterp, pairA, pairB);
  AssertPtrEquals(getCar(jInterp, aPair), pairA);
  AssertPtrEquals(getCdr(jInterp, aPair), pairB);
```

3.15.2.3 Test Suite: popListInto

CHeader : public

```
#define popListInto(aCtx, aList, aJObjVar)          \
  JObj* aJObjVar = NULL;                            \
  if (isPair(aList)) {                              \
```

```
    aJObjVar = asCar(aList);                                    \
    aList    = asCdr(aList);                                    \
  }                                                             \
  if (aCtx->tracingOn) {                                        \
    JoyLoLInterp *jInterp = aCtx->jInterp;                      \
    StringBufferObj *aStrBuf = newStringBuffer(aCtx);           \
    strBufPrintf(aStrBuf, "%s.%s = ", #aList, #aJObjVar);       \
    printLoL(aStrBuf, aJObjVar);                                \
    strBufPrintf(aStrBuf, "\n");                                \
    jInterp->writeStdOut(jInterp, getCString(aStrBuf));         \
    strBufClose(aStrBuf);                                       \
  }
```

3.15.2.4 Test Suite: concatLists

CHeader : public

```
typedef JObj *(ConcatLists)(
  JoyLoLInterp *jInterp,
  JObj         *firstList,
  JObj         *secondList
);

#define concatLists(jInterp, firstList, secondList)       \
  (                                                       \
    assert(getPairsClass(jInterp)->concatListsFunc),      \
    (getPairsClass(jInterp)                               \
      ->concatListsFunc(jInterp, firstList, secondList))  \
  )
```

CHeader : private

```
extern JObj* concatListsImpl(
  JoyLoLInterp *jInterp,
  JObj         *firstList,
  JObj         *secondList
);
```

CCode : default

```
JObj* concatListsImpl(
  JoyLoLInterp *jInterp,
  JObj         *firstList,
  JObj         *secondList
```

```
) {
  assert(jInterp);

  JObj *result = NULL;

  if (firstList && isPair(firstList)) {
    result = firstList;
  } else {
    result = newPairImpl(jInterp, firstList, NULL);
  }

  if (!secondList) return result;
  // ensure that the second list is a LIST
  if (!isPair(secondList)) {
    secondList = newPair(jInterp, secondList, NULL);
  }

  if (!asCar(result)) return secondList;

  // find end of firstList/result
  JObj* lolList = result;
  while(asCdr(lolList) && isPair(asCdr(lolList))) {
    lolList = asCdr(lolList);
  }

  // ensure that if firstList/result ends in a non-pair we make it a pair
  if (asCdr(lolList) && !isPair(asCdr(lolList))) {
    asCdr(lolList) = newPair(jInterp, asCdr(lolList), NULL);
    assert(asCdr(lolList));
    lolList = asCdr(lolList);
  }

  // place secondList at the end of firstList/result
  assert(!asCdr(lolList));
  asCdr(lolList) = secondList;

  return result;
}
```

Test case

should concatinate two LoLs

Code 3.15.2

```
AssertPtrNotNull(jInterp);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);

JObj* result = concatLists(jInterp, NULL, NULL);
AssertPtrNotNull(result);
AssertIntTrue(isPair(result));
printLoL(aStrBuf, result);
AssertStrEquals(getCString(aStrBuf), "( ( ) ) ");
strBufClose(aStrBuf);

JObj *trueBool  = newBoolean(jInterp, TRUE);
result = concatLists(jInterp, trueBool, NULL);
AssertPtrNotNull(result);
AssertIntTrue(isPair(result));
AssertPtrNotNull(asCar(result));
AssertIntTrue(isTrue(asCar(result)));
AssertPtrNull(asCdr(result));
printLoL(aStrBuf, result);
AssertStrEquals(getCString(aStrBuf), "( true ) ");
strBufClose(aStrBuf);

JObj *falseBool = newBoolean(jInterp, FALSE);
result = concatLists(jInterp, trueBool, falseBool);
AssertPtrNotNull(result);
AssertIntTrue(isPair(result));
AssertPtrNotNull(asCar(result));
AssertIntTrue(isTrue(asCar(result)));
AssertPtrNotNull(asCdr(result));
AssertIntTrue(isFalse(asCdr(result)));
printLoL(aStrBuf, result);
AssertStrEquals(getCString(aStrBuf), "( true false ) ");
strBufClose(aStrBuf);

JObj* firstList = result;
result = concatLists(jInterp, result, NULL);
AssertPtrEquals(firstList, result);
printLoL(aStrBuf, result);
AssertStrEquals(getCString(aStrBuf), "( true false ) ");
strBufClose(aStrBuf);

result = concatLists(jInterp, firstList, trueBool);
AssertPtrNotNull(result);
AssertPtrEquals(firstList, result);
```

```
  AssertPtrNotNull(asCar(result));
  AssertIntTrue(isTrue(asCar(result)));
  AssertPtrNotNull(asCdr(result));
  AssertIntTrue(isPair(asCdr(result)));
  AssertIntTrue(isFalse(asCar(asCdr(result))));
  printLoL(aStrBuf, result);
  AssertStrEquals(getCString(aStrBuf), "( true false true ) ");
  strBufClose(aStrBuf);
```

3.15.2.5 Test Suite: copyLoL

CHeader : public

```
typedef JObj* (CopyLoL)(
  JoyLoLInterp* jInterp,
  JObj* aLoL
);

#define copyLoL(jInterp, aLoL)      \
  (                                 \
    assert(getPairsClass(jInterp)   \
      ->copyLoLFunc),               \
    (getPairsClass(jInterp)         \
      ->copyLoLFunc(jInterp, aLoL)) \
  )
```

CHeader : private

```
extern JObj* copyLoLImpl(
  JoyLoLInterp* jInterp,
  JObj* aLoL
);
```

CCode : default

```
JObj* copyLoLImpl(
  JoyLoLInterp* jInterp,
  JObj* aLoL
) {
  assert(jInterp);
  if (!aLoL) return NULL;
  if (isAtom(aLoL)) return aLoL;

  return newPair(jInterp,
```

Code 3.15.2

```
                copyLoLImpl(jInterp, asCar(aLoL)),
                copyLoLImpl(jInterp, asCdr(aLoL))
                );
}
```

Test case

should make a correct copy

```
JObj *bool   = newBoolean(jInterp, TRUE);
JObj *pairA  = newPair(jInterp, bool, NULL);
JObj *pairB  = newPair(jInterp, pairA, bool);
JObj *aPair0 = newPair(jInterp, pairA, pairB);
JObj *aPair1 = copyLoL(jInterp, aPair0);
AssertPtrNotNull(bool);
AssertPtrNotNull(pairA);
AssertPtrNotNull(pairB);
AssertPtrNotNull(aPair0);
AssertPtrNotNull(aPair1);
AssertPtrNotNull(asType(aPair1));
AssertPtrEquals(asType(aPair1),
  (JClass*)getPairsClass(jInterp));

AssertPtrNotNull(asCar(aPair1));
AssertPtrNotEquals(asCar(aPair1), pairA);

AssertPtrNotNull(asCar(asCar(aPair1)));
AssertPtrEquals(asCar(asCar(aPair1)), bool);
AssertIntTrue(isBoolean(asCar(asCar(aPair1))));
AssertIntTrue(isTrue(asCar(asCar(aPair1))));

AssertPtrNull(asCdr(asCar(aPair1)));

AssertPtrNotNull(asCdr(aPair1));
AssertPtrNotEquals(asCdr(aPair1), pairB);

AssertPtrNotNull(asCar(asCdr(aPair1)));
AssertIntTrue(isPair(asCar(asCdr(aPair1))));

AssertPtrNotNull(asCdr(asCdr(aPair1)));
AssertPtrEquals(asCdr(asCdr(aPair1)), bool);
AssertIntTrue(isBoolean(asCdr(asCdr(aPair1))));
AssertIntTrue(isTrue(asCdr(asCdr(aPair1))));
```

3.15.2.6 Test Suite: equalLoL

CHeader : public

```
typedef Boolean (EqualLoL)(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);

#define equalLoL(jInterp, lolA, lolB, ttl)      \
  (                                             \
    assert(getPairsClass(jInterp)               \
      ->equalLoLFunc),                          \
    (getPairsClass(jInterp)                     \
      ->equalLoLFunc(jInterp, lolA, lolB, ttl)) \
  )
#define lolEqual(jInterp, areEqual, lolA, lolB, ttl)  \
  if (areEqual && (lolA)) {                                    \
    assert(asType(lolA));                                      \
    areEqual = (asType(lolA)->equalityFunc)                \
      (jInterp, (lolA), (lolB), (ttl));                \
  }
```

CHeader : private

```
Boolean equalLoLImpl(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalLoLImpl(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "equalLoL %p %p %zu\n", lolA, lolB, timeToLive);
```

Code 3.15.2

```
  if (!lolA && !lolB) return TRUE;
  if (!lolA || !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (asType(lolA) &&
     (asTag(lolA) != PairsTag)) {
    return (asType(lolA)->equalityFunc)
      (jInterp, lolA, lolB, timeToLive);
  }

  if (timeToLive < 1) return TRUE;

  if (!equalLoLImpl(
      jInterp,
      asCar(lolA),
      asCar(lolB),
      (timeToLive-1)
    )) {
    return FALSE;
  }
  return equalLoLImpl(
    jInterp,
    asCdr(lolA),
    asCdr(lolB),
    (timeToLive -1)
  );
}
```

Test case

should return true if pairs are equal

```
  JObj *pairA = newPair(jInterp, NULL, NULL);
  JObj *pairB = newPair(jInterp, NULL, NULL);
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getPairsClass(jInterp));
  AssertPtrNotNull(getPairsClass(jInterp)->equalLoLFunc);
  AssertIntTrue(equalLoL(jInterp, NULL, NULL, 10));
  AssertIntTrue(equalLoL(jInterp, pairA, pairB, 10));
  // need to test with NON-Pairs
```

Test case

should return false if pairs are not equal

```
JObj *pairA = newPair(jInterp, NULL,  NULL);
JObj *pairB = newPair(jInterp, pairA, NULL);
AssertIntFalse(equalLoL(jInterp, pairA, pairB, 10));
// need to test with NON-Pairs
```

3.15.2.7 Test Suite: printing pairs

CHeader : public

```
#define lolPrintStr(                                    \
  aStrBuf, printedOk, aLoL, opener, closer, ttl)    \
  if (aLoL) {                                                \
    size_t isList = asTag(aLoL) == PairsTag;         \
    if (isList) strBufPrintf((aStrBuf), (opener));  \
    assert(asType(aLoL));                                    \
    (printedOk) = (printedOk) &&                     \
      (asType(aLoL)->printFunc)                      \
        ((aStrBuf), (aLoL), (ttl));                  \
    if (isList) strBufPrintf((aStrBuf), (closer));  \
  } else {                                                   \
    strBufPrintf((aStrBuf), (opener));                \
    strBufPrintf((aStrBuf), (closer));                \
  }
```

CHeader : private

```
extern Boolean printPairsCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printPairsCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asType(aLoL));
```

Code 3.15.2

```
  assert(asTag(aLoL) == PairsTag);

  if (timeToLive < 1) {
    strBufPrintf(aStrBuf, "... ");
    return TRUE;
  }

  size_t printedOk = TRUE;
  lolPrintStr(
    aStrBuf,
    printedOk,
    asCar(aLoL),
    "( ", ") ",
    timeToLive-1
  );
  lolPrintStr(
    aStrBuf,
    printedOk,
    asCdr(aLoL),
    "",   "",
    timeToLive-1
  );
  return printedOk;
}
```

Test case

should print pairs

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(getPairsClass(jInterp));

  JObj* aNewPair = newPair(jInterp,
                           newPair(jInterp, NULL, NULL),
                           newPair(jInterp, NULL, NULL));
  AssertPtrNotNull(aNewPair);
  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  printLoL(aStrBuf, aNewPair);
  AssertStrEquals(getCString(aStrBuf), "( ( ( ) ) ( ) ) ");
  strBufClose(aStrBuf);
```

3.15.2.8 Test Suite: registerPairs

CHeader : public

```
typedef struct pairs_class_struct {
  JClass       super;
  NewPair     *newPairFunc;
  CarCdr      *carFunc;
  CarCdr      *cdrFunc;
  ConcatLists *concatListsFunc;
  EqualLoL    *equalLoLFunc;
  CopyLoL     *copyLoLFunc;
} PairsClass;
```

CCode : default

```
static Boolean initializePairs(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerPairs(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerPairs(JoyLoLInterp *jInterp) {
  assert(jInterp);

  PairsClass* theCoAlg           = joyLoLCalloc(1, PairsClass);
  theCoAlg->super.name           = PairsName;
  theCoAlg->super.objectSize     = sizeof(PairsObj);
  theCoAlg->super.initializeFunc = initializePairs;
  theCoAlg->super.registerFunc   = registerPairWords;
  theCoAlg->super.equalityFunc   = equalLoLImpl;
  theCoAlg->super.printFunc      = printPairsCoAlg;
  theCoAlg->newPairFunc          = newPairImpl;
  theCoAlg->carFunc              = carImpl;
  theCoAlg->cdrFunc              = cdrImpl;
  theCoAlg->concatListsFunc      = concatListsImpl;
  theCoAlg->equalLoLFunc         = equalLoLImpl;
```

```
  theCoAlg->copyLoLFunc          = copyLoLImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // sanity check...
  assert(tag == PairsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Pairs coAlg

```
  // CTestSetup has already created a jInterp
  // and run registerPairs
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(getPairsClass(jInterp));
  PairsClass *coAlg = getPairsClass(jInterp);
  AssertIntTrue(registerPairs(jInterp));
  AssertPtrNotNull(getPairsClass(jInterp));
  AssertPtrEquals(getPairsClass(jInterp), coAlg);
  AssertIntEquals(
    getPairsClass(jInterp)->super.objectSize,
    sizeof(PairsObj)
  );
```

3.15.3 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```c
static int lua_pairs_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_pairs [] = {
  {"gitVersion", lua_pairs_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_pairs (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerPairs(jInterp);
  luaL_newlib(lstate, lua_pairs);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedPairs` does just this.

CHeader : public

```c
Boolean requireStaticallyLinkedPairs(
  lua_State *lstate
);
```

CCode : default

```c
Boolean requireStaticallyLinkedPairs(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_pairs(lstate);
  lua_setfield(lstate, -2, "joylol.pairs");
  lua_setfield(lstate, -2, "loaded");
```

```
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.15.4 JoyLoL words

CCode : default

```
static void isNilAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (!top)
    result = newBoolean(jInterp, TRUE);
  else
    result = newBoolean(jInterp, FALSE);
  pushCtxData(aCtx, top);
  pushCtxData(aCtx, result);
}
```

CCode : default

```
static void isPairAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isPair(top))
    result = newBoolean(jInterp, TRUE);
  else
    result = newBoolean(jInterp, FALSE);
  pushCtxData(aCtx, top);
  pushCtxData(aCtx, result);
}
```

CCode : default

```
static void isAtomAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);
```

```
  popCtxDataInto(aCtx, top);
  JObj* result = NULL;
  if (isAtom(top))
    result = newBoolean(jInterp, TRUE);
  else
    result = newBoolean(jInterp, FALSE);
  pushCtxData(aCtx, top);
  pushCtxData(aCtx, result);
}
```

3.15.4.1 Test Suite: initPairsCoAlgebra

CHeader : private

```
extern Boolean registerPairWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerPairWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  extendJoyLoLInRoot(jInterp, "isNil",  "", isNilAP, "");
  extendJoyLoLInRoot(jInterp, "isList", "", isPairAP, "");
  extendJoyLoLInRoot(jInterp, "isPair", "", isPairAP, "");
  extendJoyLoLInRoot(jInterp, "isAtom", "", isAtomAP, "");

  return TRUE;
}
```

3.15.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <assert.h>
#include <joylol/jInterps.h>
```

```
4   #include <joylol/booleans.h>
5   #include <joylol/stringBuffers.h>
6   #include <joylol/cFunctions.h>
7   #include <joylol/texts.h>
8   #include <joylol/assertions.h>
9   #include <joylol/contexts.h>
10  #include <joylol/pairs.h>
11  #include <joylol/pairs-private.h>
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.booleans");
requireStaticallyLinkedPairs(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.16 Parsers

3.16.1 Goals

The 'native' JoyLoL parser is a Parsing Expression Grammar (PEG). It is intended to be a similar as possible to the Lua based LPeg parser grammar.

3.16.2 Code

CCode : default

```
1 // The Parser parses a stream of characters obtained from a specific
2 // text. Since one of the texts is backed by readline interaction with
3 // a user, it is **critical** that nextSymbol ONLY get called when
4 // the parser actually needs a nextSymbol (and NOT before).
```

We start by providing some example strings that we want to be able to parse. These will each be used as `CTest` examples below.

```
static Symbol* simpleList[] = {
  "this is a simple list ;; this is a comment ",
  NULL
};
//
static Symbol* simpleListWithSemiColons[] = {
  " this ; is;a ;; this is a comment ",
  " and;again ;; this is a second comment ",
  NULL
};
//
static Symbol* complexListWithSemiColons[] = {
  " this ; is;a test with some semi-colons ;; this is a comment ",
  " and;again;123;456;789 12;34 ;; this is a second comment ",
  " and;one;more;time ;; this is a third comment ",
  NULL
};
//
static Symbol* moreComplexList[] = {
  " this ( is ( a ( more ( complex ( list () ) ) ) ) ) ;; this is a comment ",
  NULL
};
//
```

```
static Symbol* differentList[] = {
  " this is  a different ( ( () () () ) ( ( ( ) ) ) ) list ;; with a comment  ",
  NULL
};
//
//static Symbol* factorialStrs[] = {
//  " ( 0 =nat ) ( pop 1 ) ( dup 1 - factorial * ) ifte ",
//  NULL
//};
//
//static Symbol* incorrectMarkerList[] = {
//  " this ( is a list with incorrectly > matched list markers ",
//  NULL
//};
//
//static Symbol* unmatchedMarkerList[] = {
//  " this ( is a list [ with unmatched list markers ",
//  NULL
//};
//
```

3.16.2.1 Test Suite: match list symbols

CHeader : private

```
extern char matchingListSymbol(
  Symbol* curSymbol
);
```

CCode : default

```
char matchingListSymbol(
  Symbol* curSymbol
) {
  if (1 < strlen(curSymbol)) return 0;

  if (curSymbol[0] == '(') return ')';
  if (curSymbol[0] == '[') return ']';
  if (curSymbol[0] == '{') return '}';

  return 0;
}
```

CHeader : private

```
extern size_t isClosingSymbol(
  Symbol* curSymbol
);
```

CCode : default

```
size_t isClosingSymbol(
  Symbol* curSymbol
) {
  if (1 < strlen(curSymbol)) return 0;

  if (curSymbol[0] == ')') return 1;
  if (curSymbol[0] == ']') return 1;
  if (curSymbol[0] == '}') return 1;

  return 0;
}
```

CHeader : private

```
void reportMismatchedClosingSymbol(
  TextObj* aText,
  char expected,
  char found
);
```

CCode : default

```
void reportMismatchedClosingSymbol(
  TextObj* aText,
  char expected,
  char found
) {
  if (expected == 0) return;

  char message[100];
  memset(message, 0, 100);
  strcat(message, "closing list expected [");
  message[strlen(message)] = expected;
  strcat(message, "] but found [");
  message[strlen(message)] = found;
  strcat(message, "]");
  reportError(aText, message);
}
```

CHeader : private

```
void reportUnmatchedClosingSymbol(
  TextObj* aText,
  char expected
);
```

CCode : default

```
void reportUnmatchedClosingSymbol(
  TextObj* aText,
  char expected
) {
  if (expected == 0) return;

  char message[100];
  memset(message, 0, 100);
  strcat(message, "adding unmatched list closing [");
  message[strlen(message)] = expected;
  strcat(message, "]");
  reportError(aText, message);
}
```

Test case

should match List Symbols

```
  AssertIntEquals(matchingListSymbol("("), ')');
  AssertIntEquals(matchingListSymbol("["), ']');
  AssertIntEquals(matchingListSymbol("{"), '}');
  AssertIntEquals(matchingListSymbol("#"), 0);
  AssertIntEquals(matchingListSymbol("silly"), 0);
```

CHeader : private

```
extern JObj* parseList(
  TextObj* aText,
  char closingChar
);
```

CCode : default

```
JObj* parseList(
  TextObj* aText,
  char closingChar
```

```
4   ) {
5     assert(aText);
6     assert(aText->jInterp);
7     DEBUG(aText->jInterp,
8       "parseList %p '%c'\n", aText, closingChar);
9
10    JObj* firstPair = NULL;
11    JObj* lastPair  = NULL;
12
13    while (!aText->completed) {
14
15      nextSymbol(aText);
16      DEBUG(aText->jInterp,
17        "parseList ns: %p %zu\n",
18          aText->sym, (aText->sym ? (size_t)asTag(aText->sym) : 0));
19      if (!aText->sym) {
20        reportUnmatchedClosingSymbol(aText, closingChar);
21        return firstPair;
22      }
23
24      // check to see if this is a closing list symbol
25      if (isSymbol(aText->sym) &&
26          isClosingSymbol(asSymbol(aText->sym))) {
27        if (asSymbol(aText->sym)[0] != closingChar) {
28          reportMismatchedClosingSymbol(aText, closingChar,
29                                        asSymbol(aText->sym)[0]);
30        }
31        if (closingChar == '}') {
32          DEBUG(aText->jInterp, "parseList(assertion-a) %p %zu\n",
33            firstPair, (firstPair ? (size_t)asTag(firstPair) : 0));
34          firstPair = newAssertion(aText->jInterp, firstPair);
35          assert(isAssertion(firstPair));
36          DEBUG(aText->jInterp, "parseList(assertion-b) %p %zu\n",
37            firstPair, (firstPair ? (size_t)asTag(firstPair) : 0));
38        }
39        return firstPair;
40      }
41      assert(aText->jInterp);
42      JObj* aNewPair = newPair(aText->jInterp, NULL, NULL);
43
44      // check to see if this is an opening list symbol
45      if (isSymbol(aText->sym)) {
46        char matchingSymbol = matchingListSymbol(asSymbol(aText->sym));
```

```
47        if (0 < matchingSymbol) {
48          JObj *newList = parseList(aText, matchingSymbol);
49          asCar(aNewPair) = newList;
50        } else asCar(aNewPair) = aText->sym;
51      } else asCar(aNewPair) = aText->sym;
52
53      if (!firstPair) firstPair       = aNewPair;
54      if (lastPair)   asCdr(lastPair) = aNewPair;
55
56      lastPair = aNewPair;
57    }
58
59    if (0 < closingChar) reportUnmatchedClosingSymbol(aText, closingChar);
60    return firstPair;
61  }
```

3.16.2.2 Test Suite: parse one symbol

CHeader : public

```
1   typedef JObj *(ParseOneSymbol)(
2     TextObj *aText
3   );
4
5   #define parseOneSymbol(aText)                \
6     (                                          \
7       assert(aText),                           \
8       assert(getParsersClass(aText->jInterp)   \
9         ->parseOneSymbolFunc),                 \
10      (getParsersClass(aText->jInterp)         \
11        ->parseOneSymbolFunc(aText))           \
12    )
```

CHeader : private

```
1   extern JObj* parseOneSymbolImpl(
2     TextObj* aText
3   );
```

CCode : default

```
1   JObj* parseOneSymbolImpl(
2     TextObj* aText
3   ) {
```

```
  assert(aText);
  assert(aText->jInterp);
  DEBUG(aText->jInterp, "parseOneSymbol %p\n", aText);
  nextSymbol(aText);
  if (!aText->sym) {
    return newSignal(aText->jInterp, SIGNAL_END_OF_TEXT);
  }

  if (aText->sym->type &&
     (asTag(aText->sym) != SymbolsTag)) return aText->sym;

  // check to see if this is an opening list symbol
  char matchingSymbol = matchingListSymbol(asSymbol(aText->sym));
  if (0 < matchingSymbol) {
    return parseList(aText, matchingSymbol);
  }
  return aText->sym;
}
```

Test case

parse Simple List One Symbol At A Time

```
  AssertPtrNotNull(jInterp);

  TextObj* aText =
    createTextFromArrayOfStrings(jInterp, simpleList);
  AssertPtrNotNull(aText);

  JObj* aLoL = parseOneSymbol(aText);
  AssertPtrNotNull(aLoL);
  AssertPtrNotNull(asType(aLoL));
  AssertIntEquals(asTag(aLoL), SymbolsTag);
  AssertStrEquals(asSymbol(aLoL), "this");

  aLoL = parseOneSymbol(aText);
  AssertPtrNotNull(aLoL);
  AssertPtrNotNull(asType(aLoL));
  AssertIntEquals(asTag(aLoL), SymbolsTag);
  AssertStrEquals(asSymbol(aLoL), "is");

  aLoL = parseOneSymbol(aText);
  AssertPtrNotNull(aLoL);
```

Code 3.16.2

```
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "a");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "simple");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "list");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isSignal(aLoL));
AssertIntEquals(asSignal(aLoL), SIGNAL_END_OF_TEXT);
AssertIntTrue(aText->completed);
```

Test case

parse SemiColons One Symbol At A Time

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromArrayOfStrings(jInterp, simpleListWithSemiColons);

AssertPtrNotNull(aText);

JObj* aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "this");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
```

```
AssertStrEquals(asSymbol(aLoL), ";");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "is");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), ";");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "a");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "and");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), ";");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "again");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isSignal(aLoL));
AssertIntEquals(asSignal(aLoL), SIGNAL_END_OF_TEXT);
AssertIntTrue(aText->completed);
```

Test case

parse Complex List One Symbol At A Time

```
AssertPtrNotNull(jInterp);

TextObj* aText = createTextFromArrayOfStrings(jInterp, moreComplexList);
AssertPtrNotNull(aText);

JObj* aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), SymbolsTag);
AssertStrEquals(asSymbol(aLoL), "this");

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertPtrNotNull(asType(aLoL));
AssertIntEquals(asTag(aLoL), PairsTag);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
AssertStrEquals(getCString(aStrBuf),
  "( is ( a ( more ( complex ( list ( ) ) ) ) ) ) ");
strBufClose(aStrBuf);

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isSignal(aLoL));
AssertIntEquals(asSignal(aLoL), SIGNAL_END_OF_TEXT);
AssertIntTrue(aText->completed);
```

Test case

parse Boolean and find it in main dictionary

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->rootCtx);

DictObj* mainDic = jInterp->rootCtx->dict;
AssertPtrNotNull(mainDic);

TextObj* aText =
```

```
  createTextFromString(jInterp, " true false ");
AssertPtrNotNull(aText);

JObj* aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isAtom(aLoL));
AssertIntTrue(isSymbol(aLoL));
AssertStrEquals(asSymbol(aLoL), "true");
DictNodeObj* aNode =
  getSymbolEntry(mainDic, asSymbol(aLoL));
AssertPtrNotNull(aNode);
AssertPtrNotNull(aNode->value);
AssertIntTrue(isBoolean(aNode->value));
AssertIntTrue(asBoolean(aNode->value));

aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isAtom(aLoL));
AssertIntTrue(isSymbol(aLoL));
AssertStrEquals(asSymbol(aLoL), "false");
aNode = getSymbolEntry(mainDic, asSymbol(aLoL));
AssertPtrNotNull(aNode);
AssertPtrNotNull(aNode->value);
AssertIntTrue(isBoolean(aNode->value));
AssertIntFalse(asBoolean(aNode->value));
```

Test case

parse Print Naturals

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromString(jInterp, " 1234567890 not1234567890 ");
AssertPtrNotNull(aText);

JObj* aLoL = parseOneSymbol(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isAtom(aLoL));
AssertIntTrue(isNatural(aLoL));
AssertIntEquals( mpz_cmp_si(asNatural(aLoL), 1234567890), 0);
StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
```

```
  AssertStrEquals(getCString(aStrBuf), "1234567890 ");
  strBufClose(aStrBuf);

  aLoL = parseOneSymbol(aText);
  AssertPtrNotNull(aLoL);
  AssertIntTrue(isAtom(aLoL));
  AssertIntTrue(isSymbol(aLoL));
  AssertStrEquals(asSymbol(aLoL), "not1234567890");
```

3.16.2.3 Test Suite: parse all symbols

CHeader : public

```
typedef JObj *(ParseAllSymbols)(
  TextObj *aText
);

#define parseAllSymbols(aText)              \
  (                                         \
    assert(aText),                          \
    assert(getParsersClass(aText->jInterp)  \
      ->parseAllSymbolsFunc),               \
    (getParsersClass(aText->jInterp)        \
      ->parseAllSymbolsFunc(aText))         \
  )
```

CHeader : private

```
extern JObj* parseAllSymbolsImpl(
  TextObj* aText
);
```

CCode : default

```
JObj* parseAllSymbolsImpl(
  TextObj* aText
) {
  assert(aText);
  assert(aText->jInterp);
  DEBUG(aText->jInterp, "parseAllSymbols %p]\n", aText);
  return parseList(aText, 0);
}
```

We begin by parsing and printing some strings which exercise 'happy paths' in our parsing code.

Test case

parse SemiColons All At Once

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromArrayOfStrings(jInterp, complexListWithSemiColons);

AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);

AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 30);
AssertStrEquals(getCString(aStrBuf),
  "( this ; is ; a test with some semi-colons and ; again ; 123 ; 456 ; 789 12 ; 34 and ;
strBufClose(aStrBuf);
```

Test case

parse Print Simple List and test copyLoLs

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromArrayOfStrings(jInterp, simpleList);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
AssertStrEquals(getCString(aStrBuf), "( this is a simple list ) ");
strBufClose(aStrBuf);

JObj* lolCopy = copyLoL(jInterp, aLoL);
AssertPtrNotEquals(aLoL, lolCopy);
AssertIntTrue(equalLoL(jInterp, aLoL, lolCopy, 10));
```

Test case

parse Complex List From Single String

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromString(jInterp, moreComplexList[0]);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);

AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 20);
AssertStrEquals(getCString(aStrBuf),
  "( this ( is ( a ( more ( complex ( list ( ) ) ) ) ) ) ) ");
strBufClose(aStrBuf);
```

Test case

parse Print Complex List

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromArrayOfStrings(jInterp, moreComplexList);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 20);
AssertStrEquals(getCString(aStrBuf),
  "( this ( is ( a ( more ( complex ( list ( ) ) ) ) ) ) ) ");
strBufClose(aStrBuf);

JObj* lolCopy = copyLoL(jInterp, aLoL);
AssertPtrNotEquals(aLoL, lolCopy);
AssertIntTrue(equalLoL(jInterp, aLoL, lolCopy, 10));
```

Test case

parse Print Different List

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromArrayOfStrings(jInterp, differentList);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
AssertStrEquals(getCString(aStrBuf),
  "( this is a different ( ( ( ) ( ) ( ) ) ( ( ( ) ) ) ) list ) ");
strBufClose(aStrBuf);
```

Test case

parse assertion

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromString(jInterp, " { true } ");
AssertPtrNotNull(aText);

//jInterp->debug = TRUE;

JObj *aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
AssertIntTrue(isPair(aLoL));
AssertIntTrue(isAssertion(asCar(aLoL)));
AssertPtrNull(asCdr(aLoL));
```

Now we parse and print some strings which exercise 'unhappy paths' in our parsing code.

Test case

parse Incorrect Marker List

```
CoAlgebras* coAlgs = createCoAlgebras();
```

Code 3.16.2

```
AssertPtrNotNull(coAlgs);

Text* aText = createTextFromArrayOfStrings(coAlgs, incorrectMarkerList);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
AssertStrEquals(getCString(aStrBuf),
  "this ( is a list with incorrectly ) matched list markers");
strBufClose(aStrBuf);
```

IGNORED

Test case

parse Unmatched Marker List

```
CoAlgebras* coAlgs = createCoAlgebras();
AssertPtrNotNull(coAlgs);

Text* aText = createTextFromArrayOfStrings(coAlgs, unmatchedMarkerList);
AssertPtrNotNull(aText);

JObj* aLoL = parseAllSymbols(aText);
AssertPtrNotNull(aLoL);
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
printLoLTTL(aStrBuf, aLoL, 10);
AssertStrEquals(getCString(aStrBuf),
  "this ( is a list ( with unmatched list markers ) )");
strBufClose(aStrBuf);
```

IGNORED

3.16.2.4 Test Suite: registerParsers

CHeader : public

```
1 typedef struct parsers_class_struct {
2   JClass       super;
3   ParseOneSymbol  *parseOneSymbolFunc;
4   ParseAllSymbols *parseAllSymbolsFunc;
5 } ParsersClass;
```

CCode : default

```
static Boolean initializeParsers(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerParsers(
  JoyLoLInterp *jInterp
);
```

CCode : default

```
Boolean registerParsers(
  JoyLoLInterp *jInterp
) {
  assert(jInterp);

  ParsersClass* theCoAlg          = joyLoLCalloc(1, ParsersClass);
  theCoAlg->super.name            = ParsersName;
  theCoAlg->super.objectSize      = sizeof(JObj);
  theCoAlg->super.initializeFunc  = initializeParsers;
  theCoAlg->super.registerFunc    = registerParserWords;
  theCoAlg->super.equalityFunc    = NULL;
  theCoAlg->super.printFunc       = NULL;
  theCoAlg->parseOneSymbolFunc    = parseOneSymbolImpl;
  theCoAlg->parseAllSymbolsFunc   = parseAllSymbolsImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == ParsersTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Parsers coAlg

```
// CTestsSetup has already created a jInterp
// and run registerParsers

AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getParsersClass(jInterp));
ParsersClass *coAlg = getParsersClass(jInterp);
AssertIntTrue(registerParsers(jInterp));
AssertPtrNotNull(getParsersClass(jInterp));
AssertPtrEquals(getParsersClass(jInterp), coAlg);
AssertIntEquals(
  getParsersClass(jInterp)->super.objectSize,
  sizeof(JObj)
)
```

3.16.3 Patterns

```
\startJoyLoLWord[pegPattern]

\preDataStack[][]

\preProcessStack[][]

\startCCode

\stopCCode

\startJoyLoLCode

\stopJoyLoLCode

\postDataStack[]

\postProcessStack[]

\stopJoyLoLWord

\startJoyLoLWord[pegSet]
```

```
\preDataStack[][]

\preProcessStack[][]

\startCCode

\stopCCode

\startJoyLoLCode

\stopJoyLoLCode

\postDataStack[]

\postProcessStack[]

\stopJoyLoLWord

\startJoyLoLWord[pegRange]

\preDataStack[][]

\preProcessStack[][]

\startCCode

\stopCCode

\startJoyLoLCode

\stopJoyLoLCode

\postDataStack[]

\postProcessStack[]

\stopJoyLoLWord
```

3.16.4 Captures

3.16.5 JoyLoL's parser

```
\startJoyLoLWord[parseJoyLoL]
```

```
\preDataStack[][]

\preProcessStack[][]

\startJoyLoLCode

\stopJoyLoLCode

\startLuaCode

local luaTest = { }
function luaTest.test()
  -- this is a test
  return { }
end

return luaTest

\stopLuaCode

\postDataStack[]

\postProcessStack[]

\stopJoyLoLWord
```

3.16.6 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_parsers_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
```

```
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_parsers [] = {
  {"gitVersion", lua_parsers_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_parsers (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerParsers(jInterp);
  luaL_newlib(lstate, lua_parsers);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedParsers` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedParsers(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedParsers(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_parsers(lstate);
  lua_setfield(lstate, -2, "joylol.parsers");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.16.7 JoyLoL words

CHeader : private

```
extern Boolean registerParserWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerParserWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
){
  return TRUE;
}
```

3.16.8 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/assertions.h>
#include <joylol/booleans.h>
#include <joylol/symbols.h>
#include <joylol/signals.h>
#include <joylol/naturals.h>
#include <joylol/pairs.h>
#include <joylol/dictNodes.h>
#include <joylol/texts.h>
#include <joylol/parsers.h>
#include <joylol/parsers-private.h>
// dictionary
// printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.booleans");
requireLuaModule(lstate, "joylol.signals");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.naturals");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.texts");
requireStaticallyLinkedParsers(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.17 JoyLoL rules

3.17.1 Goals

A code rules structure contains a collection of pre and post condition assertion which can be used by the cross compiler.

3.17.2 Code

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CHeader : public

```c
typedef struct rule_object_struct {
  JObj    super;
  Symbol *name;
  Symbol *body;
} RuleObj;
```

3.17.2.1 Test Suite: newRule

CHeader : public

```c
typedef RuleObj* (NewRule)(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
);

#define newRule(jInterp, aName, aBody)      \
  (                                         \
    assert(getRulesClass(jInterp)           \
      ->newRuleFunc),                       \
```

Code 3.17.2

```
    (getRulesClass(jInterp)                  \
      ->newRuleFunc(jInterp, aName, aBody)) \
  )
//#define asRule(aLoL) (((aLoL)->flags) & BOOLEAN_FLAG_MASK)
```

CHeader : private

```
extern RuleObj* newRuleImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
);
```

CCode : default

```
RuleObj* newRuleImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aName,
  Symbol       *aBody
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  RuleObj *result =
    (RuleObj*)newObject(jInterp, RulesTag);
  assert(result);

//  result->super.type  = jInterp->coAlgs[RulesTag];
  result->name        = strdup(aName);
  result->body        = strdup(aBody);

  return result;
}
```

Test case

should create a new Rule

```
  AssertPtrNotNull(jInterp);

  RuleObj* aNewRule =
    newRule(jInterp, "ansiC", "a rule body");
  AssertPtrNotNull(aNewRule);
  AssertPtrNotNull(asType(aNewRule));
```

```
  AssertIntEquals(asTag(aNewRule), RulesTag);
  AssertIntTrue(isAtom(aNewRule));
  AssertIntTrue(isRule(aNewRule));
  AssertIntFalse(isPair(aNewRule));
```

Test case

print Rule

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  RuleObj* aLoL =
    newRule(jInterp, "ansiC", "a rule body");
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, (JObj*)aLoL);
  AssertStrEquals(getCString(aStrBuf), "rule ");
  strBufClose(aStrBuf);
```

3.17.2.2 Test Suite: isRule

CHeader : public

```
#define isRule(aLoL)             \
  (                                \
    (                              \
      (aLoL) &&                    \
      asType(aLoL) &&              \
      (asTag(aLoL) == RulesTag) \
    ) ?                            \
      TRUE :                       \
      FALSE                        \
  )
```

CHeader : private

```
extern Boolean equalityRuleCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
```

```
);
```

CCode : default

```
Boolean equalityRuleCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "ruleCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != RulesTag) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
}
```

3.17.2.3 Test Suite: printing rules

CHeader : private

```
extern Boolean printRuleCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printRuleCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == RulesTag);

  strBufPrintf(aStrBuf, "rule ");
  return TRUE;
```

```
}
```

Test case

should print rules

```
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs[RulesTag]);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

RuleObj* aNewRule =
  newRule(jInterp, "ansiC", "a rule body");
AssertPtrNotNull(aNewRule);
printLoL(aStrBuf, (JObj*)aNewRule);
AssertStrEquals(getCString(aStrBuf), "rule ");
strBufClose(aStrBuf);
```

3.17.2.4 Test Suite: registerRules

CHeader : public

```
typedef struct rules_class_struct {
  JClass       super;
  NewRule      *newRuleFunc;
} RulesClass;
```

CCode : default

```
static Boolean initializeRules(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerRules(JoyLoLInterp *jInterp);
```

Code 3.17.3

CCode : default

```
Boolean registerRules(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  RulesClass* theCoAlg
    = joyLoLCalloc(1, RulesClass);
  assert(theCoAlg);

  theCoAlg->super.name           = RulesName;
  theCoAlg->super.objectSize     = sizeof(RuleObj);
  theCoAlg->super.initializeFunc = initializeRules;
  theCoAlg->super.registerFunc   = registerRuleWords;
  theCoAlg->super.equalityFunc   = equalityRuleCoAlg;
  theCoAlg->super.printFunc      = printRuleCoAlg;
  theCoAlg->newRuleFunc = newRuleImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == RulesTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Rules coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerRules
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getRulesClass(jInterp));
  RulesClass *coAlg = getRulesClass(jInterp);
  registerRules(jInterp);
  AssertPtrNotNull(getRulesClass(jInterp));
  AssertPtrEquals(getRulesClass(jInterp), coAlg);
  AssertIntEquals(
    getRulesClass(jInterp)->super.objectSize,
    sizeof(RuleObj)
  )
```

3.17.3 Words

CHeader : private

```
extern Boolean registerRuleWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerRuleWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  return TRUE;
}
```

3.17.4 Lua functions

CCode : default

```
static int lua_rules_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_rules [] = {
  {"gitVersion", lua_rules_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_rules (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerRules(jInterp);
```

```
  luaL_newlib(lstate, lua_rules);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedRules` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedRules(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedRules(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_rules(lstate);
  lua_setfield(lstate, -2, "joylol.rules");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.17.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_rules(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
```

```
7  #include <joylol/texts.h>
8  #include <joylol/cFunctions.h>
9  #include <joylol/assertions.h>
10 #include <joylol/contexts.h>
11 #include <joylol/rules.h>
12 #include <joylol/rules-private.h>
13 // dictionary
14 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedRules(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.18 Internal Signals

3.18.1 Goals

An internal signal is used to signal different events between separate parts of the interpreter.

3.18.2 Code

CCode : default

```c
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

3.18.2.1 Test Suite: newSignal

CHeader : public

```c
#define SIGNAL_END_OF_TEXT 1
typedef JObj* (NewSignal)(
  JoyLoLInterp*,
  size_t
);

#define newSignal(jInterp, aSignal)       \
  (                                       \
    assert(getSignalsClass(jInterp)       \
      ->newSignalFunc),                   \
    (getSignalsClass(jInterp)             \
      ->newSignalFunc(jInterp, aSignal))  \
  )
#define asSignal(aLoL) ((aLoL)->flags)
```

CHeader : private

```c
extern JObj* newSignalImpl(
```

Code 3.18.2

```
  JoyLoLInterp *jInterp,
  size_t        aSignal
);
```

CCode : default

```
JObj* newSignalImpl(
  JoyLoLInterp *jInterp,
  size_t        aSignal
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JObj* result = newObject(jInterp, SignalsTag);
  assert(result);

  result->type  = jInterp->coAlgs[SignalsTag];
  result->flags = aSignal;
  return result;
}
```

Test case

should create a new signal

```
  AssertPtrNotNull(jInterp);

  JObj* aNewSignal = newSignal(jInterp, 12);
  AssertPtrNotNull(aNewSignal);
  AssertPtrNotNull(asType(aNewSignal));
  AssertIntEquals(asTag(aNewSignal), SignalsTag);
  AssertIntEquals(asFlags(aNewSignal), 12);
  AssertIntTrue(isAtom(aNewSignal));
  AssertIntTrue(isSignal(aNewSignal));
  AssertIntFalse(isPair(aNewSignal));
```

Test case

print Signal

```
  AssertPtrNotNull(jInterp);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
```

```
  AssertPtrNotNull(aStrBuf);

  JObj* aLoL = newSignal(jInterp, 12);
  AssertPtrNotNull(aLoL);
  printLoL(aStrBuf, aLoL);
  AssertStrEquals(getCString(aStrBuf), "signal:12 ");
  strBufClose(aStrBuf);
```

3.18.2.2 Test Suite: isSignal

CHeader : public

```
#define isSignal(aLoL)              \
  (                                 \
    (                               \
      (aLoL) &&                     \
      asType(aLoL) &&               \
      (asTag(aLoL) == SignalsTag) \
    ) ?                             \
      TRUE :                        \
      FALSE                         \
  )
```

Test case

should return appropriate signal values

```
  JObj *aSignal = newSignal(jInterp, 12);
  AssertPtrNotNull(aSignal);
  AssertPtrNotNull(asType(aSignal));
  AssertIntEquals(asTag(aSignal), SignalsTag);
  AssertIntEquals(asSignal(aSignal), 12);
```

CHeader : private

```
extern Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityBoolCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "boolCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != SignalsTag) return FALSE;
  if (asSignal(lolA) != asSignal(lolB)) return FALSE;
  return TRUE;
}
```

3.18.2.3 Test Suite: printing signals

CHeader : private

```
extern Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printBoolCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == SignalsTag);

  strBufPrintf(aStrBuf, "signal:%zu ", asSignal(aLoL));
  return TRUE;
}
```

Test case

should print signals

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[SignalsTag]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aNewSignal = newSignal(jInterp, 12);
  AssertPtrNotNull(aNewSignal);
  printLoL(aStrBuf, aNewSignal);
  AssertStrEquals(getCString(aStrBuf), "signal:12 ");
  strBufClose(aStrBuf);
```

3.18.2.4 Test Suite: registerSignals

CHeader : public

```
typedef struct signals_class_struct {
  JClass     super;
  NewSignal *newSignalFunc;
} SignalsClass;
```

CCode : default

```
static Boolean initializeSignals(
  JoyLoLInterp *jInterp,
  JClass       *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerSignals(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerSignals(JoyLoLInterp *jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);
```

Code 3.18.2

```
  SignalsClass* theCoAlg
    = joyLoLCalloc(1, SignalsClass);
  assert(theCoAlg);

  theCoAlg->super.name           = SignalsName;
  theCoAlg->super.objectSize     = sizeof(JObj);
  theCoAlg->super.initializeFunc = initializeSignals;
  theCoAlg->super.registerFunc   = registerSignalWords;
  theCoAlg->super.equalityFunc   = equalityBoolCoAlg;
  theCoAlg->super.printFunc      = printBoolCoAlg;
  theCoAlg->newSignalFunc        = newSignalImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == SignalsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Signals coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerSignals
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getSignalsClass(jInterp));
  SignalsClass *coAlg = getSignalsClass(jInterp);
  registerSignals(jInterp);
  AssertPtrNotNull(getSignalsClass(jInterp));
  AssertPtrEquals(getSignalsClass(jInterp), coAlg);
  AssertIntEquals(
    getSignalsClass(jInterp)->super.objectSize,
    sizeof(JObj)
  )
```

3.18.3 Words

CHeader : private

```
extern Boolean registerSignalWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerSignalWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  return TRUE;
}
```

3.18.4 Lua functions

CCode : default

```
static int lua_signals_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_signals [] = {
  {"gitVersion", lua_signals_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_signals (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerSignals(jInterp);
  luaL_newlib(lstate, lua_signals);
  return 1;
```

```
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedSignals` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedSignals(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedSignals(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_signals(lstate);
  lua_setfield(lstate, -2, "joylol.signals");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.18.5 Conclusions

CHeader : public

CHeader : private

```
extern size_t joylol_register_signals(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
#include <joylol/dictionaries.h>
#include <joylol/texts.h>
```

```
9  #include <joylol/cFunctions.h>
10 #include <joylol/assertions.h>
11 #include <joylol/contexts.h>
12 #include <joylol/signals.h>
13 #include <joylol/signals-private.h>
14 // dictionary
15 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedSignals(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.19 String buffers

3.19.1 Goals

The symbols extension extends the core JoyLoL by providing support for string buffers which can be appended to.

3.19.2 Code

CHeader : public

```
typedef struct string_buffer_object_struct {
  JObj           super;
  JoyLoLInterp *jInterp;
  ContextObj   *theCtx;
  FILE         *memFile;
  char         *buffer;
  size_t        bufSize;
} StringBufferObj;
```

CHeader : private

```
#define asMemFile(aLoL)    (((StringBufferObj*)(aLoL))->memFile)
#define asBuffer(aLoL)     (((StringBufferObj*)(aLoL))->buffer)
#define asBufSize(aLoL)    (((StringBufferObj*)(aLoL))->bufSize)
```

3.19.2.1 Test Suite: newStringBuffer

CHeader : public

```
typedef StringBufferObj* (NewStringBuffer)(
  ContextObj *theCtx
);

#define newStringBuffer(theCtx)                    \
  (                                                \
    assert(theCtx),                                \
    assert(theCtx->jInterp),                       \
    assert(getStringBuffersClass(theCtx->jInterp)  \
      ->newStringBufferFunc),                      \
    getStringBuffersClass(theCtx->jInterp)         \
      ->newStringBufferFunc(theCtx)                \
```

```
  )
```

CHeader : private

```
extern StringBufferObj* newStringBufferImpl(ContextObj *aCtx);
```

CCode : default

```
StringBufferObj* newStringBufferImpl(ContextObj *aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  StringBufferObj* result =
    (StringBufferObj*)newObject(jInterp, StringBuffersTag);
  assert(result);
  asMemFile(result) = NULL;
  asBuffer(result)  = NULL;
  asBufSize(result) = 0;
  result->theCtx    = aCtx;
  result->jInterp   = jInterp;
  return result;
}
```

CHeader : public

```
typedef void (StrBufClose)(
  StringBufferObj*
);

#define strBufClose(aStrBuf)                          \
  (                                                   \
    assert(aStrBuf),                                  \
    assert(aStrBuf->jInterp),                         \
    assert(getStringBuffersClass(aStrBuf->jInterp)    \
      ->strBufCloseFunc),                             \
    getStringBuffersClass(aStrBuf->jInterp)           \
      ->strBufCloseFunc(aStrBuf)                      \
  )
```

CHeader : private

```
extern void strBufCloseImpl(StringBufferObj* aStrBuf);
```

CCode : default

```
void strBufCloseImpl(StringBufferObj* aStrBuf) {
```

```
  if (!isStringBuffer(aStrBuf)) return;

  if (asMemFile(aStrBuf)) fclose(asMemFile(aStrBuf));
  if (asBuffer(aStrBuf))  free(asBuffer(aStrBuf));
  asMemFile(aStrBuf) = NULL;
  asBuffer(aStrBuf)  = NULL;
  asBufSize(aStrBuf) = 0;
}
```

CHeader : private

```
extern void strBufReOpen(StringBufferObj* aStrBuf);
```

CCode : default

```
void strBufReOpen(StringBufferObj* aStrBuf) {
  if (!isStringBuffer(aStrBuf)) return;

  if (asMemFile(aStrBuf)) strBufCloseImpl(aStrBuf);
  asMemFile(aStrBuf) =
    open_memstream(&asBuffer(aStrBuf), &asBufSize(aStrBuf));
}
```

Test case

should create some new stringBuffers

```
  AssertPtrNotNull(jInterp);

  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);
  AssertPtrNotNull(asType(aStrBuf));
  AssertIntEquals(asTag(aStrBuf), StringBuffersTag);
  AssertPtrNull(asMemFile(aStrBuf));
  AssertPtrNull(asBuffer(aStrBuf));
  AssertIntZero(asBufSize(aStrBuf));
  AssertIntTrue(isStringBuffer(aStrBuf));
  AssertIntTrue(isAtom(aStrBuf));
  AssertIntFalse(isPair(aStrBuf));
  strBufClose(aStrBuf);
  AssertPtrNull(asMemFile(aStrBuf));
  AssertPtrNull(asBuffer(aStrBuf));
  AssertStrEquals(getCString(aStrBuf), "");
  AssertPtrNotNull(asBuffer(aStrBuf));
```

```
strBufClose(aStrBuf);
AssertPtrNull(asMemFile(aStrBuf));
AssertPtrNull(asBuffer(aStrBuf));
AssertIntZero(asBufSize(aStrBuf));
strBufPrintf(aStrBuf, "a test string");
AssertPtrNotNull(asMemFile(aStrBuf));
AssertStrEquals(getCString(aStrBuf), "a test string");
AssertPtrNotNull(asBuffer(aStrBuf));
AssertIntEquals(asBufSize(aStrBuf), 13);
strBufClose(aStrBuf); // need to release the FILE*
```

3.19.2.2 Test Suite: isStringBuffer

CHeader : public

```
#define isStringBuffer(aStrBuf)             \
  (                                         \
    (                                       \
      (aStrBuf) &&                          \
      asType(aStrBuf) &&                    \
      (asTag(aStrBuf) == StringBuffersTag)  \
    ) ?                                     \
    TRUE : FALSE                            \
  )
```

Test case

should return true if a string buffer

```
StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertIntTrue(isStringBuffer(aStrBuf));
//AssertIntTrue(symbolIs(aSym, "this is a test"));
```

Test case

should return false if not a string buffer

```
AssertIntFalse(isStringBuffer(NULL));
JObj *aObj = newObject(jInterp, BooleansTag);
AssertIntFalse(isStringBuffer(aObj));
```

3.19.2.3 Test Suite: getCString and strBufGetAsSymbol

CHeader : public

```
typedef Symbol *(GetCString)(
  StringBufferObj*
);

#define getCString(aStrBuf)                              \
  (                                                      \
    assert(aStrBuf),                                     \
    assert(aStrBuf->jInterp),                            \
    assert(getStringBuffersClass(aStrBuf->jInterp)       \
      ->getCStringFunc),                                 \
    (getStringBuffersClass(aStrBuf->jInterp)             \
      ->getCStringFunc(aStrBuf))                         \
  )
```

CHeader : private

```
extern Symbol *getCStringImpl(StringBufferObj *aStrBuf);
```

CCode : default

```
Symbol *getCStringImpl(StringBufferObj *aStrBuf) {
  if (!isStringBuffer(aStrBuf)) return NULL;
  if (!asMemFile(aStrBuf)) strBufReOpen(aStrBuf);
  fflush(asMemFile(aStrBuf));
  return asBuffer(aStrBuf);
}
```

3.19.2.4 Test Suite: strBufPrintf

CHeader : public

```
typedef Boolean (StrBufPrintf)(
  StringBufferObj*,
  Symbol*,
  ...
);

#define strBufPrintf(aStrBuf, ...)                       \
  (                                                      \
    assert(aStrBuf),                                     \
    assert(aStrBuf->jInterp),                            \
```

```
11      assert(getStringBuffersClass(aStrBuf->jInterp)  \
12        ->strBufPrintfFunc),                          \
13      (getStringBuffersClass(aStrBuf->jInterp)        \
14        ->strBufPrintfFunc(aStrBuf, __VA_ARGS__ ))    \
15    )
```

CHeader : private

```
1   extern Boolean strBufPrintfImpl(
2     StringBufferObj   *aStrBuf,
3     const char        *format,
4     ...
5   );
```

CCode : default

```
1   Boolean strBufPrintfImpl(
2     StringBufferObj   *aStrBuf,
3     const char        *format,
4     ...
5   ) {
6     if (!isStringBuffer(aStrBuf)) return FALSE;
7     if (!asMemFile(aStrBuf)) strBufReOpen(aStrBuf);

8     va_list printfArgs;
9     va_start(printfArgs, format);
10    int numChars = vfprintf(asMemFile(aStrBuf), format, printfArgs);
11    va_end(printfArgs);
12    return (0 <= numChars);
13  }
```

Test case

should printf to a sting buffer

```
StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
strBufPrintf(aStrBuf, "a test [%s]", "an inner string");
AssertStrEquals(getCString(aStrBuf), "a test [an inner string]");
```

3.19.2.5 Test Suite: stringBuffer equality

CHeader : private

```
1   Boolean stringBuffersEqual(
```

```
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean stringBuffersEqual(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "stringBuffersEqual a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA || !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (asTag(lolA)  != StringBuffersTag) return FALSE;
  if (strcmp(
    getCStringImpl((StringBufferObj*)lolA),
    getCStringImpl((StringBufferObj*)lolB)
    ) != 0) return FALSE;
  return TRUE;
}
```

Test case

should return true if stringBuffers are equal

```
  AssertIntTrue(stringBuffersEqual(jInterp, NULL, NULL, 10));
  StringBufferObj *strBufA = newStringBuffer(jInterp->rootCtx);
  strBufPrintf(strBufA, "test");
  StringBufferObj *strBufB = newStringBuffer(jInterp->rootCtx);
  strBufPrintf(strBufB, "test");
  AssertIntTrue(stringBuffersEqual(
    jInterp, (JObj*)strBufA, (JObj*)strBufB, 10));
```

Test case

should return false if stringBuffers are not equal

```
  StringBufferObj *strBufA = newStringBuffer(jInterp->rootCtx);
```

```
  strBufPrintf(strBufA, "testA");
  StringBufferObj *strBufB = newStringBuffer(jInterp->rootCtx);
  strBufPrintf(strBufB, "testB");
  AssertIntFalse(stringBuffersEqual(
    jInterp, NULL, (JObj*)strBufB, 10));
  AssertIntFalse(stringBuffersEqual(
    jInterp, (JObj*)strBufA, NULL, 10));
  AssertIntFalse(stringBuffersEqual(
    jInterp, (JObj*)strBufA, (JObj*)strBufB, 10));
```

3.19.2.6 Test Suite: printing stringBuffers

CHeader : private

```
1  extern size_t printStringBufferCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  );
```

CCode : default

```
1  Boolean printStringBufferCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  ) {
6    assert(aStrBuf);
7    assert(asTag(aStrBuf) == StringBuffersTag);

8    assert(aLoL);
9    assert(asTag(aLoL) == StringBuffersTag);
10
11   strBufPrintfImpl(aStrBuf, "%s ",
12     getCStringImpl((StringBufferObj*)aLoL));
13   return TRUE;
14 }
```

CHeader : public

```
1  typedef void (PrintLoL)(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
```

```
);

#define printLoL(aStrBuf, aLoL)                     \
  (                                                 \
    assert(aStrBuf),                                \
    assert(aStrBuf->jInterp),                       \
    assert(getStringBuffersClass(aStrBuf->jInterp)  \
      ->printLoLFunc),                              \
    (getStringBuffersClass(aStrBuf->jInterp)        \
      ->printLoLFunc(aStrBuf, aLoL, 20))            \
  )
#define printLoLTTL(aStrBuf, aLoL, ttl)             \
  (                                                 \
    assert(aStrBuf),                                \
    assert(aStrBuf->jInterp),                       \
    assert(getStringBuffersClass(aStrBuf->jInterp)  \
      ->printLoLFunc),                              \
    (getStringBuffersClass(aStrBuf->jInterp)        \
      ->printLoLFunc(aStrBuf, aLoL, ttl))           \
  )
```

CHeader : private

```
void printLoLImpl(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
void printLoLImpl(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aStrBuf);
  DEBUG(aStrBuf->jInterp,
    "printLoL(start) %p %p %zu\n", aStrBuf, aLoL, timeToLive
  );

  if (aLoL) {
    assert(asType(aLoL));
    assert(asTag(aLoL));
```

Code 3.19.2

```
    DEBUG(aStrBuf->jInterp,
      "printLoL(call) %p %p %p %uz\n",
      aStrBuf, aLoL, asType(aLoL), asTag(aLoL)
    );
    assert(asType(aLoL)->tag == asTag(aLoL));
    assert(asType(aLoL)->printFunc);

    if (isPair(aLoL)) strBufPrintf(aStrBuf, "( ");
    Boolean result =
      (asType(aLoL)->printFunc)
        (aStrBuf, aLoL, timeToLive);
    if (isPair(aLoL)) strBufPrintf(aStrBuf, ") ");
    assert(result);
  } else strBufPrintf(aStrBuf, "( ) ");

  DEBUG(aStrBuf->jInterp,
    "printLoL(done) %p %p %zu\n", aStrBuf, aLoL, timeToLive
  );
}
```

Test case

should print stringBuffers

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(getStringBuffersClass(jInterp));

  StringBufferObj* aTestStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aTestStrBuf);
  strBufPrintf(aTestStrBuf, "test string");

  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);
  printStringBufferCoAlg(
    aStrBuf, (JObj*)aTestStrBuf, 10);
  AssertStrEquals(getCString(aStrBuf), "test string ");

  strBufClose(aStrBuf);
  printLoL(aStrBuf, (JObj*)aTestStrBuf);
  AssertStrEquals(getCString(aStrBuf), "test string ");
  strBufClose(aStrBuf);
  strBufClose(aTestStrBuf);
```

3.19.2.7 Test Suite: registerStringBuffers

CHeader : public

```
typedef struct stringBuffers_class_struct {
  JClass           super;
  NewStringBuffer *newStringBufferFunc;
  StrBufClose     *strBufCloseFunc;
  GetCString      *getCStringFunc;
  StrBufPrintf    *strBufPrintfFunc;
  PrintLoL        *printLoLFunc;
} StringBuffersClass;
```

CCode : default

```
static Boolean initializeStringBuffers(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerStringBuffers(JoyLoLInterp* jInterp);
```

CCode : default

```
Boolean registerStringBuffers(JoyLoLInterp* jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  StringBuffersClass* theCoAlg  =
    joyLoLCalloc(1, StringBuffersClass);
  theCoAlg->super.name            = StringBuffersName;
  theCoAlg->super.objectSize      = sizeof(StringBufferObj);
  theCoAlg->super.initializeFunc  = initializeStringBuffers;
  theCoAlg->super.registerFunc    = registerStringBufferWords;
  theCoAlg->super.equalityFunc    = stringBuffersEqual;
  theCoAlg->super.printFunc       = printStringBufferCoAlg;
  theCoAlg->newStringBufferFunc   = newStringBufferImpl;
  theCoAlg->strBufCloseFunc       = strBufCloseImpl;
  theCoAlg->getCStringFunc        = getCStringImpl;
  theCoAlg->strBufPrintfFunc      = strBufPrintfImpl;
```

```
  theCoAlg->printLoLFunc          = printLoLImpl;
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // sanity check...
  assert(tag == StringBuffersTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the stringBuffers coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerStringBuffers
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getStringBuffersClass(jInterp));
  StringBuffersClass *coAlg =
    getStringBuffersClass(jInterp);
  AssertIntTrue(registerStringBuffers(jInterp));
  AssertPtrNotNull(getStringBuffersClass(jInterp));
  AssertPtrEquals(getStringBuffersClass(jInterp), coAlg);
  AssertIntEquals(
    getStringBuffersClass(jInterp)->super.objectSize,
    sizeof(StringBufferObj)
  )
```

3.19.3 Words

CCode : default

```
static void printLoLAP(ContextObj* aCtx) {
  assert(aCtx);
  JoyLoLInterp *jInterp = aCtx->jInterp;
  assert(jInterp);

  popCtxDataInto(aCtx, lol);
  popCtxDataInto(aCtx, depth);
```

```
  if (!isNatural(depth)) {
    raiseExceptionMsg(aCtx,
      "printLoL expected a natural as depth");
    return;
  }
  size_t depthInt = SIZE_MAX;
  double depthDbl = asNaturalDbl(jInterp, depth);
  if (depthDbl < ((double)depthInt)) {
    depthInt = ((size_t)depthDbl);
  }

  StringBufferObj* aStrBuf = newStringBufferImpl(aCtx);

  printLoLImpl(aStrBuf, lol, depthInt);
  JObj* result =
    newSymbol(jInterp, getCString(aStrBuf), "strBuf", 0);
  pushCtxData(aCtx, result);

  strBufClose(aStrBuf);
}
```

CHeader : private

```
extern Boolean registerStringBufferWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerStringBufferWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  extendJoyLoLInRoot(jInterp, "printLoL", "", printLoLAP, "");
  return TRUE;
}
```

3.19.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
```

```
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_stringBuffers_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_stringBuffers [] = {
  {"gitVersion", lua_stringBuffers_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_stringBuffers (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerStringBuffers(jInterp);
  luaL_newlib(lstate, lua_stringBuffers);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedStringBuffers` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedStringBuffers(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedStringBuffers(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_stringBuffers(lstate);
  lua_setfield(lstate, -2, "joylol.stringBuffers");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.19.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/cFunctions.h>
#include <joylol/symbols.h>
#include <joylol/naturals.h>
#include <joylol/stringBuffers.h>
#include <joylol/texts.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/stringBuffers-private.h>
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
  requireLuaModule(lstate, "joylol.pairs");
  requireLuaModule(lstate, "joylol.booleans");
  requireLuaModule(lstate, "joylol.cFunctions");
  requireLuaModule(lstate, "joylol.texts");
  requireLuaModule(lstate, "joylol.contexts");
  requireLuaModule(lstate, "joylol.dictionaries");
```

```
requireLuaModule(lstate, "joylol.dictNodes");
requireStaticallyLinkedStringBuffers(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.20 Symbols

3.20.1 Goals

The symbols extension extends the core JoyLoL by providing support for constant strings, aka 'symbols'.

3.20.2 Code

CHeader : public

```
typedef struct symbol_object_struct {
  JObj super;
  Symbol *sym;
  Symbol *file;
  size_t  line;
} SymbolObj;

#define LOOKUP_SYMBOL_FLAG 0x8L
#define asSymbol(aObj) (((SymbolObj*)(aObj))->sym)
#define asFile(aObj)   (((SymbolObj*)(aObj))->file)
#define asLine(aObj)   (((SymbolObj*)(aObj))->line)
```

3.20.2.1 Test Suite: newString

3.20.2.2 Test Suite: newSymbol

CHeader : public

```
typedef JObj *(NewSymbol)(
  JoyLoLInterp *jInterp,
  Symbol       *theSymbol,
  Symbol       *fileName,
  size_t        line,
  Boolean       isALookupSymbol
);

#define newSymbol(jInterp, aSymbol, fileName, line)             \
  (                                                             \
    assert(getSymbolsClass(jInterp)                             \
      ->newSymbolFunc),                                         \
    (getSymbolsClass(jInterp)                                   \
```

```
      ->newSymbolFunc(jInterp, aSymbol, fileName, line, FALSE)) \
  )
```

CHeader : private

```
extern JObj *newSymbolImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aSymbol,
  Symbol       *fileName,
  size_t        line,
  Boolean       isALookupSymbol
);
```

CCode : default

```
JObj* newSymbolImpl(
  JoyLoLInterp *jInterp,
  Symbol       *aSymbol,
  Symbol       *fileName,
  size_t        line,
  Boolean       isALookupSymbol
) {
  assert(aSymbol);
  assert(jInterp);
  if (!fileName) fileName = "unknown(symbol)";
  JObj* result = newObject(jInterp, SymbolsTag);
  assert(result);
  asSymbol(result) = strdup(aSymbol);
  asFile(result)   = strdup(fileName);
  asLine(result)   = line;
  if (isALookupSymbol) {
    result->flags |= LOOKUP_SYMBOL_FLAG;
  }
  return result;
}
```

Test case

should create some new symbols

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);

  const char* testStr = "test string";
```

```
JObj* aNewSymbol = newSymbol(jInterp, testStr, "testStr", 11);
AssertPtrNotNull(aNewSymbol);
AssertPtrNotNull(asType(aNewSymbol));
AssertIntEquals(asTag(aNewSymbol), SymbolsTag);
AssertPtrNotNull(asSymbol(aNewSymbol));
AssertPtrNotEquals(asSymbol(aNewSymbol), testStr);
AssertIntEquals(strcmp(asSymbol(aNewSymbol), testStr), 0);
AssertStrEquals(asFile(aNewSymbol), "testStr");
AssertIntEquals(asLine(aNewSymbol), 11);
AssertIntTrue(isSymbol(aNewSymbol));
AssertIntTrue(isAtom(aNewSymbol));
AssertIntFalse(isPair(aNewSymbol));

aNewSymbol = newSymbol(jInterp, testStr, NULL, 0);
AssertPtrNotNull(aNewSymbol);
AssertStrEquals(asFile(aNewSymbol), "unknown(symbol)");
```

3.20.2.3 Test Suite: isSymbol and symbolIs

CHeader : public

```
#define isSymbol(aLoL)             \
  (                                \
    (                              \
      (aLoL) &&                    \
      asType(aLoL) &&              \
      (asTag(aLoL) == SymbolsTag)  \
    ) ?                            \
      TRUE :                       \
      FALSE                        \
  )
#define isLookupSymbol(aLoL)                  \
  (                                           \
    (                                         \
      (aLoL) &&                               \
      asType(aLoL) &&                         \
      (asTag(aLoL) == SymbolsTag)             \
      isFlagSet(aLoL, LOOKUP_SYMBOL_FLAG) &&  \
    ) ?                                       \
      TRUE :                                  \
      FALSE                                   \
  )
```

Code 3.20.2

CHeader : public

```
typedef Boolean (SymbolIs)(
  JObj *aLoL,
  Symbol   *aSymbol
);

#define symbolIs(jInterp, aLoL, aSymbol)  \
  (                                       \
    assert(getSymbolsClass(jInterp)       \
      ->symbolIsFunc),                    \
    (getSymbolsClass(jInterp)             \
      ->symbolIsFunc(aLoL, aSymbol))      \
  )
```

CHeader : private

```
extern Boolean symbolIsImpl(
  JObj *aLoL,
  Symbol *aSymbol
);
```

CCode : default

```
Boolean symbolIsImpl(
  JObj *aLoL,
  Symbol *aSymbol
) {
  if (isSymbol(aLoL) &&
      (strcmp(asSymbol(aLoL), aSymbol) == 0)) {
    return TRUE;
  }
  return FALSE;
}
```

Test case

should return true if a symbol

```
  JObj *aSym = newSymbol(jInterp, "this is a test", NULL, 0);
  AssertIntTrue(isSymbol(aSym));
  AssertIntTrue(symbolIs(jInterp, aSym, "this is a test"));
```

Test case

should return false if not a symbol

```
  AssertIntFalse(isSymbol(NULL));
  AssertIntFalse(symbolIs(jInterp, NULL, "this is NOT a test"));
  JObj *aObj = newObject(jInterp, BooleansTag);
  AssertIntFalse(isSymbol(aObj));
  AssertIntFalse(symbolIs(jInterp, aObj, "this is NOT a test"));
  JObj *aSym = newSymbol(jInterp, "this is a test", NULL, 0);
  AssertIntFalse(symbolIs(jInterp, aSym, "this is NOT a test"));
```

3.20.2.4 Test Suite: symbol equality

CHeader : private

```
Boolean symbolsEqual(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean symbolsEqual(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "symbolsEqual a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA || !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA) != SymbolsTag) return FALSE;
  if (strcmp(asSymbol(lolA), asSymbol(lolB)) != 0) return FALSE;
  return TRUE;
}
```

Test case

should return true if symbols are equal

```
  AssertIntTrue(symbolsEqual(jInterp, NULL, NULL, 10));
  JObj *symA = newSymbol(jInterp, "the same text", NULL, 0);
  JObj *symB = newSymbol(jInterp, "the same text", NULL, 0);
  AssertIntTrue(symbolsEqual(jInterp, symA, symB, 10));
```

Test case

should return false if symbols are not equal

```
  JObj *symA = newSymbol(jInterp, "text A", NULL, 0);
  JObj *symB = newSymbol(jInterp, "text B", NULL, 0);
  AssertIntFalse(symbolsEqual(jInterp, NULL, symB, 10));
  AssertIntFalse(symbolsEqual(jInterp, symA, NULL, 10));
  AssertIntFalse(symbolsEqual(jInterp, symA, symB, 10));
```

3.20.2.5 Test Suite: printing symbols

CHeader : private

```
1  extern size_t printSymbolCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  );
```

CCode : default

```
1  Boolean printSymbolCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  ) {
6    assert(aLoL);
7    assert(asTag(aLoL) == SymbolsTag);
8
9    if (isFlagSet(aLoL, LOOKUP_SYMBOL_FLAG)) {
10     ContextObj  *aCtx    = aStrBuf->theCtx;
11     assert(aCtx);
12     DictObj     *theDict = aCtx->dict;
13     assert(theDict);
```

```
14      DictNodeObj *assoc   =
15        getSymbolEntry(theDict, asSymbol(aLoL));
16      if (assoc && assoc->value) {
17        printLoLTTL(aStrBuf, assoc->value, timeToLive-1);
18        return TRUE;
19      }
20    }
21
22    if (strchr(asSymbol(aLoL), ' ')) {
23      strBufPrintf(aStrBuf, "\"%s\" ", asSymbol(aLoL));
24    } else {
25      strBufPrintf(aStrBuf, "%s ", asSymbol(aLoL));
26    }
27    return TRUE;
28  }
```

Test case

should print symbols

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[SymbolsTag]);

  JObj* aNewSymbol = newSymbol(jInterp, "test string", NULL, 0);
  AssertPtrNotNull(aNewSymbol);

  StringBufferObj* aStrBuf = newStringBuffer(jInterp->rootCtx);

  printLoL(aStrBuf, aNewSymbol);
  AssertStrEquals(getCString(aStrBuf),
    "\"test string\" ");
  strBufClose(aStrBuf);
```

3.20.2.6 Test Suite: registerSymbols

CHeader : public

```
1   typedef struct symbols_class_struct {
2     JClass  super;
3     NewSymbol  *newSymbolFunc;
4     SymbolIs   *symbolIsFunc;
```

```
} SymbolsClass;
```

CCode : default

```
static Boolean initializeSymbols(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerSymbols(JoyLoLInterp* jInterp);
```

CCode : default

```
Boolean registerSymbols(JoyLoLInterp* jInterp) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  SymbolsClass* theCoAlg = joyLoLCalloc(1, SymbolsClass);
  theCoAlg->super.name           = SymbolsName;
  theCoAlg->super.objectSize     = sizeof(SymbolObj);
  theCoAlg->super.initializeFunc = initializeSymbols;
  theCoAlg->super.registerFunc   = registerSymbolWords;
  theCoAlg->super.equalityFunc   = symbolsEqual;
  theCoAlg->super.printFunc      = printSymbolCoAlg;
  theCoAlg->newSymbolFunc        = newSymbolImpl;
  theCoAlg->symbolIsFunc         = symbolIsImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // sanity check...
  assert(tag == SymbolsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Symbols coAlg

```
// CTestsSetup has already created a jInterp
// and run registerSymbols
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getSymbolsClass(jInterp));
SymbolsClass *coAlg = getSymbolsClass(jInterp);
AssertIntTrue(registerSymbols(jInterp));
AssertPtrNotNull(getSymbolsClass(jInterp));
AssertPtrEquals(getSymbolsClass(jInterp), coAlg);
AssertIntEquals(
  getSymbolsClass(jInterp)->super.objectSize,
  sizeof(SymbolObj)
)
```

3.20.3 Words

CCode : default

```
1  static void isSymbolAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top);
6    JObj* result = NULL;
7    if (isSymbol(top))
8      result = newBoolean(jInterp, TRUE);
9    else
10     result = newBoolean(jInterp, FALSE);
11   pushCtxData(aCtx, result);
12 }
```

CCode : default

```
1  static void equalSymbolAP(ContextObj* aCtx) {
2    assert(aCtx);
3    JoyLoLInterp *jInterp = aCtx->jInterp;
4    assert(jInterp);
5    popCtxDataInto(aCtx, top1);
6    popCtxDataInto(aCtx, top2);
7    size_t result = FALSE;
```

```
  if (isSymbol(top1) && isSymbol(top2)) {
    if (strcmp(asSymbol(top1), asSymbol(top2)) == 0) result = TRUE;
  }
  DEBUG(jInterp, "equalSymbol: %zu\n", result);
  pushCtxData(aCtx, newBoolean(jInterp, result));
}
```

CHeader : private

```
extern Boolean registerSymbolWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerSymbolWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  assert(jInterp);
  ContextObj *rootCtx = jInterp->rootCtx;
  assert(rootCtx);
  DictObj *dict = rootCtx->dict;
  assert(dict);

  DictNodeObj* newLine = getSymbolEntry(dict, "newLine");
  newLine->value = newSymbolImpl(jInterp, "\n", "newLine", 0, TRUE);

  extendJoyLoLInRoot(jInterp, "isSymbol", "", isSymbolAP,    "");
  extendJoyLoLInRoot(jInterp, "=Sym",     "", equalSymbolAP, "");

  return TRUE;
}
```

3.20.4 Lua functions

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
```

```
  { "subject",          "updated textadept lexer for JoyLoL"},
  { "notes",            ""},
  { NULL,                 NULL}
};
```

CCode : default

```
static int lua_symbols_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_symbols [] = {
  {"gitVersion", lua_symbols_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_symbols (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerSymbols(jInterp);
  luaL_newlib(lstate, lua_symbols);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedSymbols` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedSymbols(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedSymbols(
  lua_State *lstate
) {
```

```
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_symbols(lstate);
  lua_setfield(lstate, -2, "joylol.symbols");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.20.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/booleans.h>
#include <joylol/cFunctions.h>
#include <joylol/stringBuffers.h>
#include <joylol/symbols.h>
#include <joylol/texts.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/dictionaries.h>
#include <joylol/dictNodes.h>
#include <joylol/symbols-private.h>
```

```
  addJoyLoLLuaPath(lstate);
  requireStaticallyLinkedJInterps(lstate);
  requireLuaModule(lstate, "joylol.assertions");
  requireLuaModule(lstate, "joylol.pairs");
  requireLuaModule(lstate, "joylol.booleans");
  requireLuaModule(lstate, "joylol.cFunctions");
  requireLuaModule(lstate, "joylol.texts");
  requireLuaModule(lstate, "joylol.contexts");
  requireLuaModule(lstate, "joylol.dictionaries");
  requireLuaModule(lstate, "joylol.dictNodes");
```

```
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedSymbols(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.21 Templates

3.21.1 Goals

A template

3.21.2 Code

3.21.2.1 Test Suite: registerTemplates

CHeader : public

```
typedef struct templates_class_struct {
  JClass super;
} TemplatesClass;
```

CCode : default

```
static Boolean initializeTemplates(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerTemplates(JoyLoLInterp *jInterp);
```

CCode : default

```
Boolean registerTemplates(JoyLoLInterp *jInterp) {
  assert(jInterp);

  TemplatesClass* theCoAlg =
    joyLoLCalloc(1, TemplatesClass);
  theCoAlg->super.name           = TemplatesName;
  theCoAlg->super.objectSize     = sizeof(JObj);
  theCoAlg->super.initializeFunc = initializeTemplates;
  theCoAlg->super.registerFunc   = registerTemplateWords;
  theCoAlg->super.equalityFunc   = NULL;
  theCoAlg->super.printFunc      = NULL;
```

```
  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == TemplatesTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Templates coAlg

```
  // CTestsSetup has already created a jInterp
  // and run registerTemplates

  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs);
  AssertPtrNotNull(getTemplatesClass(jInterp));
  TemplatesClass *coAlg =
    getTemplatesClass(jInterp);
  AssertIntTrue(registerTemplates(jInterp));
  AssertPtrNotNull(getTemplatesClass(jInterp));
  AssertPtrEquals(getTemplatesClass(jInterp), coAlg);
  AssertIntEquals(
    getTemplatesClass(jInterp)->super.objectSize,
    sizeof(JObj)
  )
```

3.21.3 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
```

```
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_templates_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_templates [] = {
  {"gitVersion", lua_templates_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_templates (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerTemplates(jInterp);
  luaL_newlib(lstate, lua_templates);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedTemplates` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedTemplates(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedTemplates(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
```

```
  luaopen_joylol_templates(lstate);
  lua_setfield(lstate, -2, "joylol.templates");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.21.4 JoyLol words

CHeader : private

```
extern Boolean registerTemplateWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerTemplateWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
) {
  return TRUE;
}
```

3.21.5 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/templates.h>
#include <joylol/templates-private.h>
// dictionary
// printer
```

```
  addJoyLoLLuaPath(lstate);
```

```
requireStaticallyLinkedJInterps(lstate);
requireStaticallyLinkedTemplates(lstate);
getJoyLoLInterpInto(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

3.22 Texts

3.22.1 Goals

A text

3.22.2 Strings

CHeader : public

```
typedef struct text_object_struct TextObj;
```

CCode : default

```
// Texts are a collection of characters, which are used by the Parser
// to extract successive symbols.
//
// Texts are created on one of three backing suppliers of characters:
// 1. a single string
// 2. a NULL terminated array of strings
// 3. an external file
// 4. a readline interaction with a user
//
// In all four cases, the Parser's nextSymbol method requests successive
// **lines** of characters (deliminated by new-line-characters).
//
// It is critical, for correct interaction with the user via readline,
// that the initial line is NOT obtained until actually requested by
// the parser's nextSymbol method.
//
// It is also critical that once completed, none of the sources, get
// asked for subsequent lines.
//
// When the text has been completed, the nextLine function ensures
// that aText->curLine is NULL.
```

3.22.2.1 Test Suite: texts from a string

CHeader : private

```
extern void nextLineFromString(TextObj* aText);
```

CCode : default

```
1  void nextLineFromString(TextObj* aText) {
2    assert(aText);
3    assert(aText->jInterp);
4    DEBUG(aText->jInterp, "->nextLineFromString [%s]{%p:%p}\n",
5      aText->curLine, aText->curChar, aText->lastChar);
6    if (!aText) return;  // there is nothing we can do!
7    if (!aText->curLine) return; // there is nothing we can do!
8
9    // there is no next line so... we have already reached the end of the text
10   aText->completed  = TRUE;
11   aText->curLine    = NULL;
12   aText->curChar    = NULL;
13   aText->lastChar   = NULL;
14 }
```

CHeader : public

```
1  typedef TextObj *(CreateTextFromString)(
2    JoyLoLInterp *jInterp,
3    Symbol       *aString
4  );
5
6  #define createTextFromString(jInterp, aString)      \
7    (                                                 \
8      assert(getTextsClass(jInterp)                   \
9        ->createTextFromStringFunc),                  \
10     (getTextsClass(jInterp)                         \
11       ->createTextFromStringFunc(jInterp, aString)) \
12   )
```

CHeader : private

```
1  extern TextObj* createTextFromStringImpl(
2    JoyLoLInterp* jInterp,
3    Symbol* aString
4  );
```

CCode : default

```
1  TextObj* createTextFromStringImpl(
2    JoyLoLInterp *jInterp,
3    Symbol* aString
4  ) {
5    assert(jInterp);
6    assert(aString);
```

```
7   TextObj* aText = (TextObj*)newObject(jInterp, TextsTag);
8   assert(aText);
9   //
10  // array of strings specific initializations
11  //
12  aText->curLine   = aString;
13  aText->curChar   = aText->curLine;
14  aText->lastChar  = aText->curChar + strlen(aText->curLine);
15  aText->nextLine  = nextLineFromString;
16  //
17  // general initializations
18  //
19  aText->jInterp   = jInterp;
20  aText->completed = FALSE;
21  aText->sym       = NULL;
22  //
23  aText->inputFile = NULL;
24  //
25  aText->newPrompt       = NULL;
26  aText->continuePrompt  = NULL;
27  aText->curPrompt       = NULL;
28  //
29  aText->textLines  = NULL;
30  aText->fileName   = strdup(aString);
31  aText->curLineNum = 0;
32
33  return aText;
34 }
```

Test case

should create Text From A String

```
AssertPtrNotNull(jInterp);

char* aString = "this (is a test) (of strings)";
TextObj* aText = createTextFromString(jInterp, aString);
AssertPtrNotNull(aText);

AssertPtrNull(aText->textLines);
AssertIntEquals(aText->curLineNum, 0);
AssertPtrEquals(aText->curLine, aString);
```

```
AssertPtrEquals(aText->curChar, aString);
AssertPtrEquals(aText->lastChar,
  aString + strlen(aText->curLine)
);

AssertPtrNotNull(aText->nextLine);
aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, NULL);
AssertPtrEquals((void*)aText->curChar, NULL);
AssertIntEquals((aText->lastChar - aText->curChar), 0);

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, NULL);
AssertPtrEquals((void*)aText->curChar, NULL);
AssertIntEquals((aText->lastChar - aText->curChar), 0);
```

Test case

nextSymbol should get next symbol

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromString(jInterp, "this (is a test) (of strings)");
AssertPtrNotNull(aText);

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "this");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "(");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "a");

nextSymbol(aText);
```

```
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "test");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ")");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "(");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "of");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "strings");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ")");

nextSymbol(aText);
AssertPtrNull(aText->sym);
```

Test case

nextSymbol should deal with quotes

```
AssertPtrNotNull(jInterp);

TextObj* aText =
  createTextFromString(jInterp, "this \" is a test \" 'of strings'");
AssertPtrNotNull(aText);

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "this");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), " is a test ");
```

```
  nextSymbol(aText);
  AssertPtrNotNull(aText->sym);
  AssertStrEquals(asSymbol(aText->sym), "of strings");

  nextSymbol(aText);
  AssertPtrNull(aText->sym);
```

3.22.2.2 Test Suite: texts from an array of strings

CHeader : private

```
1  extern void nextLineFromArray(TextObj* aText);
```

CCode : default

```
1  void nextLineFromArray(TextObj* aText) {
2    assert(aText);
3    assert(aText->jInterp);
4    DEBUG(aText->jInterp, "->nextLineFromArray %s\n", "");
5    if (!aText) return;  // there is nothing we can do!
6    if (!aText->textLines) return; // there is nothing we can do!
7
8    if (!aText->textLines[aText->curLineNum]) {
9      // we have already reached the end of the text
10     aText->completed  = TRUE;
11     aText->curLine    = NULL;
12     aText->curChar    = NULL;
13     aText->lastChar   = NULL;
14     return;
15   }
16
17   aText->curLine  = aText->textLines[aText->curLineNum];
18
19   aText->curChar  = aText->curLine;
20
21   if (aText->curLine) {
22     aText->lastChar = aText->curChar + strlen(aText->curLine);
23   } else aText->lastChar = aText->curChar;
24
25   aText->curLineNum++;
26 }
```

CHeader : public

```
typedef TextObj *(CreateTextFromArrayOfStrings)(
  JoyLoLInterp *jInterp,
  Symbol       *someTextLines[]
);

#define createTextFromArrayOfStrings(jInterp, textLines)      \
  (                                                           \
    assert(getTextsClass(jInterp)                             \
      ->createTextFromArrayOfStringsFunc),                    \
    (getTextsClass(jInterp)                                   \
      ->createTextFromArrayOfStringsFunc(jInterp, textLines)) \
  )
```

CHeader : private

```
extern TextObj* createTextFromArrayOfStringsImpl(
  JoyLoLInterp* jInterp,
  Symbol* someTextLines[]
);
```

CCode : default

```
TextObj* createTextFromArrayOfStringsImpl(
  JoyLoLInterp *jInterp,
  Symbol* someTextLines[]
) {
  assert(jInterp);
  assert(someTextLines);

  TextObj* aText = (TextObj*)newObject(jInterp, TextsTag);
  assert(aText);
  //
  // array of strings specific initializations
  //
  aText->textLines  = someTextLines;
  aText->curLineNum = 0;
  aText->nextLine   = nextLineFromArray;
  //
  // general initializations
  //
  aText->jInterp    = jInterp;
  aText->completed  = FALSE;
  aText->sym        = NULL;
  aText->curLine    = NULL;
```

```
  aText->curChar    = NULL;
  aText->lastChar   = NULL;
  //
  aText->inputFile  = NULL;
  aText->fileName   = strdup("arrayOfStrings");
  //
  aText->newPrompt       = NULL;
  aText->continuePrompt  = NULL;
  aText->curPrompt       = NULL;

  return aText;
}
```

```
static Symbol* someLines[] = {
  "  This is a first line  ",
  "  This is a second line   ",
  "",
  " ([<{ }>]) ",
  "           ",
  "  This is the last line  ",
  NULL
};
```

Test case

should create Text From Array Of Strings

```
  AssertPtrNotNull(jInterp);

  TextObj* aText = createTextFromArrayOfStrings(jInterp, someLines);
  AssertPtrNotNull(aText);

  AssertPtrEquals((void*)aText->textLines, (void*)someLines);
  AssertIntEquals(aText->curLineNum, 0);
  AssertPtrNull(aText->curLine);
  AssertPtrNull(aText->curChar);
  AssertPtrNull(aText->lastChar);

  AssertPtrNotNull(aText->nextLine);
  aText->nextLine(aText);

  AssertPtrEquals((void*)aText->curLine, (void*)someLines[0]);
```

```
AssertPtrEquals((void*)aText->curChar, (void*)someLines[0]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[0]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, (void*)someLines[1]);
AssertPtrEquals((void*)aText->curChar, (void*)someLines[1]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[1]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, (void*)someLines[2]);
AssertPtrEquals((void*)aText->curChar, (void*)someLines[2]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[2]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, (void*)someLines[3]);
AssertPtrEquals((void*)aText->curChar, (void*)someLines[3]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[3]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, (void*)someLines[4]);
AssertPtrEquals((void*)aText->curChar, (void*)someLines[4]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[4]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, (void*)someLines[5]);
AssertPtrEquals((void*)aText->curChar, (void*)someLines[5]);
AssertIntEquals((aText->lastChar - aText->curChar),
                      strlen(someLines[5]));

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, NULL);
AssertPtrEquals((void*)aText->curChar, NULL);
AssertIntEquals((aText->lastChar - aText->curChar), 0);

aText->nextLine(aText);
AssertPtrEquals((void*)aText->curLine, NULL);
AssertPtrEquals((void*)aText->curChar, NULL);
AssertIntEquals((aText->lastChar - aText->curChar), 0);
```

Test case

nextSymbol should get next symbol

```
AssertPtrNotNull(jInterp);

TextObj* aText = createTextFromArrayOfStrings(jInterp, someLines);
AssertPtrNotNull(aText);

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "a");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "first");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "a");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "second");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");

// " ([<{ }>]) "
```

```
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "(");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "[");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "<");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "{");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "}");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ">");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "]");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ")");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "the");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "last");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");

nextSymbol(aText);
AssertPtrNull(aText->sym);
```

3.22.3 Strings

3.22.3.1 Test Suite: texts from files

CHeader : private

```
1  extern void nextLineFromFile(TextObj* aText);
```

CCode : default

```
1  void nextLineFromFile(TextObj* aText) {
2    assert(aText);
3    assert(aText->jInterp);
4    DEBUG(aText->jInterp, "->nextLineFromFile %s\n", "");
5    if (!aText) return; // there is nothing we can do!
6    if (!aText->inputFile) return; // there is nothing we can do!
7
8    // getline returns alloc'ed memory so we need to free it here.
9    if (aText->curLine) free((void*)aText->curLine);
10
11   aText->curLine    = NULL;
12   aText->curChar    = NULL;
13   aText->lastChar   = NULL;
14
15   char *linePtr    = NULL;
16   size_t n         = 0;
17   if (!getline(&linePtr, &n, aText->inputFile) && n < 1) {
18     DEBUG(aText->jInterp, "<-nextLineFromFile %s\n", "getline returned 0");
19     aText->completed = TRUE;
20     return;
21   }
22
23   if (feof(aText->inputFile)) {
24     // we have already reached the end of the text;
25     DEBUG(aText->jInterp, "<-nextLineFromFile %s\n", "feof returned file end");
26     aText->completed = TRUE;
27     aText->curLine   = NULL;
28     aText->curChar   = NULL;
29     aText->lastChar  = NULL;
30     return;
31   }
32
```

```
33    aText->curLine    = linePtr;
34    aText->curLineNum++;
35    aText->curChar    = aText->curLine;
36    aText->lastChar   = aText->curChar + strlen(aText->curLine);
37    DEBUG(aText->jInterp, "<-nextLineFromFile [%s]\n", aText->curLine);
38  }
```

CHeader : public

```
1   typedef TextObj *(CreateTextFromInputFile)(
2     JoyLoLInterp *jInterp,
3     FILE         *anInputFile,
4     Symbol       *aFileName
5   );
6
7   #define createTextFromInputFile(jInterp, aFile, aFileName)      \
8     (                                                             \
9       assert(getTextsClass(jInterp)                               \
10        ->createTextFromInputFileFunc),                           \
11      (getTextsClass(jInterp)                                     \
12        ->createTextFromInputFileFunc(jInterp, aFile, aFileName)) \
13    )
```

CHeader : private

```
1   extern TextObj* createTextFromInputFileImpl(
2     JoyLoLInterp *jInterp,
3     FILE         *anInputFile,
4     Symbol       *aFileName
5   );
```

CCode : default

```
1   TextObj* createTextFromInputFileImpl(
2     JoyLoLInterp *jInterp,
3     FILE         *anInputFile,
4     Symbol       *aFileName
5   ) {
6     assert(jInterp);
7     assert(anInputFile);
8     if (!aFileName) aFileName = "unknown(file)";

9     TextObj* aText = (TextObj*)newObject(jInterp, TextsTag);
10    assert(aText);
11    //
```

```
12    // external file specific initializations
13    //
14    aText->inputFile  = anInputFile;
15    aText->nextLine   = nextLineFromFile;
16    //
17    // general initializations
18    //
19    aText->jInterp    = jInterp;
20    aText->completed  = FALSE;
21    aText->sym        = NULL;
22    aText->curLine    = NULL;
23    aText->curChar    = NULL;
24    aText->lastChar   = NULL;
25    //
26    aText->textLines  = NULL;
27    aText->fileName   = strdup(aFileName);
28    aText->curLineNum = 0;
29    //
30    aText->newPrompt      = NULL;
31    aText->continuePrompt = NULL;
32    aText->curPrompt      = NULL;
33    return aText;
34  }
```

Test case

should create Text From Input File

```
  AssertPtrNotNull(jInterp);

  FILE* inputFile = fopen("doc/testSomeLines.txt", "r");
  AssertPtrNotNull(inputFile);

  TextObj* aText = createTextFromInputFile(jInterp, inputFile, "testSomeLines.txt");
  AssertPtrNotNull(aText);
  AssertPtrNotNull(aText->inputFile);
  AssertPtrNull(aText->curLine);
  AssertPtrNull(aText->curChar);
  AssertPtrNull(aText->lastChar);
  aText->nextLine(aText);
  AssertStrEquals(aText->curLine, "  This is a first line  \n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
  AssertIntEquals((aText->lastChar - aText->curChar),
```

```
                        strlen(someLines[0]) + 1);
  aText->nextLine(aText);
  AssertStrEquals(aText->curLine, "  This is a second line   \n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
  AssertIntEquals((aText->lastChar - aText->curChar),
                        strlen(someLines[1]) + 1);
  aText->nextLine(aText);
  AssertStrEquals(aText->curLine, "\n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
  AssertIntEquals((aText->lastChar - aText->curChar),
                        strlen(someLines[2]) + 1);
  aText->nextLine(aText);
  AssertStrEquals(aText->curLine, " ([<{ }>]) \n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
  AssertIntEquals((aText->lastChar - aText->curChar),
                        strlen(someLines[3]) + 1);
  // these do not match litterally since file does not seem to have tabs
  aText->nextLine(aText);
//  AssertStrEquals(aText->curLine, "            \n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
//  AssertIntEquals((aText->lastChar - aText->curChar),
//                        strlen(someLines[4]) + 1);
  aText->nextLine(aText);
  AssertStrEquals(aText->curLine, "  This is the last line  \n");
  AssertPtrEquals((void*)aText->curChar, (void*)aText->curLine);
  AssertIntEquals((aText->lastChar - aText->curChar),
                        strlen(someLines[5]) + 1);
  // the file has a couple more blank lines at the end
  aText->nextLine(aText);
  aText->nextLine(aText);
  aText->nextLine(aText);
  aText->nextLine(aText);
  aText->nextLine(aText);
  aText->nextLine(aText);
  aText->nextLine(aText);
  AssertPtrEquals((void*)aText->curLine, NULL);
  AssertPtrEquals((void*)aText->curChar, NULL);
  AssertIntEquals((aText->lastChar - aText->curChar), 0);
  fclose(inputFile);
```

Test case

should get symbols using nextSymbol

```
AssertPtrNotNull(jInterp);

FILE* inputFile = fopen("doc/testSomeLines.txt", "r");
AssertPtrNotNull(inputFile);

TextObj* aText = createTextFromInputFile(jInterp, inputFile, "testSomeLines.txt");
AssertPtrNotNull(aText);

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "a");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "first");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "a");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "second");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");
```

```
// " ([<{ }>]) "
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "(");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "[");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "<");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "{");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "}");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ">");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "]");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), ")");

nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "This");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "is");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "the");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "last");
nextSymbol(aText);
AssertPtrNotNull(aText->sym);
AssertStrEquals(asSymbol(aText->sym), "line");

nextSymbol(aText);
```

```
  AssertPtrNull(aText->sym);
  fclose(inputFile);
```

3.22.4 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_texts_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_texts [] = {
  {"gitVersion", lua_texts_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_texts (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerTexts(jInterp);
  luaL_newlib(lstate, lua_texts);
  return 1;
}
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedTexts` does just this.

CHeader : public

```
Boolean requireStaticallyLinkedTexts(
  lua_State *lstate
);
```

CCode : default

```
Boolean requireStaticallyLinkedTexts(
  lua_State *lstate
) {
  lua_getglobal(lstate, "package");
  lua_getfield(lstate, -1, "loaded");
  luaopen_joylol_texts(lstate);
  lua_setfield(lstate, -2, "joylol.texts");
  lua_setfield(lstate, -2, "loaded");
  lua_pop(lstate, 1);
  return TRUE;
}
```

3.22.5 JoyLoL words

CHeader : private

```
extern Boolean registerTextWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
);
```

CCode : default

```
Boolean registerTextWords(
  JoyLoLInterp *jInterp,
  JClass       *theCoAlg
){
  return TRUE;
}
```

3.22.6 Code

CHeader : public

```
struct text_object_struct;

typedef void (NextLineFunc)(struct text_object_struct*);

typedef struct text_object_struct {
  JObj super;
  //
  // fields used by all backing types
  //
  size_t        completed;
  Symbol       *fileName;
  size_t        curLineNum;
  JObj         *sym;
  Symbol       *curLine;
  Symbol       *curChar;
  Symbol       *lastChar;
  NextLineFunc *nextLine;
  JoyLoLInterp *jInterp;
  //
  // array of strings specific fields
  //
  Symbol** textLines;
  //
  // external file specific fields
  //
  FILE* inputFile;
  //
  // readline specific fields
  //
  Symbol*  newPrompt;
  Symbol*  continuePrompt;
  Symbol*  curPrompt;
  DictNodeObj* curNode;
  Symbol*  curCompletionText;
  size_t   curCompletionLen;
} TextObj;
```

CCode : default

```
// Texts are a collection of characters, which are used by the Parser
// to extract successive symbols.
```

```
//
// Texts are created on one of three backing suppliers of characters:
// 1. a fixed array of strings
// 2. an external file
// 3. a readline interaction with a user
//
// In all three cases, the Parser's nextSymbol method requests successive
// **lines** of characters (deliminated by new-line-characters).
//
// It is critical, for correct interaction with the user via readline,
// that the initial line is NOT obtained until actually requested by
// the parser's nextSymbol method.
//
// It is also critical that once completed, none of the sources, get
// asked for subsequent lines.
//
// When the text has been completed, the nextLine function ensures
// that aText->curLine is NULL.
```

CHeader : public

```
typedef void (FreeText)(
  TextObj *aText
);

#define freeText(aText)                    \
  (                                        \
    assert(aText),                         \
    assert(getTextsClass(aText->jInterp)   \
      ->freeTextFunc),                     \
    (getTextsClass(aText->jInterp)         \
      ->freeTextFunc(aText))               \
  )
```

CHeader : private

```
extern void freeTextImpl(
  TextObj* aText
);
```

CCode : default

```
void freeTextImpl(
  TextObj* aText
) {
```

```
  if (!aText) return;

  //free(aText);
}
```

CHeader : private

```
extern void nextLineExit(TextObj* aText);
```

CCode : default

```
void nextLineExit(TextObj* aText) {
  if (!aText) return; // there is nothing we can do!
  aText->sym       = NULL;
  aText->curLine   = NULL;
  aText->curChar   = NULL;
  aText->lastChar  = NULL;
  aText->completed = TRUE;
}
```

//
// nextSymbol //

CHeader : public

```
typedef void (NextSymbol)(
  TextObj *aText
);

#define nextSymbol(aText)                  \
  (                                        \
    assert(aText),                         \
    assert(getTextsClass(aText->jInterp)   \
      ->nextSymbolFunc),                   \
    (getTextsClass(aText->jInterp)         \
      ->nextSymbolFunc(aText))             \
  )
```

CHeader : private

```
extern void nextSymbolImpl(
  TextObj* aText
);
```

CCode : default

```
void nextSymbolImpl(
```

```
  TextObj* aText
) {
  assert(aText);
  assert(aText->jInterp);

  DEBUG(aText->jInterp,
    "->nextSymbol [%s]{%s}\n", aText->curLine, aText->curChar);
  aText->sym = NULL;

  // ensure we have a non-empty line
  while ((!aText->completed) && (aText->curChar == aText->lastChar)) {
    aText->nextLine(aText);
  }
  if (!aText->curLine) {
    DEBUG(aText->jInterp, "<-nextSymbol End of Text %s\n","");
    return; // we have exhausted this text
  }

  size_t  parsingNatural = TRUE;
  Symbol* symStart       = aText->curChar;
  Symbol* symEnd         = aText->curChar;
  Symbol* lastChar       = aText->lastChar;
  char    matchingQuote  = 0;
  while (symStart == symEnd) {
    while (symEnd < lastChar) {
      if (matchingQuote) {
        if (*symEnd == matchingQuote) break; // we have found our quote
        //
        // we are in the middle of a quote
        //
        symEnd++;
        parsingNatural = FALSE;
        //
      } else if (*symEnd == '\"' || *symEnd == '\'') {
        //
        // the beginning of a quote
        //
        symStart++;
        matchingQuote = *symEnd;
        symEnd++;
        //
      } else if (*symEnd < 33) {
        //
```

Code 3.22.6

```
45        // white space
46        //
47        if (symStart != symEnd)  break; // we have found a symbol
48        //
49        // ignore whitespace (of any type)
50        //
51        symStart++;
52        symEnd++;
53        //
54      } else if (
55        *symEnd == '('  || *symEnd == ')'  ||
56        *symEnd == '['  || *symEnd == ']'  ||
57        *symEnd == '{'  || *symEnd == '}' ) {
58        //
59        // '(', ')', '<', '>', '[', ']'
60        //
61        if (symStart != symEnd) break; // we have found a symbol
62        //
63        // '(' ')' are symbols in their own right
64        //
65        parsingNatural = FALSE;
66        symEnd++;
67        break;
68        //
69      } else if ( *symEnd == ';') {
70        //
71        if (symStart != symEnd) break; // we have found a symbol
72        //
73        const char* symEndP1 = symEnd + 1;
74        if ( symEndP1 < lastChar && *symEndP1 == ';') {
75          //
76          // we have found the beginning of a single line comment
77          // ... so restart search for the next symbol on the next line
78          //
79          while (++symEnd < lastChar) {
80            if (*symEnd == '\n' || *symEnd == '\r') {
81              //
82              // we have found the end of this line
83              //
84              while (++symEnd < lastChar) {
85                if (*symEnd != '\n' && *symEnd != '\r') {
86                  //
87                  // we have found the beginning of the next line
```

```
88                  //
89                  break;
90                }
91              }
92              break;
93            }
94          }
95          // ensure that the comment and all line-ends are skipped
96          symStart = symEnd;
97        } else {
98          //
99          // ';' is a symbol in its own right
100         //
101         parsingNatural = FALSE;
102         symEnd++;
103         break;
104         //
105       }
106       //
107     } else {
108       //
109       // any other character is part of a symbol
110       //
111       // check to see if we are still parsing a natural
112       //
113       if ('0' <= *symEnd && *symEnd <= '9') {
114         // do nothing
115       } else {
116         parsingNatural = FALSE;
117       }
118       //
119       symEnd++;
120       //
121     }
122   }
123
124   if (symStart == symEnd) {
125     aText->nextLine(aText);
126     if (!aText->curLine) return; // no more symbols
127     symStart = aText->curChar;
128     symEnd   = aText->curChar;
129     lastChar = aText->lastChar;
130   }
```

```
131   }
132
133   char* aSymbol = strndup(symStart, symEnd - symStart); // TODO this thrashes memory ;-(
134   if (matchingQuote && symEnd < lastChar) symEnd++; // skip ending quote
135   aText->curChar = symEnd;
136   DEBUG(aText->jInterp, "--nextSymbol == [%s]\n", aSymbol);
137
138   // check to see if this is the exit or quit symbol ...
139   // ... and if so reset the nextLine function
140   if (strcmp(aSymbol, "exit") == 0 || strcmp(aSymbol, "quit") == 0) {
141     aText->nextLine = nextLineExit;
142   }
143
144   if (parsingNatural) {
145     DEBUG(aText->jInterp, "--nextSymbol == %s (natural)\n", aSymbol);
146     // we have parsed a natural... so create a new natural
147     //
148     aText->sym = newNatural(aText->jInterp, aSymbol);
149     //
150     DEBUG(aText->jInterp, "<-nextSymbol %s (natural)\n", aSymbol);
151   } else {
152     // we have parsed a symbol... so create a new symbol
153     //
154     if (strchr(aSymbol, '.')) {
155       aText->sym =
156         newSymbol(aText->jInterp, aSymbol,
157           aText->fileName, aText->curLineNum);
158     } else {
159       assert(aText->jInterp);
160       assert(aText->jInterp->rootCtx);
161       assert(aText->jInterp->rootCtx->dict);
162       DictNodeObj* aSym = getSymbolEntry(
163           aText->jInterp->rootCtx->dict,
164           aSymbol
165         );
166       assert(aSym);
167       if (aSym->value && isSymbol(aSym->value)) {
168         aText->sym = aSym->value;
169       } else {
170         aText->sym =
171           newSymbol(aText->jInterp, aSym->symbol,
172             aText->fileName, aText->curLineNum);
173       }
```

```
    }
    //
    DEBUG(aText->jInterp, "<-nextSymbol [%s] (symbol)\n",
      ((SymbolObj*)(aText->sym))->sym);
  }
  free(aSymbol);
}
```

CHeader : public

```
typedef void (ReportError)(
  TextObj *aText,
  Symbol  *message
);

#define reportError(aText, message)       \
  (                                       \
    assert(aText),                        \
    assert(getTextsClass(aText->jInterp)  \
      ->reportErrorFunc),                 \
    (getTextsClass(aText->jInterp)        \
      ->reportErrorFunc(aText, message))  \
  )
```

CHeader : private

```
extern void reportErrorImpl(
  TextObj *aText,
  Symbol  *message
);
```

CCode : default

```
void reportErrorImpl(
  TextObj *aText,
  Symbol  *message
) {
  assert(aText);
  JoyLoLInterp *jInterp = aText->jInterp;
  assert(jInterp);
  StringBufferObj *aStrBuf =
    newStringBuffer(jInterp->rootCtx);
  strBufPrintf(aStrBuf, "\n\n%s\n", message);
  if (aText->curLine) {
    strBufPrintf(aStrBuf, "while parsing\n");
```

```
13      strBufPrintf(aStrBuf, "\t[%s]\n", aText->curLine);
14      strBufPrintf(aStrBuf,
15        "at character %zu\n\n", aText->curChar - aText->curLine);
16    } else {
17      strBufPrintf(aStrBuf, "at end of text\n\n");
18    }
19    jInterp->writeStdOut(jInterp, getCString(aStrBuf));
20    strBufClose(aStrBuf);
21  }
```

CHeader : private

```
1   extern Boolean equalityTextsCoAlg(
2     JoyLoLInterp *jInterp,
3     JObj         *lolA,
4     JObj         *lolB,
5     size_t        timeToLive
6   );
```

CCode : default

```
1   Boolean equalityTextsCoAlg(
2     JoyLoLInterp *jInterp,
3     JObj         *lolA,
4     JObj         *lolB,
5     size_t        timeToLive
6   ) {
7     assert(jInterp);
8     DEBUG(jInterp, "textsCoAlg-equal a:%p b:%p\n", lolA, lolB);
9     if (!lolA && !lolB) return TRUE;
10    if (!lolA && lolB)  return FALSE;
11    if (lolA  && !lolB) return FALSE;
12    if (asType(lolA) != asType(lolB)) return FALSE;
13    if (!asType(lolA)) return FALSE;
14    if (asTag(lolA)  != TextsTag) return FALSE;
15    if (lolA != lolB) return FALSE;
16    return TRUE;
17  }
```

3.22.6.1 Test Suite: printing texts

CHeader : private

```
1   extern Boolean printStrTextsCoAlg(
```

```
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
);
```

CCode : default

```
Boolean printStrTextsCoAlg(
  StringBufferObj *aStrBuf,
  JObj            *aLoL,
  size_t           timeToLive
) {
  assert(aLoL);
  assert(asTag(aLoL) == TextsTag);

  strBufPrintf(aStrBuf, "--texts-- ");
  return TRUE;
}
```

Test case

should print texts

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[TextsTag]);

  TextObj* aNewText = createTextFromString(jInterp, "a string");
  AssertPtrNotNull(aNewText);
  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  printLoL(aStrBuf, (JObj*)aNewText);
  AssertStrEquals(getCString(aStrBuf), "--texts-- ");
  strBufClose(aStrBuf);
```

3.22.6.2 Test Suite: registerTexts

CHeader : public

```
typedef struct texts_class_struct {
  JClass                    super;
  NextSymbol                   *nextSymbolFunc;
  ReportError                  *reportErrorFunc;
  FreeText                     *freeTextFunc;
  CreateTextFromInputFile      *createTextFromInputFileFunc;
```

Code 3.22.6

```
  CreateTextFromString         *createTextFromStringFunc;
  CreateTextFromArrayOfStrings *createTextFromArrayOfStringsFunc;
} TextsClass;
```

CCode : default

```
static Boolean initializeTexts(
  JoyLoLInterp *jInterp,
  JClass   *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerTexts(
  JoyLoLInterp *jInterp
);
```

CCode : default

```
Boolean registerTexts(
  JoyLoLInterp *jInterp
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  TextsClass* theCoAlg = joyLoLCalloc(1, TextsClass);
  assert(theCoAlg);

  theCoAlg->super.name           = TextsName;
  theCoAlg->super.objectSize     = sizeof(TextObj);
  theCoAlg->super.initializeFunc = initializeTexts;
  theCoAlg->super.registerFunc   = registerTextWords;
  theCoAlg->super.equalityFunc   = equalityTextsCoAlg;
  theCoAlg->super.printFunc      = printStrTextsCoAlg;
  theCoAlg->nextSymbolFunc       = nextSymbolImpl;
  theCoAlg->reportErrorFunc      = reportErrorImpl;
  theCoAlg->freeTextFunc         = freeTextImpl;
  theCoAlg->createTextFromInputFileFunc =
    createTextFromInputFileImpl;
  theCoAlg->createTextFromStringFunc =
    createTextFromStringImpl;
```

```
  theCoAlg->createTextFromArrayOfStringsFunc =
    createTextFromArrayOfStringsImpl;

  size_t tag =
    registerJClass(jInterp, (JClass*)theCoAlg);

  // do a sanity check...
  assert(tag == TextsTag);
  assert(jInterp->coAlgs[tag]);

  return TRUE;
}
```

Test case

should register the Texts coAlg

```
// CTestsSetup has already created a jInterp
// and run registerTexts
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
AssertPtrNotNull(getTextsClass(jInterp));
TextsClass *coAlg = getTextsClass(jInterp);
AssertIntTrue(registerTexts(jInterp));
AssertPtrNotNull(getTextsClass(jInterp));
AssertPtrEquals(getTextsClass(jInterp), coAlg);
AssertIntEquals(
  getTextsClass(jInterp)->super.objectSize,
  sizeof(TextObj)
)
```

3.22.7 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <joylol/jInterps.h>
#include <joylol/stringBuffers.h>
```

```
6  #include <joylol/symbols.h>
7  #include <joylol/naturals.h>
8  #include <joylol/dictNodes.h>
9  #include <joylol/dictionaries.h>
10 #include <joylol/cFunctions.h>
11 #include <joylol/texts.h>
12 #include <joylol/assertions.h>
13 #include <joylol/contexts.h>
14 #include <joylol/texts-private.h>
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.stringBuffers");
requireLuaModule(lstate, "joylol.symbols");
requireLuaModule(lstate, "joylol.naturals");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.contexts");
requireStaticallyLinkedTexts(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

4 Extension CoAlgebras

546

546
546

4.1 JoylolTests

4.1.1 Goals

The JoyLoL Tests coAlgebra provides a collection of assertions native to JoyLoL with which JoyLoL code can be unit tested.

4.1.2 Code

4.1.2.1 Test Suite: JoylolTestsTagMap

CHeader : private

```c
#define JoylolTestsName "joylolTests"
#include <joylol/hash_map.h>
typedef struct JoylolTestsTagMap JoylolTestsTagMap;
extern JoylolTestsTagMap *jtTagMap;
extern int    jtTagMap_set(JoylolTestsTagMap*, void*, size_t);
extern size_t jtTagMap_get(JoylolTestsTagMap*, void*);
```

CCode : default

```c
JOYLOL_HASH_MAP_DEFINE(
  JoylolTestsTagMap,
  void*,
  size_t,
  jtTagMap,
  ptrHash,
  ptrEquals
)

JoylolTestsTagMap *jtTagMap = NULL;

__attribute__((constructor))
static void initializeSharedLibrary(void) {
  if (jtTagMap) {
    printf("ERROR: JoylolTests TagMap already initialized: %p\n", jtTagMap);
    exit(-1);
  }
  jtTagMap = jtTagMap_new(16,3, 4);
}
```

Test case

should be able to add multiple key-values

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jtTagMap);
  jtTagMap_set(jtTagMap, (void*)0x100, 1);
  jtTagMap_set(jtTagMap, (void*)0x200, 2);
  jtTagMap_set(jtTagMap, (void*)0x300, 3);
  AssertIntEquals(jtTagMap_get(jtTagMap, (void*)0x100), 1);
  AssertIntEquals(jtTagMap_get(jtTagMap, (void*)0x200), 2);
  AssertIntEquals(jtTagMap_get(jtTagMap, (void*)0x300), 3);
```

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

4.1.2.2 Test Suite: newJoylolTest

CHeader : public

```
typedef struct joylolTest_object_struct JTestObj;
typedef struct joylolTest_object_struct {
  JObj      super;
} JTestObj;
```

CHeader : private

```
#define JoylolTestsName "joylolTests"
extern JObj* newJoylolTest(
  JoyLoLInterp* jInterp
);
```

CCode : default

```
JObj* newJoylolTest(
  JoyLoLInterp* jInterp
```

```
3   ) {
4     assert(jInterp);
5     assert(jInterp->coAlgs);

6     size_t jtTag = jtTagMap_get(jtTagMap, jInterp);
7     assert(jtTag); // should not be the unused tag

8     JObj* result = newObject(jInterp, jtTag);
9     assert(result);

10    result->type  = jInterp->coAlgs[jtTag];
11
12    return result;
13  }
```

Test case

should create a new joylolTest

```
AssertPtrNotNull(jInterp);

JObj* aNewJoylolTest = newJoylolTest(jInterp);
AssertPtrNotNull(aNewJoylolTest);
AssertPtrNotNull(asType(aNewJoylolTest));
AssertIntEquals(asTag(aNewJoylolTest),
  jtTagMap_get(jtTagMap, jInterp));
AssertIntTrue(isAtom(aNewJoylolTest));
AssertIntTrue(isJoylolTest(aNewJoylolTest));
AssertIntFalse(isPair(aNewJoylolTest));
```

Test case

print JoylolTest

```
AssertPtrNotNull(jInterp);

StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
AssertPtrNotNull(aStrBuf);

JObj* aLoL = newJoylolTest(jInterp);
AssertPtrNotNull(aLoL);
printLoL(aStrBuf, aLoL);
AssertStrEquals(getCString(aStrBuf), "joylolTest ");
```

```
strBufClose(aStrBuf);
```

4.1.2.3 Test Suite: isJoylolTest

CHeader : public

```
#define isJoylolTest(aLoL)                                    \
  (                                                           \
    (                                                         \
      (aLoL) &&                                               \
      asType(aLoL) &&                                         \
      (asTag(aLoL) == jtTagMap_get(jtTagMap, jInterp))        \
    ) ?                                                       \
      TRUE :                                                  \
      FALSE                                                   \
  )
```

CHeader : private

```
extern Boolean equalityJoylolTestsCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
);
```

CCode : default

```
Boolean equalityJoylolTestsCoAlg(
  JoyLoLInterp *jInterp,
  JObj         *lolA,
  JObj         *lolB,
  size_t        timeToLive
) {
  DEBUG(jInterp, "joylolTestsCoAlg-equal a:%p b:%p\n", lolA, lolB);
  if (!lolA && !lolB) return TRUE;
  if (!lolA && lolB)  return FALSE;
  if (lolA  && !lolB) return FALSE;
  if (asType(lolA) != asType(lolB)) return FALSE;
  if (!asType(lolA)) return FALSE;
  if (asTag(lolA)  != jtTagMap_get(jtTagMap, jInterp)) return FALSE;
  if (lolA != lolB) return FALSE;
  return TRUE;
```

```
16 }
```

4.1.2.4 Test Suite: printing joylolTests

CHeader : private

```
1  extern Boolean printJoylolTestsCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  );
```

CCode : default

```
1  Boolean printJoylolTestsCoAlg(
2    StringBufferObj *aStrBuf,
3    JObj            *aLoL,
4    size_t           timeToLive
5  ) {
6    assert(aLoL);
7    JoyLoLInterp *jInterp = aStrBuf->jInterp;
8    assert(jInterp);

9    assert(asTag(aLoL) == jtTagMap_get(jtTagMap, jInterp));
10
11   strBufPrintf(aStrBuf, "joylolTest ");

12   return TRUE;
13 }
```

Test case

should print joylolTests

```
  AssertPtrNotNull(jInterp);
  AssertPtrNotNull(jInterp->coAlgs[jtTagMap_get(jtTagMap, jInterp)]);

  StringBufferObj *aStrBuf = newStringBuffer(jInterp->rootCtx);
  AssertPtrNotNull(aStrBuf);

  JObj* aNewJoylolTest = newJoylolTest(jInterp);
  AssertPtrNotNull(aNewJoylolTest);
  printLoL(aStrBuf, aNewJoylolTest);
```

```
  AssertStrEquals(getCString(aStrBuf), "joylolTest ");
  strBufClose(aStrBuf);
```

4.1.2.5 Test Suite: registerJoylolTests

CHeader : public

```
typedef struct joylolTests_class_struct {
  JClass         super;
} JoylolTestsClass;
```

CCode : default

```
static Boolean initializeJoylolTests(
  JoyLoLInterp *jInterp,
  JClass       *aJClass
) {
  assert(jInterp);
  assert(aJClass);
  registerJoylolTestWords(jInterp);
  return TRUE;
}
```

CHeader : private

```
extern Boolean registerJoylolTests(
  JoyLoLInterp *jInterp
);
```

CCode : default

```
Boolean registerJoylolTests(
  JoyLoLInterp *jInterp
) {
  assert(jInterp);
  assert(jInterp->coAlgs);

  JoylolTestsClass* theCoAlg
    = joyLoLCalloc(1, JoylolTestsClass);
  assert(theCoAlg);

  theCoAlg->super.name           = JoylolTestsName;
  theCoAlg->super.objectSize     = sizeof(JTestObj);
  theCoAlg->super.initializeFunc = initializeJoylolTests;
```

```
12    theCoAlg->super.equalityFunc   = equalityJoylolTestsCoAlg;
13    theCoAlg->super.printFunc      = printJoylolTestsCoAlg;

14    size_t tag =
15      registerJClass(jInterp, (JClass*)theCoAlg);

16    jtTagMap_set(jtTagMap, jInterp, tag);

17    // do a sanity check...
18    assert(jInterp->coAlgs[tag]);

19    return TRUE;
20  }
```

Test case

should register the JoylolTests coAlg

```
// CTestsSetup has already created a jInterp
// and run registerJoylolTests
AssertPtrNotNull(jInterp);
AssertPtrNotNull(jInterp->coAlgs);
size_t jtTag = jtTagMap_get(jtTagMap, jInterp);
JClass *coAlg = getJClass(jInterp, jtTag);
AssertPtrNotNull(coAlg);
registerJoylolTests(jInterp);
AssertPtrNotNull(getJClass(jInterp, jtTag));
AssertPtrEquals(getJClass(jInterp, jtTag), coAlg);
AssertIntEquals(
  getJClass(jInterp, jtTag)->objectSize,
  sizeof(JTestObj)
)
```

4.1.3 Words

TODO: Consider changing the defineContext and defineNaming syntax to be similar to new define syntax; Fix dynamic definitions: defineTestsSetup, defineTestsTeardown, defineTestSuiteSetup, defineTestSuiteTeardown,

4.1.3.1 Test Suite: tests and assert naming scopes

We start by defining a naming scope to contain all of the Joylol test associated words. This naming space is denoted ‘tests’ in the ‘globals’ naming scope. The ‘tests’ naming scope’s parent naming scope is also ‘globals’.

JoylolCode : default

```
;; create a tests naming scope
globals      ;; parent naming scope
globals      ;; defining naming scope
tests        ;; defined name of this new naming scope
defineNaming ;; make the definition
```

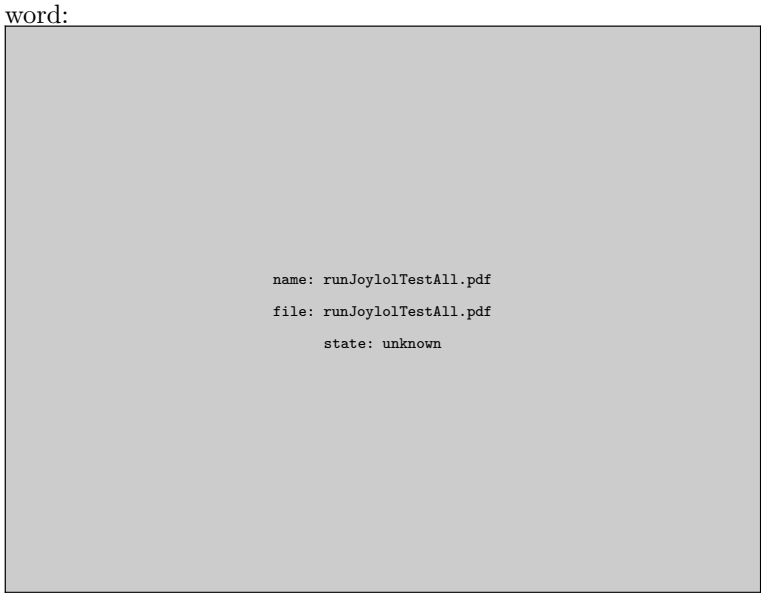
We also create a naming scope to contain all of the assertion related words.

JoylolCode : default

```
;; create an assert naming scope
tests        ;; parent naming scope
globals      ;; defining naming scope
assert       ;; defined name of this new naming scope
defineNaming ;; make the definition
```

4.1.3.2 Test Suite: running all tests

The following figure is a summary of the word/switch pattern of `runAllTests` word:



JoylolCode : default

```
(
  runAllTests           ;; name of word
  { true }              ;; pre condition
  (                     ;; start of the runAllTests word
    "running ALL TESTS" ;;
    writeOutNL          ;; announce our presence
```

We start by localizing the running context's naming scope and then defining the testsRunnerNS and testsMonitorNS naming scopes in the localized naming scope.

JoylolCode : default

```
    ;; localize the naming scope of the current context
    allTestsLocals  ;; name of the localized naming scope
    localizeNaming  ;; localize the naming scope
```

JoylolCode : default

```
    ;; undefine any existing testsRunnerNS
    ( testsRunnerNS )
    thisNaming
    undefine

    ;; create the testsRunnerNS
    thisNaming    ;; define the parent's naming scope
    thisNaming    ;; define in the localized naming scope
    testsRunnerNS ;; name of the new naming scope
    defineNaming  ;; create the new naming scope
                  ;; to be used as the testsRunner's naming scope
```

JoylolCode : default

```
    ;; undefine any existing testsMonitorNS
    ( testsMonitorNS )
    thisNaming
    undefine

    ;; create the testsMonitorNS
    thisNaming    ;; define the parent's naming scope
    thisNaming    ;; define in the localized naming scope
    testsMonitorNS;; name of the new naming scope
    defineNaming  ;; create the new naming scope
                  ;; to be used as the testsMonitor's naming scope
```

We now want to define default tests setup and teardown words in the testsRunnerNS.

JoylolCode : default

```
    (
      testsSetup  ;; name of the new word
```

```
    { true }    ;; pre condition
    ()          ;; empty default setup actions
    { true }    ;; post condition
  )
  testsRunnerNS ;; define in the testsRunnerNS
  define        ;; define the testsSetup word
```

JoylolCode : default

```
  (
    testsTeardown ;; name of the new word
    { true }    ;; pre condition
    ()          ;; empty default teardown actions
    { true }    ;; post condition
  )
  testsRunnerNS ;; define in the testsRunnerNS
  define        ;; define the testsTeardown word
```

We now create the testsRunner context.

JoylolCode : default

```
  ;; undefine any existing testsRunner context
  ( testsRunner )
  thisNaming
  undefine

  ;; define the testsRunner context
                ;; testsRunner process (uses provided list of words)
  ( tests.testsFinished )
  append        ;; ensure we get the tests results
                ;; when all suites have been run
  ()            ;; testsRunner data
  testsRunnerNS ;; naming scope of the new context
  thisNaming    ;; defined in the localized naming scope
  testsRunner   ;; name of the new context
  defineContext ;; define the new context
```

We now create the testsMonitor context.

JoylolCode : default

```
  ;; undefine any existing testsMonitor context
  ( testsMonitor )
  thisNaming
  undefine

  ;; define the testsMonitor context
  ( interpret )   ;; begin the testsMonitor process stack
  thisContext     ;; interpret thisContext onto data stack
```

```
    append          ;; append the result
    ( switchCtx )   ;;
    append          ;; append the word switchCtx to the
                    ;; testsMonitor's initial process stack

    (
      0             ;; number of attempted suites
      0             ;; number of failed suites
      ()            ;; list of failure reports
    )               ;; testsMonitor data

    testsMonitorNS  ;; naming scope of the new context
    thisNaming      ;; defining naming scope
    testsMonitor    ;; defined name of the new context
    defineContext   ;; define the context
```

Finally we want to switch to the testsRunner context, however, we also want to specify what actions to take when the testsMonitor context switches context back to this context. We do this by placing our context switch to testsRunner and any subsequent actions into a list and then we interpret this list.

JoylolCode : default

```
    ;; switch to the testsRunner context and run all of the tests
    (
      ;; switch to the testsRunner context
      testsRunner ;; place the testsRunner context on the stack
      switchCtx   ;; switch to the testsRunner context

      ;; when we are re-activated
      "finished ALL TESTS"  ;;
      writeOutNL            ;; say we are done
    )
    interpret     ;; place this list of actions onto
                  ;; our process stack and do them
  )
  { true }        ;; post condition
)
tests
define
```

We now define the `defineTestsSetup` and `defineTestsTeardown` words. While defined in the tests naming scope, they will be run in the testsRunner and its associated testsRunnerNS.

JoylolCode : default

```
(
```

```
  defineTestsSetup ;; name of word
  { true }         ;; pre condition
  (
    "defined tests setup"
    writeOutNL     ;; tell everyone we have been run

                   ;; we use the provided definition
    (
      setupTests   ;; name of the new word
      { true }     ;; pre condition
      swap12D
      { true }     ;; post condition
    )
    thisNaming     ;; defining naming scope
    define
  )
  { true }         ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  defineTestsTeardown ;; name of word
  { true }            ;; pre condition
  (
    "defined tests teardown"
    writeOutNL        ;; tell everyone we have been run

                      ;; we use the provided definition
    (
      teardownTests   ;; name of the new word
      { true }        ;; pre condition
      swap12D
      { true }        ;; post condition
    )
    thisNaming        ;; defining naming scope
    define
  )
  { true }            ;; post condition
)
tests
define
```

Once a particular test suite has finished, we need summarize the results and pass them to the tests runner. We do this with a pair of words one, `suiteFinished`, run in the suiteRunner and the other, `reportSuiteResults`, run in the suiteMonitor.

JoylolCode : default

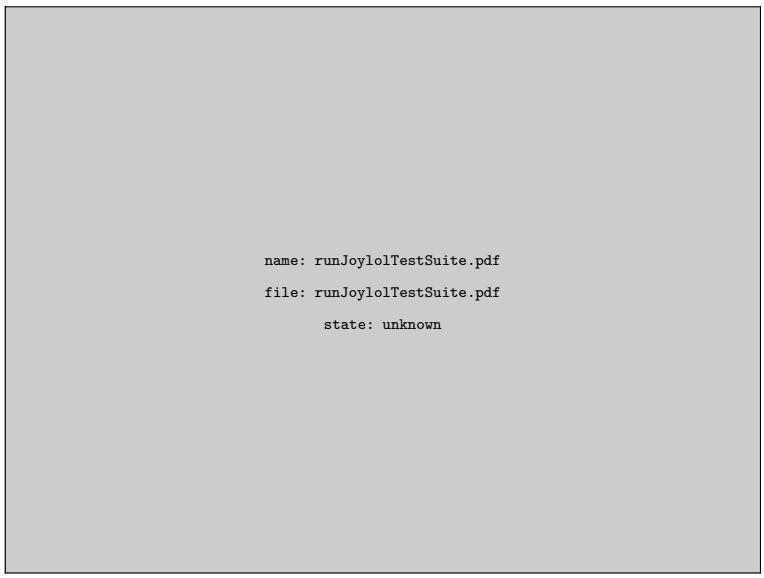
```
(
  testsFinished  ;; name of word
  { true }       ;; pre condition
  (
    ( tests.reportTestsResult )
    testsMonitor
    switchCtx
  )
  { true }       ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  reportTestsResult ;; name of word
  { true }          ;; pre condition
  (
                    ;; there is nothing more to do
                    ;; the testsMonitor already has
                    ;; the context switch
  )
  { true }          ;; post condition
)
tests
define
```

4.1.3.3 Test Suite: running a test suite

The following figure is a summary of the word/switch pattern of `runTestSuite` word:



JoylolCode : default

```
(
  runTestSuite
  { true }                ;; pre condition
  (                       ;; start of the runTestSuite word
    "running test suite"  ;;
    writeOutNL            ;; announce our presence
```

We are running in the testsRunner context so we already have a 'localized' naming scope. So for the `runTestSuite` word we only need to create the suiteRunnerNS and suiteMonitorNS naming scopes.

JoylolCode : default

```
    ;; undefine any existing suiteRunnerNS
    ( suiteRunnerNS )
    thisNaming
    undefine

    ;; create the suiteRunnerNS
    thisNaming    ;; define the parent's naming scope
    thisNaming    ;; define in the localized naming scope
    suiteRunnerNS ;; name of the new naming scope
    defineNaming  ;; create the new naming scope
                  ;; to be used as the suiteRunner's naming scope
```

JoylolCode : default

```
    ;; undefine any existing suiteMonitorNS
```

```
    ( suiteMonitorNS )
    thisNaming
    undefine

    ;; create the suiteMonitorNS
    thisNaming      ;; define the parent's naming scope
    thisNaming      ;; define in the localized naming scope
    suiteMonitorNS  ;; name of the new naming scope
    defineNaming    ;; create the new naming scope
                    ;; to be used as the suiteMonitor's naming scope
```

We now want to define the default suite setup and teardown words in the suiteRunnerNS.

JoylolCode : default

```
    (
      suiteSetup  ;; name of the new word
      { true }    ;; pre condition
      ()          ;; empty default setup actions
      { true }    ;; post condition
    )
    suiteRunnerNS ;; define in the suiteRunnerNS
    define        ;; define the suiteSetup word
```

JoylolCode : default

```
    (
      suiteTeardown ;; name of the new word
      { true } ;; pre condition
      ()            ;; empty default teardown actions
      { true } ;; post condition
    )
    suiteRunnerNS ;; define in the suiteRunnerNS
    define        ;; define the suiteTeardown word
```

We now create the suiteRunner context.

JoylolCode : default

```
    ;; undefine any existing suiteRunner context
    ( suiteRunner )
    thisNaming
    undefine

    ;; define the suiteRunner context
                ;; suiteRunner process (uses provided list of words)
    ( tests.suiteFinished )
    append        ;; ensure we get the suite results
                  ;; at the end of the suite
```

```
    ()            ;; suiteRunner data
    suiteRunnerNS ;; naming scope of the new context
    thisNaming    ;; defined in the localized naming scope
    suiteRunner   ;; name of the new context
    defineContext ;; define the new context
```

We now create the suiteMonitor context.

JoylolCode : default

```
    ;; undefine any existing suiteMonitor context
    ( suiteMonitor )
    thisNaming
    undefine

    ;; define the suiteMonitor context
    ( interpret )   ;; begin the suiteMonitor process stack
    thisContext     ;; interpret thisContext onto data stack
    append          ;; append the result
    ( switchCtx )   ;;
    append          ;; append the word switchCtx to the
                    ;; suiteMonitor's initial process stack

    (
      0             ;; number of attempted cases
      0             ;; number of failed cases
      ()            ;; list of failure reports
    )               ;; suiteMonitor data

    suiteMonitorNS  ;; naming scope of the new context
    thisNaming      ;; defining naming scope
    suiteMonitor    ;; name of the new context
    defineContext   ;; define the context
```

Finally we want to switch to the suiteRunner context, however, again, we also want to specify what actions to take when the suiteMonitor context switches back to this context. We do this by placing our context switch to suiteRunner and any subsequent actions into a list and then we interpret this list.

JoylolCode : default

```
    ;; switch to the suiteRunner context and run the suite
    (
      ;; switch to the suiteRunner context
      suiteRunner ;; place the suiteRunner context on the stack
      switchCtx   ;; switch to the suiteRunner context

      ;; when we are re-activated
```

```
      ;; ... HOW DO WE SWITCH BACK???!!
    )
    interpret
  )
  { true }        ;; post condition
)
tests
define
```

We now define the `recordTestSutieDetails`, `defineTestSuiteSetup`, and `defineTestSuiteTeardown` words.

JoylolCode : default

```
(
  recordTestSuiteDetails
  { true } ;; pre condition
  (
    (
      newLine
      "-----------------------------------"
      newLine
      "jTS:"
    )           ;; format a sign that we are running
                ;; the test suite
    prepend     ;; prepend this sign to the name of the suite
    100         ;; limit the printLoL
    swap12D     ;; put the lol and the depth limit
                ;; in the correct order
    printLoL    ;; print it to a string
    writeOutNL  ;; write out the string
  )
  { true }      ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  defineTestSuiteSetup
  { true }        ;; pre condition
  (
    "defined test suite setup"
    writeOutNL    ;; tell everyone we have been run

                  ;; we use the provided definition
```

```
    (
      setupSuite  ;; name of the new word
      { true }    ;; pre condition
      swap12D
      { true }    ;; post condition
    )
    thisNaming  ;; defining naming scope
    define
  )
  { true } ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  defineTestSuiteTeardown
  { true } ;; pre condition
  (
    "defined test suite teardown"
    writeOutNL    ;; tell everyone we have been run

    (
      teardownSuite ;; name of the new word
                    ;; we use the provided definition
      { true }      ;; pre condition
      swap12D
      { true }      ;; post condition
    )
    thisNaming      ;; defining naming scope
    define
  )
  { true }        ;; post condition
)
tests
define
```

Once a particular test suite has finished, we need summarize the results and pass them to the tests runner. We do this with a pair of words one, `suiteFinished`, run in the suiteRunner and the other, `reportSuiteResults`, run in the suiteMonitor.

JoylolCode : default

```
(
  suiteFinished
  { true }   ;; pre condition
```

```
  (
    ( tests.reportSuiteResults )
    suiteMonitor
    switchCtx
  )
  { true } ;; post condition
)
tests
define
```

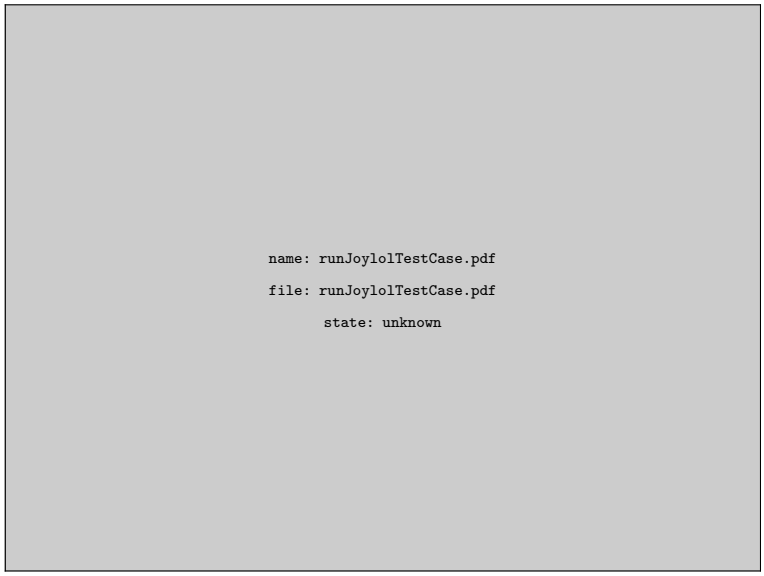
JoylolCode : default

```
(
  reportSuiteResults
  { true }  ;; pre condition
  (
            ;; there is nothing more to do
            ;; the suiteMonitor already has
            ;; the context switch
  )
  { true }  ;; post condition
)
tests
define
```

4.1.3.4 Test Suite: runTestCase

Our next task, over the next following code fragments, is to define the runTestCase word. This word takes a test case, as a list of the words to be tested intermingled with assertions, and creates the caseMonitor and caseRunner contexts which together run the test case provided. To be completely re-entrant, the caseMonitor and caseRunner contexts are stored in the running context's localized naming scope.

The following figure is a summary of the word/switch pattern of `runTestCase` word:



JoylolCode : default

```
(
  runTestCase   ;; the name of the new word
  { true }      ;; pre condition
  (             ;; start of the runTestCase word
```

We are running in the suiteRunner context so we already have a 'localized' naming scope. So for the `runTestCase` word we only need to create the caseRunnerNS and caseMonitorNS naming scopes.

JoylolCode : default

```
    ;; undefine any existing caseRunnerNS
    ( caseRunnerNS )
    thisNaming
    undefine

    ;; create the caseRunnerNS
    thisNaming   ;; parent naming scope
    thisNaming   ;; define in localized naming scope
    caseRunnerNS ;; name of the new naming scope
    defineNaming ;; create a new naming scope
                 ;; to be used as the caseRunner's naming scope
```

JoylolCode : default

```
    ;; undefine any existing caseMonitorNS
    ( caseMonitorNS )
    thisNaming
```

```
    undefine

    ;; create the caseMonitorNS
    thisNaming    ;; parent naming scope
    thisNaming    ;; define in the localized naming scope
    caseMonitorNS ;; name of the new naming scope
    defineNaming  ;; create a new naming scope
                  ;; to be used as the caseMonitor's naming scope
```

At this point the test case's list of words and assertions are back on the top of the stack. This list will be used as the caseRunner's initial process stack. We can now add the initial data stack for the caseRunner context, and then setup the new context's naming scope, the defining naming scope and finally the name of the new context.

JoylolCode : default

```
    ;; undefine any existing caseRunner context
    ( caseRunner )
    thisNaming
    undefine

    ;; define the caseRunner context
                  ;; caseRunner process (uses provided list of words)
    ( tests.caseFinished )
    append        ;; ensure we get the case results
                  ;; at the end of the case
    ()            ;; caseRunner data
    caseRunnerNS  ;; the naming scope of the new context
    thisNaming    ;; define in the localized naming scope
    caseRunner    ;; name of the new context
    defineContext ;; create and define the context
```

Now we create a caseMonitor context which will keep track of the test results. The caseMonitor's initial process stack should contain the three words, `interpret`, the current context, and `switchCtx` in this order. When an assertion switches from the caseRunner's context to the caseMonitor context the top of the caseRunner's stack is place onto the top of the caseMonitor's data stack. The `interpret` word will then place the words on the top of the caseMonitor's data stack onto the caseMonitor's process stack to be interpreted. The assertion being interpreted *should* ensure it re-places the `interpret` word back on the caseMonitor's process stack for use by the next assertion.

The last two words, the current context and `switchCtx`, as used by the textMonitor interpreting the `results` word to know which context to switch to with the summary results.

Since the `thisContext` word has to be interpreted in the calling context, we have to `append` the result of interpreting the `thisContext` word rather than just

placing the `thisContext` word directly on the caseMonitor's process stack. Once the

JoylolCode : default

```
    ;; undefine any existing caseMonitor context
    ( caseMonitor )
    thisNaming
    undefine

    ;; define the caseMonitor context
    ( interpret ) ;; begin the caseMonitor process stack
    thisContext   ;; interpret thisContext
    append        ;; append the result of interpreting thisContext
    ( switchCtx ) ;;
    append        ;; append the word switch to the
                  ;; caseMonitor's initial process stack

    (
      false       ;; should fail indicator
      0           ;; number of attempted assertions
      0           ;; number of failed assertions
      ()          ;; list of failure reports
    )             ;; caseMonitor data

    caseMonitorNS ;; the naming scope of the new context
    thisNaming    ;; defining naming scope
    caseMonitor   ;; defined name of this new context
    defineContext ;; create and define the context
```

Finally we switch to the caseRunner context.

JoylolCode : default

```
    ;; switch to the caseRunner context and run the assertions
    caseRunner    ;; place the caseRunner context on the stack
    switchCtx     ;; switch to the caseRunner context

  )               ;; end of the runTestCase definition
  { true }        ;; post condition
)
tests             ;; define the new word in this naming scope
define            ;; define the new runTestCase word
```

We now define the `recordTestCaseDetails` word.

JoylolCode : default

```
(
  recordTestCaseDetails
```

```
  { true }      ;; pre condition
  (
    "  jTC:"    ;; format a sign that we are running
                ;; the test case
    prepend     ;; prepend this sign to the name of the case
    100         ;; limit the printLoL
    swap12D     ;; put the lol and the depth limit
                ;; in the correct order
    printLoL    ;; print it to string
    writeOutNL  ;; write out the string
  )
  { true }      ;; post condition
)
tests
define
```

4.1.3.5 Test Suite: communicate case results

Once a particular test case has finished, we need summarize the results and pass them to the suite runner. We do this with a pair of words one, `caseFinished`, run in the caseRunner and the other, `reportCaseResults`, run in the caseMonitor.

JoylolCode : default

```
(
  caseFinished
  { true } ;; pre condition
  (
    ( tests.reportCaseResults )
    caseMonitor
    switchCtx
  )
  { true } ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  reportCaseResults
  { true } ;; pre condition
  (
    pop1D   ;; ignore the should fail boolean
            ;; there is nothing more to do
            ;; the caseMonitor already has
```

```
            ;; the context swtich
  )
  { true }  ;; post condition
)
tests
define
```

4.1.3.6 Test Suite: assert.reportAssertion

In this section we develop the `reportAssertion` word in the `assert` naming scope. This word takes the result of an individual assertion, switches to the `caseMonitor` context and records it before switching back to the `caseRunner` context.

The `reportAssertion` word assumes that any previous assertion has left an assertion result list on the top of the caseRunner's data stack. This assertion result list consists of true/false followed by a failure report list (only if the assertion failed).

JoylolCode : default

```
(
  reportAssertion ;; the name of the new word
  { true }        ;; pre condition
  (               ;; start of the reportAssertion word
                  ;; the assertion report is assumed to be on top
    (
      assert.recordAssertion  ;; the task to be done
      caseRunner  ;; return to caseRunner
      switchCtx   ;; perform the switch/return
      interpret   ;; leave an initial interpret for the next switch/call
    )
    append
    caseMonitor   ;; place the caseMonitor context on top of stack
    switchCtx     ;; switch to the caseMonitor context
                  ;; (the caseMonitor context will interpret to
                  ;;  current top of the stack)
  )               ;; end of the reportAssertion word
  { true }        ;; post condition
)
assert          ;;define the new word in the assert naming scope
define          ;; define the reportAssertion word
```

4.1.3.7 Test Suite: assert.recordAssertion

The `recordAssertion` word is run on the caseMonitor context. It assumes that the top of the data stack is a true/false depending upon the success/failure of the

previous assertion. If the top of the stack is false, then the `recordAssertion` word also expects a list containing the assertion failure reasons as the second item on the stack. The subsequent item on the stack is then the context to which to return so that any further assertions can be run. Finally the last item on the stack is the caseMonitor's record of assertion results.

JoylolCode : default

```
(
  recordAssertion
  { true }    ;; pre condition
  (
    showStack ;; need to deal with shouldFail....
    (
      isTrue
    )  ;; test
    (
      ;; the top of the stack is the caseMonitor's
      ;; record of assertion results
      extract
      ;; the top of the stack is the number of attempts value
      1 + ;; add one to the top of the stack
      prepend
      prepend
      ;; the top of the stack is the
      ;; updated record of assertion results
    )  ;; then action ;; just update the number of attempted tests
    (
      ;; the top of the stack is an assertion failure report list
      ;; then comes the caseMonitor's record of assertion results
      append
      extract
      ;; top    comes the number of attempts value
      ;; second comes the number of failures value
      ;; third  comes the collection of failure reports
      ;; fourth comes the assertion failure report list
      1 + ;; add one to the number of attempted
      swap12D
      1 + ;; add one to the number of failed
      append
      rollupD ;; put current record third
      append  ;; append the assertion failure report to list or failures
      append  ;; complete the record of assertion results
      ;; top comes the updated record of assertion results
    )  ;; else action ;; update everything
```

```
    ifte
  )
  { true }   ;; post condition
)
assert
define
```

4.1.3.8 Test Suite: assertShouldFail

The `assert.shouldFail` word wraps the interpretation of the body of the should fail assertion in a pair of `tests.startShouldFail` and `tests.stopShouldFail` words. The `startShouldFail` word ensures the top of the caseMonitor data stack contains `true` so that any failures are interpreted as successes. The `stopShouldFail` word ensures that the top of the caseMonitor data stack contains a `false` to ensure any failures are reported as failures.

We begin by defining the `tests.startShouldFail` word. It builds the actions which will pop the top of the data stack and replaces it with a `true` and then switch back to the current context. Having setup the appropriate list of actions on the top of the current context's data stack, the `startShouldFail` word then switches to the caseMonitor context. The `switchCtx` word pops the top of the current context's data stack and pushes it on the top of the caseMonitor's data stack. This all assumes that the top of the caseMonitor's process stack is the `interpret` word which will have the effect of interpreting the list of actions copied over from the current context to the caseMonitor's data stack.

JoylolCode : default

```
(
  startShouldFail
  { true }        ;; pre condition
  (
    (
      pop1D
      true
      dup
    )
    thisContext
    append
    ( switchCtx )
    append
    caseMonitor
    switchCtx
    pop1D
  )
```

```
  { true } ;; post condition
)
tests
define
```

JoylolCode : default

```
(
  stopShouldFail
  { true } ;; pre condition
  (
    (
      pop1D
      false
      dup
    )
    thisContext
    append
    ( switchCtx )
    append
    caseMonitor
    switchCtx
    pop1D
  )
  { true } ;; post condition
)
tests
define
```

The `assert.shouldFail` word now uses the `tests.startShouldFail` and `tests.stopShouldFail` words to bracket the interpretation of the actions found quoted on the current context's data stack.

JoylolCode : default

```
(
  shouldFail
  { true } ;; pre condition
  (
    tests.startShouldFail
    interpret
    tests.stopShouldFail
  )
  { true } ;; post condition
)
assert
define
```

Test case

assert.shouldFail

```
( assert.fail ) assert.shouldFail
```

4.1.3.9 Test Suite: assert.fail

JoylolCode : default

```
(
  fail
  { true } ;; pre condition
  (
    false
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

assert.fail should fail

```
( assert.fail ) assert.shouldFail
```

4.1.3.10 Test Suite: assert.succeed

JoylolCode : default

```
(
  succeed
  { true } ;; pre condition
  (
    true
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

assert.succeed should succeed

```
assert.succeed
```

4.1.3.11 Test Suite: assert.isBoolean

JoylolCode : default

```
(
  isBoolean
  { true } ;; pre condition
  (
    isBoolean
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a boolean

```
true
assert.isBoolean
```

Test case

should fail if an object is not a boolean

```
(
  notABoolean
  assert.isBoolean
) assert.shouldFail
```

4.1.3.12 Test Suite: assert.isTrue

JoylolCode : default

```
(
  isTrue
```

```
  { true } ;; pre condition
  (
    isTrue
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is true

```
true
assert.isTrue

notABooleanButTrue
assert.isTrue
```

Test case

should fail if an object is not true

```
(
  false
  assert.isTrue
) assert.shouldFail
(
  ()
  assert.isTrue
) assert.shouldFail
```

4.1.3.13 Test Suite: assert.isFalse

JoylolCode : default

```
(
  isFalse
  { true } ;; pre condition
  (
    isFalse
    assert.reportAssertion
```

```
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is false

```
false
assert.isFalse

()
assert.isFalse
```

Test case

should fail if an object is not false

```
(
  true
  assert.isFalse
) assert.shouldFail
(
  notABooleanButTrue
  assert.isFalse
) assert.shouldFail
```

4.1.3.14 Test Suite: assert.isNil

JoylolCode : default

```
(
  isNil
  { true } ;; pre condition
  (
    isNil
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
```

```
11  define
```

Test case

should succeed if an object is nil

```
()
assert.isNil
```

Test case

should fail if an object is not nil

```
(
  ( someThingNotNil )
  assert.isNil
) assert.shouldFail
```

4.1.3.15 Test Suite: assert.isNotNil

JoylolCode : default

```
1   (
2     isNotNil
3     { true } ;; pre condition
4     (
5       isNil
6       not
7       assert.reportAssertion
8     )
9     { true } ;; post condition
10  )
11  assert
12  define
```

Test case

should succeed if an object is not nil

```
( this is a list )
assert.isNotNil
```

Test case

should fail if an object is nil

```
(
  ( )
  assert.isNotNil
) assert.shouldFail
```

4.1.3.16 Test Suite: assert.isAnAtom

JoylolCode : default

```
(
  isAnAtom
  { true } ;; pre condition
  (
    isAnAtom
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is an atom

```
thisIsAnAtom
assert.isAnAtom
thisContext
assert.isAnAtom
123
assert.isAnAtom
```

Test case

should fail if an object is not an atom

```
(
  ( this is a list )
  assert.isAnAtom
) assert.shouldFail
```

4.1.3.17 Test Suite: assert.isAPair

JoylolCode : default

```
(
  isAPair
  { true } ;; pre condition
  (
    isAPair
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a pair

```
( this is a list)
assert.isAPair
```

Test case

should succeed if an object is not a pair

```
(
  true
  assert.isAPair
) assert.shouldFail
```

4.1.3.18 Test Suite: assert.isANatural

JoylolCode : default

```
(
  isANatural
  { true } ;; pre condition
  (
    isANatural
    assert.reportAssertion
```

```
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a natural

```
1234
assert.isANatural
```

Test case

should succeed if an object is not a natural

```
(
  aSymbol
  assert.isANatural
) assert.shouldFail
```

4.1.3.19 Test Suite: assert.isASymbol

JoylolCode : default

```
(
  isASymbol
  { true } ;; pre condition
  (
    isASymbol
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a symbol

```
aSymbol
assert.isASymbol
```

Test case

should succeed if an object is not a symbol

```
(
  12345
  assert.isASymbol
) assert.shouldFail
```

4.1.3.20 Test Suite: assert.isAContext

JoylolCode : default

```
(
  isAContext
  { true } ;; pre condition
  (
    isAContext
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a context

```
thisContext
assert.isAContext
```

Test case

should succeed if an object is not a context

```
(
  1234
  assert.isAContext
) assert.shouldFail
```

4.1.3.21 Test Suite: assert.isADictionary

JoylolCode : default

```
(
  isADictionary
  { true } ;; pre condition
  (
    isADictionary
    assert.reportAssertion
  )
  { true } ;; post condition
)
assert
define
```

Test case

should succeed if an object is a dictionary

```
thisNaming
assert.isDictionary
```

Test case

should succeed if an object is not a dictionary

```
(
  1234
  assert.isDictionary
) assert.shouldFail
```

```
\startTestSuite[assert.isADictNode]

\startJoylolCode
\stopJoylolCode

\startTestCase[should succeed if an object is]
\startJoylolTest
\stopJoylolTest
\stopTestCase

\startTestCase[should succeed if an object is not]
\startJoylolTest
```

```
\stopJoylolTest
\stopTestCase
\stopTestSuite
```

CHeader : private

```
extern void registerJoylolTestWords(
  JoyLoLInterp* jInterp
);
```

CCode : default

```
void registerJoylolTestWords(
  JoyLoLInterp* jInterp
) {
  assert(jInterp);

}
```

JoylolCode : default

```
;; a final line
```

4.1.4 Lua functions

CCode : default

```
static int lua_joylolTests_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_joylolTests [] = {
  {"gitVersion", lua_joylolTests_getGitVersion},
  {NULL, NULL}
};

int luaopen_joylol_joylolTests (lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
  registerJoylolTests(jInterp);
```

```
20    luaL_newlib(lstate, lua_joylolTests);
21    return 1;
22  }
```

In some instances, such as the typical `CTest` program `allCTests`, this Lua module (which can be `required` as a shared library) is actually statically linked into the executable. In these cases we need the ability to mimic the standard Lua `require` process. The following `requireStaticallyLinkedJoylolTests` does just this.

CHeader : public

```
1  Boolean requireStaticallyLinkedJoylolTests(
2    lua_State *lstate
3  );
```

CCode : default

```
1  Boolean requireStaticallyLinkedJoylolTests(
2    lua_State *lstate
3  ) {
4    lua_getglobal(lstate, "package");
5    lua_getfield(lstate, -1, "loaded");
6    luaopen_joylol_joylolTests(lstate);
7    lua_setfield(lstate, -2, "joylol.joylolTests");
8    lua_setfield(lstate, -2, "loaded");
9    lua_pop(lstate, 1);
10   return TRUE;
11 }
```

4.1.5 Conclusions

CHeader : public

CHeader : private

```
1  extern size_t joylol_register_joylolTests(JoyLoLInterp *jInterp);
```

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <joylol/jInterps.h>
5  #include <joylol/stringBuffers.h>
6  #include <joylol/dictNodes.h>
```

```
7  #include <joylol/dictionaries.h>
8  #include <joylol/texts.h>
9  #include <joylol/cFunctions.h>
10 #include <joylol/assertions.h>
11 #include <joylol/contexts.h>
12 #include <joylol/joylolTests.h>
13 #include <joylol/joylolTests-private.h>
14 // dictionary
15 // printer
```

```
addJoyLoLLuaPath(lstate);
requireStaticallyLinkedJInterps(lstate);
requireLuaModule(lstate, "joylol.assertions");
requireLuaModule(lstate, "joylol.pairs");
requireLuaModule(lstate, "joylol.cFunctions");
requireLuaModule(lstate, "joylol.texts");
requireLuaModule(lstate, "joylol.contexts");
requireLuaModule(lstate, "joylol.dictionaries");
requireLuaModule(lstate, "joylol.dictNodes");
requireLuaModule(lstate, "joylol.stringBuffers");
requireStaticallyLinkedJoylolTests(lstate);
getJoyLoLInterpInto(lstate, jInterp);
initializeAllLoaded(lstate, jInterp);
registerAllLoaded(lstate, jInterp);
```

Lmsfile : default

Lmsfile : default

Lmsfile : default

Lmsfile : default

587

5 Core Implementations

587
587

588

588
588

5.1 JoyLoL ConTEXt module

590

590
590

5.2 Overview

We document the options required *before* loading JoyLoL.

LuaCode : default

```lua
-- joylol loader options

options.verbose    = false
options.debug      = false
options.configFile = 'config'
options.userPath   = os.getenv('HOME')..'/.joylol'
options.localPath  = '/usr/local/lib/joylol'
options.systemPath = '/usr/lib/joylol'
```

LuaCode : default

```lua
local gitVersion = {
  authorName      = "Stephen Gaito",
  commitDate      = "2018-12-03",
  commitShortHash = "38e0564",
  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
  subject         = "updated textadept lexer for JoyLoL",
  notes           = ""
}
```

LuaCode : default

```lua
options.gitVersion = gitVersion
```

MkIVCode : default

```
\def\joyLoLQuiet{
  \directlua{
    thirddata.joylol.options.verbose = false
    thirddata.joylol.options.debug   = false
    thirddata.joylol.options.tracing = false
  }
}

\def\joyLoLVerbose{
  \directlua{
    thirddata.joylol.options.verbose = true
  }
}
```

```
14
15  \def\joyLoLDebug{
16    \directlua{
17      thirddata.joylol.options.debug   = true
18    }
19  }
20
21  \def\joyLoLTrace{
22    \directlua{
23      thirddata.joylol.options.tracing = true
24    }
25  }
26
27  \def\joyLoLNoConfiguration{
28    \directlua{
29      thirddata.joylol.options.configFile = nil
30    }
31  }
32
33  \def\joyLoLConfigFile#1{
34    \directlua{
35      thirddata.joylol.options.configFile = '#1'
36    }
37  }
38
39  \def\joyLoLUserPath#1{
40    \directlua{
41      thirddata.joylol.options.userPath = '#1'
42    }
43  }
44
45  \def\joyLoLLocalPath#1{
46    \directlua{
47      thirddata.joylol.options.localPath = '#1'
48    }
49  }
50  \def\joyLoLSystemPath#1{
51    \directlua{
52      thirddata.joylol.options.systemPath = '#1'
53    }
54  }
55  \def\joyLoLLoadAnsic{
56    \directlua{
```

```
    thirddata.joylol.options.minimalJoylol = false
  }
}
\def\joyLoLLoadLua{
  \directlua{
    thirddata.joylol.options.minimalJoylol = true
  }
}
```

594

594
594

5.3 Preamble

MkIVCode : default

```
%D \module
%D   [     file=t-joylol-opts,
%D      version=2017.05.10,
%D        title=\CONTEXT\ User module,
%D     subtitle=The loading options for the JoyLoL prgramming language for \ConTeXt\,
%D       author=Stephen Gaito,
%D         date=\currentdate,
%D    copyright=PerceptiSys Ltd (Stephen Gaito),
%D        email=stephen@perceptisys.co.uk,
%D      license=MIT License]

%C Copyright (C) 2017 PerceptiSys Ltd (Stephen Gaito)
%C
%C Permission is hereby granted, free of charge, to any person obtaining a
%C copy of this software and associated documentation files (the
%C "Software"), to deal in the Software without restriction, including
%C without limitation the rights to use, copy, modify, merge, publish,
%C distribute, sublicense, and/or sell copies of the Software, and to
%C permit persons to whom the Software is furnished to do so, subject to
%C the following conditions:
%C
%C The above copyright notice and this permission notice shall be included
%C in all copies or substantial portions of the Software.
%C
%C THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
%C OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
%C MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
%C IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
%C CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
%C TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
%C SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

% begin info
%
% title   : JoyLoL loading options
% comment : Provides the loading options required for JoyLoL
% status  : under development, mkiv only
%
% end info
```

```

\unprotect

\ctxloadluafile{t-joylol-opts}
```

MkIVCode : default

```
\protect \endinput
```

LuaCode : default

```
-- This is the lua code associated with t-joylol-opts.mkiv

if not modules then modules = { } end modules ['t-joylol-opts'] = {
    version   = 1.000,
    comment   = "joylol loading options - lua",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
    copyright = "PerceptiSys Ltd (Stephen Gaito)",
    license   = "MIT License"
}

thirddata        = thirddata        or {}
thirddata.joylol = thirddata.joylol or {}

local joylol   = thirddata.joylol
joylol.options = joylol.options or {}

local options  = joylol.options

local tInsert = table.insert
local tConcat = table.concat
local tRemove = table.remove
local tSort   = table.sort
local sFmt    = string.format
local sMatch  = string.match
local toStr   = tostring
```

5.4 Conclusion

Lmsfile : default

598

598
598

5.5 The core JoyLoL embedded in ConTeXt

5.6 Overview

We document the ConTEXt interface to the JoyLoL language.

MkIVCode : default

```
%  authorName      = "Stephen Gaito",
%  commitDate      = "2018-12-03",
%  commitShortHash = "38e0564",
%  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
%  subject         = "updated textadept lexer for JoyLoL",
%  notes           = ""
```

602

602
602

5.7 Lua main

LuaCode : default

```lua
local gitVersion = {
  authorName      = "Stephen Gaito",
  commitDate      = "2018-12-03",
  commitShortHash = "38e0564",
  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
  subject         = "updated textadept lexer for JoyLoL",
  notes           = ""
}
```

LuaCode : default

```lua
-- joylol interpreter embedded in ConTeXt

-- Start by adding the standard joylol CoAlg locations to the Lua search
-- paths

texio.write_nl("----------JoyLoL options-----------")
texio.write_nl(prettyPrint(options))
texio.write_nl("-----------------------------------")

local joylolPaths = {
  options.userPath..'/?.lua',
  options.localPath..'/?.lua',
  options.systemPath..'/?.lua',
  package.path
}
package.path = table.concat(joylolPaths, ';')

local joylolCPaths = {
  options.userPath..'/?.so',
  options.localPath..'/?.so',
  options.systemPath..'/?.so',
  package.path
}
package.cpath = table.concat(joylolCPaths, ';')

if options.verbose then print('loading [joylol.core.context]') end

local hasJoylol, loadedJoylol =
  pcall(require, 'joylol.core.context')
```

```
if not hasJoylol then
  interfaces.writestatus("joyLoL",
    "Could NOT load ANSI-C joyLoL....")
  error('Could NOT load ANSI-C JoyLoL')
end
thirddata.joylol = loadedJoylol

if options.verbose then print('loaded [joylol.core.context]\n') end

local joylol = thirddata.joylol

joylol.setVerbose(options.verbose)
joylol.setTracing(options.tracing)
joylol.setDebugging(options.debug)

if (options.configFile) then
  joylol.loadFile(options.configFile)
end
```

5.7.1 Lua interface

MkIVCode : default

```
\def\setJoyLoLVerbose{%
  \directlua{%
    thirddata.joylol.setVerbose(true)
  }
}

\def\clearJoyLoLVerbose{%
  \directlua{%
    thirddata.joylol.setVerbose(false)
  }
}
```

MkIVCode : default

```
\def\setJoyLoLTracing{%
  \directlua{%
    thirddata.joylol.setTracing(true)
  }
}
```

```
\def\clearJoyLoLTracing{%
  \directlua{%
    thirddata.joylol.setTracing(false)
  }
}
```

MkIVCode : default

```
\def\setJoyLoLDebugging{%
  \directlua{%
    thirddata.joylol.setDebugging(true)
  }
}

\def\clearJoyLoLDebugging{%
  \directlua{%
    thirddata.joylol.setDebugging(false)
  }
}
```

MkIVCode : default

```
\def\pushJoyLoLLoadPath#1{%
  \directlua{%
    thirddata.joylol.pushLoadPath('#1')
  }
}
```

MkIVCode : default

```
\def\loadJoyLoLFile#1{%
  \directlua{%
    thirddata.joylol.loadFile('#1')
  }
}
```

MkIVCode : default

```
\definetyping[JoyLoLCode]
%\setuptyping[JoyLoLCode][option=lisp]

\let\oldStopJoyLoLCode=\stopJoyLoLCode
\def\stopJoyLoLCode{%
  \oldStopJoyLoLCode%
  \directlua{thirddata.joylol.core.context.evalBuffer('_typing_')}
```

```
}
```

LuaCode : default

```
local function evalBuffer(bufferName)
  local bufferContents =
    buffers.getcontent(bufferName):gsub("\13", "\n")
  --joylol.evalString(bufferContents)
end

joylol.core.context.evalBuffer = evalBuffer
```

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",       "Stephen Gaito"},
  { "commitDate",       "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",          "updated textadept lexer for JoyLoL"},
  { "notes",            ""},
  { NULL,                 NULL}
};
```

CCode : default

```
static int lua_core_context_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_core_context [] = {
  { "gitVersion",   lua_core_context_getGitVersion },
  {NULL, NULL}
};
```

CCode : default

```
static void coreContextWriteStdOut(
```

```
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  assert(jInterp);
  lua_State *lstate = jInterp->lstate;
  assert(lstate);

  lua_getglobal(lstate, "tex");
  lua_getfield(lstate, -1, "sprint");
  lua_pushstring(lstate, aMessage);
  if (lua_pcall(lstate, 1, 0, 0)) {
    /*return*/ luaL_error(lstate,
      "Failed of coreContextWriteStdOut\nERROR:\n%s\n",
      lua_tostring(lstate, -1)
    );
  }
  lua_pop(lstate, 1);
}

static void coreContextWriteStdErr(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  assert(jInterp);
  lua_State *lstate = jInterp->lstate;
  assert(lstate);

  lua_getglobal(lstate, "texio");
  lua_getfield(lstate, -1, "write");
  lua_pushstring(lstate, aMessage);
  if (lua_pcall(lstate, 1, 0, 0)) {
    /*return*/ luaL_error(lstate,
      "Failed of coreContextWriteStdErr\nERROR:\n%s\n",
      lua_tostring(lstate, -1)
    );
  }
  lua_pop(lstate, 1);
}

static Boolean getBooleanOption(
  lua_State *lstate,
  Symbol    *optionName
) {
```

```
43    lua_getglobal(lstate, "thirddata");
44    lua_getfield(lstate, -1, "joylol");
45    lua_getfield(lstate, -1, "options");
46    lua_getfield(lstate, -1, optionName);
47    Boolean aBool = (Boolean)lua_toboolean(lstate, -1);
48    lua_pop(lstate, 4);
49    return aBool;
50  }
51
52  static Symbol *getSymbolOption(
53    lua_State *lstate,
54    Symbol    *optionName
55  ) {
56    lua_getglobal(lstate, "thirddata");
57    lua_getfield(lstate, -1, "joylol");
58    lua_getfield(lstate, -1, "options");
59    lua_getfield(lstate, -1, optionName);
60    Symbol *aSymbol = (Symbol*)lua_tostring(lstate, -1);
61    lua_pop(lstate, 4);
62    return aSymbol;
63  }
64
65  static void *coreContextCallback(
66    lua_State *lstate,
67    size_t resourceId
68  ) {
69    if (resourceId == JoyLoLCallback_StdOutMethod) {
70      StdOutputMethod *coreWriteStdOut =
71        coreContextWriteStdOut;
72      return (void*)coreWriteStdOut;
73    } else if (resourceId == JoyLoLCallback_StdErrMethod) {
74      StdOutputMethod *coreWriteStdErr =
75        coreContextWriteStdErr;
76      return (void*)coreWriteStdErr;
77    } else if (resourceId == JoyLoLCallback_Verbose) {
78      Boolean verbose = getBooleanOption(lstate, "verbose");
79      return (void*)verbose;
80    } else if (resourceId == JoyLoLCallback_Trace) {
81      Boolean tracing = getBooleanOption(lstate, "tracing");
82      return (void*)tracing;
83    } else if (resourceId == JoyLoLCallback_Debug) {
84      Boolean debug = getBooleanOption(lstate, "debug");
85      return (void*)debug;
```

```
  } else if (resourceId == JoyLoLCallback_Quiet) {
    Boolean quiet = getBooleanOption(lstate, "quiet");
    return (void*)quiet;
  } else if (resourceId == JoyLoLCallback_ConfigFile) {
    Symbol *configFile = getSymbolOption(lstate, "configFile");
    return (void*)configFile;
  } else if (resourceId == JoyLoLCallback_UserPath) {
    Symbol *userPath = getSymbolOption(lstate, "userPath");
    return (void*)userPath;
  } else if (resourceId == JoyLoLCallback_LocalPath) {
    Symbol *localPath = getSymbolOption(lstate, "localPath");
    return (void*)localPath;
  } else if (resourceId == JoyLoLCallback_SystemPath) {
    Symbol *systemPath = getSymbolOption(lstate, "systemPath");
    return (void*)systemPath;
  }
  return NULL;
}
```

CCode : default

```
int luaopen_joylol_core_context (lua_State *lstate) {
  setJoyLoLCallbackFrom(lstate, coreContextCallback);
  lua_getglobal(lstate, "require");
  lua_pushstring(lstate, "joylol.jInterps");
  if (lua_pcall(lstate, 1, 1, 0)) {
    return luaL_error(lstate,
      "Failed to load [joylol.jInterps]\nERROR:\n%s\n",
      lua_tostring(lstate, -1)
    );
  }
  getJoyLoLInterpInto(lstate, jInterp);
  lua_pushstring(lstate, "core");
  lua_createtable(lstate,0, 1); // joylol.core
  lua_pushstring(lstate, "context");
  luaL_newlib(lstate, lua_core_context);
  lua_settable(lstate, -3);
  lua_settable(lstate, -3);
  return 1;
}
```

5.8 Preamble

MkIVCode : default

```
%D \module
%D   [     file=t-joylol,
%D      version=2017.05.10,
%D        title=\CONTEXT\ User module,
%D     subtitle=The JoyLoL prgramming language for \ConTeXt\,
%D       author=Stephen Gaito,
%D         date=\currentdate,
%D    copyright=PerceptiSys Ltd (Stephen Gaito),
%D        email=stephen@perceptisys.co.uk,
%D      license=MIT License]

%C Copyright (C) 2017 PerceptiSys Ltd (Stephen Gaito)
%C
%C Permission is hereby granted, free of charge, to any person obtaining a
%C copy of this software and associated documentation files (the
%C "Software"), to deal in the Software without restriction, including
%C without limitation the rights to use, copy, modify, merge, publish,
%C distribute, sublicense, and/or sell copies of the Software, and to
%C permit persons to whom the Software is furnished to do so, subject to
%C the following conditions:
%C
%C The above copyright notice and this permission notice shall be included
%C in all copies or substantial portions of the Software.
%C
%C THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
%C OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
%C MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
%C IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
%C CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
%C TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
%C SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

% begin info
%
% title   : JoyLoL CoAlgebra definitions
% comment : Provides structured document and code generation
% status  : under development, mkiv only
%
% end info
```

```

\unprotect

\ctxloadluafile{t-joylol}
```

MkIVCode : default

```
\protect \endinput
```

LuaCode : default

```
-- This is the lua code associated with t-joylol.mkiv

if not modules then modules = { } end modules ['t-joylol'] = {
    version   = 1.000,
    comment   = "joylol programming language - lua",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
    copyright = "PerceptiSys Ltd (Stephen Gaito)",
    license   = "MIT License"
}

local function setDefs(varVal, selector, defVal)
  if not defVal then defVal = { } end
  varVal[selector] = varVal[selector] or defVal
  return varVal[selector]
end

thirddata          = thirddata        or {}
local joylol       = setDefs(thirddata, 'joylol')
local options      = setDefs(joylol, 'options')

options.verbose    =
  options.verbose    or false
options.tracing    =
  options.tracing    or false
options.debug      =
  options.debug      or false
options.quiet      =
  options.quiet      or false
options.configFile =
  options.configFile or 'config'
options.userPath   =
  options.userPath   or os.getenv('HOME')..'/.joylol'
options.localPath  =
```

```
34    options.localPath  or '/usr/local/lib/joylol'
35  options.systemPath =
36    options.systemPath or '/usr/lib/joylol'
37  options.minimalJoylol =
38    options.minimalJoylol or false
39
40  local tInsert = table.insert
41  local tConcat = table.concat
42  local tRemove = table.remove
43  local tSort   = table.sort
44  local sFmt    = string.format
45  local sMatch  = string.match
46  local toStr   = tostring

47  local function compareKeyValues(a, b)
48    return (a[1] < b[1])
49  end
50
51  local function prettyPrint(anObj, indent)
52    local result = ""
53    indent = indent or ""
54    if type(anObj) == 'nil' then
55      result = 'nil'
56    elseif type(anObj) == 'boolean' then
57      if anObj then result = 'true' else result = 'false' end
58    elseif type(anObj) == 'number' then
59      result = toStr(anObj)
60    elseif type(anObj) == 'string' then
61      result = '"'..anObj..'"'
62    elseif type(anObj) == 'function' then
63      result = toStr(anObj)
64    elseif type(anObj) == 'userdata' then
65      result = toStr(anObj)
66    elseif type(anObj) == 'thread' then
67      result = toStr(anObj)
68    elseif type(anObj) == 'table' then
69      local origIndent = indent
70      indent = indent..'  '
71      result = '{\n'
72      for i, aValue in ipairs(anObj) do
73        result = result..indent..prettyPrint(aValue, indent)..',\n'
74      end
75      local theKeyValues = { }
```

```
76      for aKey, aValue in pairs(anObj) do
77        if type(aKey) ~= 'number' or aKey < 1 or #anObj < aKey then
78          tInsert(theKeyValues,
79            { prettyPrint(aKey), aKey, prettyPrint(aValue, indent) })
80        end
81      end
82      tSort(theKeyValues, compareKeyValues)
83      for i, aKeyValue in ipairs(theKeyValues) do
84        result = result..indent..'['..aKeyValue[1]..'] = '..aKeyValue[3]..',\n'
85      end
86      result = result..origIndent..'}'
87    else
88      result = 'UNKNOWN TYPE: ['..toStr(anObj)..']'
89    end
90    return result
91  end
```

5.9 Conclusion

CHeader : public

CHeader : private

CCode : default

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <assert.h>
4  #include <readline/readline.h>
5  #include <readline/history.h>
6  #include <joylol/jInterps.h>
7  #include <joylol/cFunctions.h>
8  #include <joylol/stringBuffers.h>
9  #include <joylol/dictNodes.h>
10 #include <joylol/texts.h>
11 #include <joylol/parsers.h>
12 #include <joylol/assertions.h>
13 #include <joylol/contexts.h>
14 #include <joylol/loaders.h>
15 //#include <joylol/core/context.h>
16 //#include <joylol/core/context-private.h>
17 // dictionary
18 // printer
```

Lmsfile : default

Lmsfile : default

616

616
616

5.10 The core JoyLoL based on Lua

618

618
618

5.11 Overview

This is the overview.

5.12 Lua main

LuaCode : default

```lua
local gitVersion = {
  authorName      = "Stephen Gaito",
  commitDate      = "2018-12-03",
  commitShortHash = "38e0564",
  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
  subject         = "updated textadept lexer for JoyLoL",
  notes           = ""
}
```

LuaCode : default

```lua
-- joylol command line interpreter

-- Start by dealing with the options

joylol = { }
joylol.options = { }
local options = joylol.options

-- default options
options.verbose    = false
options.debug      = false
options.quiet      = false
options.tracing    = false
options.loadFiles  = { }
options.loadPaths  = { }
options.configFile = 'config'
options.userPath   = os.getenv('HOME')..'/.joylol'
options.localPath  = '/usr/local/lib/joylol'
options.systemPath = '/usr/lib/joylol'

helpText = {
  "usage: joylol [options] [files to load]",
  "",
  "options: ",
  " -h --help        prints this help text and exits",
  " -i --ignore      ignores default configuration file (~.joylol)",
  " -l --load <file> loads the file <file>",
  " -p --path <path> adds <path> to the list of load paths",
  " -q --quiet       toggles verbose off",
```

```
30    " -v --verbose     toggles verbose on",
31    " -t --trace       toggles tracing on",
32    " -d --debug       turns on internal debugging",
33    " -u --userPath    sets the user load path",
34    " -L --localPath   sets the local load path",
35    " -s --systemPath  sets the system load path",
36    "",
37    "files to load:",
38    "  Any remaining options are treated as files to be loaded.",
39    "  If there are no remaining options, joylol enters the read,",
40    "  eval, print loop."
41  }
42
43  while(0 < #arg) do
44    anArg = table.remove(arg, 1)
45    if anArg:match('-h') or anArg:match('--help') then
46      print(table.concat(helpText, '\n'))
47      os.exit(0);
48    elseif anArg:match('-c') or anArg:match('--config') then
49      optArg = table.remove(arg, 1)
50      options.configFile = optArg
51    elseif anArg:match('-i') or anArg:match('--ignore') then
52      options.configFile = nil
53    elseif anArg:match('-u') or anArg:match('--user') then
54      optArg = table.remove(arg, 1)
55      options.userPath = optArg
56    elseif anArg:match('-L') or anArg:match('--local') then
57      optArg = table.remove(arg, 1)
58      options.localPath = optArg
59    elseif anArg:match('-s') or anArg:match('--system') then
60      optArg = table.remove(arg, 1)
61      options.systemPath = optArg
62    elseif anArg:match('-l') or anArg:match('--load') then
63      optArg = table.remove(arg, 1)
64      table.insert(options.loadFiles, optArg)
65    elseif anArg:match('-p') or anArg:match('--path') then
66      optArg = table.remove(arg, 1)
67      table.insert(options.loadPaths, optArg)
68    elseif anArg:match('-q') or anArg:match('--quiet') then
69      options.verbose = false
70      options.debug   = false
71      options.tracing = false
72      options.quiet   = true
```

```
73    elseif anArg:match('-v') or anArg:match('--verbose') then
74      options.verbose = true
75    elseif anArg:match('-d') or anArg:match('--debug') then
76      options.debug = true
77    elseif anArg:match('-t') or anArg:match('--trace') then
78      options.tracing = true
79    else
80      --optArg = table.remove(arg, 1)
81      table.insert(options.loadFiles, anArg)
82    end
83  end
84
85  -- Now add the standard joylol CoAlg locations to
86  -- the Lua search paths
87
88  local joylolPaths = {
89    options.userPath..'/?.lua',
90    options.localPath..'/?.lua',
91    options.systemPath..'/?.lua',
92    package.path
93  }
94  package.path = table.concat(joylolPaths, ';')
95
96  local joylolCPaths = {
97    options.userPath..'/?.so',
98    options.localPath..'/?.so',
99    options.systemPath..'/?.so',
100   package.path
101 }
102 package.cpath = table.concat(joylolCPaths, ';')
103
104 -- Now load joylol
105
106 if options.verbose then print('loading [joylol.core.lua]') end
107 joylol = require 'joylol.core.lua'
108 if options.verbose then print('loaded [joylol.core.lua]\n') end
109
110 -- replace the options
111
112 joylol.options = options
113
114 --  deal with loadPaths, configFile, and loadFiles
115
```

```
for i, aPath in ipairs(options.loadPaths) do
  if options.verbose then print('adding loadPath ['..aPath..']\n') end
  joylol.pushLoadPath(aPath)
end

if (options.configFile) then
  joylol.loadFile(options.configFile)
end

for i, aFile in ipairs(options.loadFiles) do
  if options.verbose then print('loading ['..aFile..']\n') end
  joylol.loadFile(aFile)
  if options.verbose then print('loaded ['..aFile..']\n') end
end

if #options.loadFiles < 1 then
  -- Finally run the REPL
  joylol.core.lua.runREPL();
end
```

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

5.12.1 Readline code

CCode : default

```
// Texts are a collection of characters, which are used by the Parser
// to extract successive symbols.
//
// Texts are created on one of three backing suppliers of characters:
// 1. a single string
// 2. a NULL terminated array of strings
// 3. an external file
```

```
// 4. a readline interaction with a user
//
// In all four cases, the Parser's nextSymbol method requests successive
// **lines** of characters (deliminated by new-line-characters).
//
// It is critical, for correct interaction with the user via readline,
// that the initial line is NOT obtained until actually requested by
// the parser's nextSymbol method.
//
// It is also critical that once completed, none of the sources, get
// asked for subsequent lines.
//
// When the text has been completed, the nextLine function ensures
// that aText->curLine is NULL.
```

CCode : default

```
static void clearReadlinePrompt(TextObj* aText) {
  aText->curPrompt = aText->newPrompt;
}

static void setReadlinePrompts(TextObj* aText,
                        const char* newPrompt,
                        const char* continuePrompt
) {
  if (newPrompt) aText->newPrompt = newPrompt;
  else           aText->newPrompt = ">";

  if (continuePrompt) aText->continuePrompt = continuePrompt;
  else                aText->continuePrompt = ":";

  clearReadlinePrompt(aText);
}
```

CCode : default

```
static void saveReadlineHistory(TextObj* aText) {
  write_history(".joyLoL-history");
}
```

CCode : default

```
static void nextLineFromReadline(TextObj* aText) {
  assert(aText);
  assert(aText->jInterp);
```

```
4   DEBUG(aText->jInterp, "->nextLineFromReadline %s\n", "");
5   if (!aText) return; // there is nothing we can do!
6
7   if (aText->completed) {
8     // we have reached the end of the interaction with the user
9     aText->completed = TRUE;
10    aText->curLine   = NULL;
11    aText->curChar   = NULL;
12    aText->lastChar  = NULL;
13    return;
14  }
15
16  // readline returns alloc'ed memory so we free it here.
17  if (aText->curLine) free((void*)aText->curLine);
18
19  aText->curLine  = NULL;
20  aText->curChar  = NULL;
21  aText->lastChar = NULL;
22
23  aText->curLine = readline (aText->curPrompt);
24  aText->curLineNum++;

25  if (!aText->curLine) {
26    aText->completed = TRUE;
27    return;
28  }
29
30  if (*aText->curLine) {
31    history_set_pos(0); // start at the beginning
32    if (0 <= history_search (aText->curLine, -1)) { // search backwards
33      remove_history(where_history());
34    } else if (0 <= history_search (aText->curLine, 1)) { // search forwards
35      remove_history(where_history());
36    }
37    history_set_pos(0);
38    add_history (aText->curLine);
39  }
40
41  aText->curChar  = aText->curLine;
42  aText->lastChar = aText->curChar + strlen(aText->curChar);
43
44  aText->curPrompt = aText->continuePrompt;
```

```
45 }
```

CCode : default

```
1 static TextObj* currentReadLineText = NULL;
```

CCode : default

```
1  static char* dictionaryWalker(const char* text, int state) {
2    if (!currentReadLineText) return NULL;

3    JoyLoLInterp *jInterp = currentReadLineText->jInterp;
4    assert(jInterp);

5    if (!state) {
6
7      ContextObj *rootCtx = jInterp->rootCtx;
8      assert(rootCtx);
9
10     currentReadLineText->curNode =
11       findLUBSymbol(rootCtx->dict, text);
12     DEBUG(jInterp,
13           "dictionaryWalker-start %p\n",
14           currentReadLineText->curNode);
15   }
16   DictNodeObj* curNode = currentReadLineText->curNode;
17
18   if (!curNode) return NULL;
19
20   if (strncmp(curNode->symbol, text, strlen(text)) == 0) {
21     DEBUG(jInterp,
22           "dictionaryWalker %p {%s}[%s]\n",
23           curNode, text, curNode->symbol);
24     currentReadLineText->curNode = curNode->next;
25     return strdup(curNode->symbol);
26   }
27
28   currentReadLineText->curNode = NULL;
29   return NULL;
30 }
31
32 static char** dictionaryCompletion(const char* text, int start, int end)
33 {
34  return rl_completion_matches(text, dictionaryWalker);
```

```
}
```

CCode : default

```
static TextObj* createTextFromReadline(JoyLoLInterp *jInterp) {

  assert(jInterp);
  DEBUG(jInterp, "createTextFromReadline %p\n", jInterp);

  TextObj* aText = (TextObj*)newObject(jInterp, TextsTag);
  assert(aText);
  //
  // readline specific initializations
  //
  DEBUG(jInterp, "starting readline initialization %s\n", "");
  using_history();
  read_history(".joyLoL-history");
  rl_readline_name = "joyLoL";
  rl_attempted_completion_function = dictionaryCompletion;
  setReadlinePrompts(aText, NULL, NULL);
  aText->curNode = NULL;
  aText->curCompletionText = NULL;
  aText->curCompletionLen  = 0;
  aText->nextLine = nextLineFromReadline;
  //
  // general initialization
  //

  aText->jInterp    = jInterp;
  aText->completed  = FALSE;
  aText->sym        = NULL;
  aText->curLine    = NULL;
  aText->curChar    = NULL;
  aText->lastChar   = NULL;
  //
  aText->inputFile = NULL;
  aText->fileName  = strdup("readline");
  //
  aText->textLines  = NULL;
  aText->curLineNum = 0;

  assert(!currentReadLineText); // there should not be more than one
  currentReadLineText = aText;

```

```
39    return aText;
40  }
```

CHeader : public

```
1   extern void runREPLInContext(
2     ContextObj* aCtx
3   );
```

CCode : default

```
1   void runREPLInContext(
2     ContextObj* aCtx
3   ) {
4     assert(aCtx);
5     JoyLoLInterp *jInterp = aCtx->jInterp;
6     assert(jInterp);

7   //  if (aCtx->verbose) {
8       getGitVersionInto(gitVersionKeyValues, "commitShortHash", commitHash);
9       getGitVersionInto(gitVersionKeyValues, "commitDate",      commitDate);
10      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
11      strBufPrintf(aStrBuf, "Welcome to JoyLoL v0.1 ( %s ; %s )\n",
12        commitHash,
13        commitDate
14      );
15      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
16      strBufClose(aStrBuf);
17  //  }
18    aCtx->showStack = TRUE;
19    TextObj* aText = createTextFromReadline(jInterp);
20    assert(aText);

21    while(TRUE) {
22      DEBUG(jInterp, "runREPLInContext %p reading\n", aCtx);
23      JObj* aLoL = parseOneSymbol(aText);
24      DEBUG(jInterp, "runREPLInContext %p read %p\n", aCtx, aLoL);
25      //
26      if (aLoL && isSignal(aLoL) &&
27        asSignal(aLoL) == SIGNAL_END_OF_TEXT) break;
28      //
29      evalCommandInContext(aCtx, aLoL);
30      assert(!aCtx->process);
31      //
```

```
    if (aCtx->showStack) {
      DEBUG(jInterp, "runREPLInContext %p printing data %p\n", aCtx, aCtx->data);
      StringBufferObj *aStrBuf = newStringBuffer(aCtx);
      strBufPrintf(aStrBuf, ">>");
      printLoL(aStrBuf, aCtx->data);
      strBufPrintf(aStrBuf, "\n");
      jInterp->writeStdOut(jInterp, getCString(aStrBuf));
      strBufClose(aStrBuf);
      DEBUG(jInterp, "runREPLInContext %p printed data %p\n", aCtx, aCtx->data);
    }
    //
    reportException(aCtx); // report last exception if raised
  }
  saveReadlineHistory(aText);
  jInterp->writeStdOut(jInterp, "\n");
  DEBUG(jInterp, "runREPLInContext %p COMPLETED\n", aCtx);
}
```

```
static void showVersionsAP(Context* aCtx) {
  fprintf(stdout, "\n");
  reportMainVersions(stdout);
  fprintf(stdout, "\n");
  reportLibVersions(stdout);
  fprintf(stdout, "\n");
}
```

5.12.2 Lua interface

CCode : default

```
static int lua_core_lua_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static int lua_core_lua_runREPL(lua_State *lstate) {
  getJoyLoLInterpInto(lstate, jInterp);
```

```
14    runREPLInContext(jInterp->rootCtx);
15    return 0;
16  }
17
18  static const struct luaL_Reg lua_core_lua [] = {
19    { "gitVersion",   lua_core_lua_getGitVersion },
20    { "runREPL",      lua_core_lua_runREPL       },
21    {NULL, NULL}
22  };
23
24  static void coreLuaWriteStdOut(
25    JoyLoLInterp *jInterp,
26    Symbol       *aMessage
27  ) {
28    fprintf(stdout, "%s", aMessage);
29  }
30
31  static void coreLuaWriteStdErr(
32    JoyLoLInterp *jInterp,
33    Symbol       *aMessage
34  ) {
35    fprintf(stderr, "%s", aMessage);
36  }
37
38  static Boolean getBooleanOption(
39    lua_State *lstate,
40    Symbol    *optionName
41  ) {
42    lua_getglobal(lstate, "joylol");
43    lua_getfield(lstate, -1, "options");
44    lua_getfield(lstate, -1, optionName);
45    Boolean aBool = (Boolean)lua_toboolean(lstate, -1);
46    lua_pop(lstate, 3);
47    return aBool;
48  }
49
50  static Symbol *getSymbolOption(
51    lua_State *lstate,
52    Symbol    *optionName
53  ) {
54    lua_getglobal(lstate, "joylol");
55    lua_getfield(lstate, -1, "options");
56    lua_getfield(lstate, -1, optionName);
```

```
57    Symbol *aSymbol = (Symbol*)lua_tostring(lstate, -1);
58    lua_pop(lstate, 3);
59    return aSymbol;
60  }
61
62  static void *coreLuaCallback(
63    lua_State *lstate,
64    size_t resourceId
65  ) {
66    if (resourceId == JoyLoLCallback_StdOutMethod) {
67      StdOutputMethod *coreWriteStdOut =
68        coreLuaWriteStdOut;
69      return (void*)coreWriteStdOut;
70    } else if (resourceId == JoyLoLCallback_StdErrMethod) {
71      StdOutputMethod *coreWriteStdErr =
72        coreLuaWriteStdErr;
73      return (void*)coreWriteStdErr;
74    } else if (resourceId == JoyLoLCallback_Verbose) {
75      Boolean verbose = getBooleanOption(lstate, "verbose");
76      return (void*)verbose;
77    } else if (resourceId == JoyLoLCallback_Debug) {
78      Boolean debug = getBooleanOption(lstate, "debug");
79      return (void*)debug;
80    } else if (resourceId == JoyLoLCallback_Trace) {
81      Boolean tracing = getBooleanOption(lstate, "tracing");
82      return (void*)tracing;
83    } else if (resourceId == JoyLoLCallback_Quiet) {
84      Boolean quiet = getBooleanOption(lstate, "quiet");
85      return (void*)quiet;
86    } else if (resourceId == JoyLoLCallback_ConfigFile) {
87      Symbol *configFile = getSymbolOption(lstate, "configFile");
88      return (void*)configFile;
89    } else if (resourceId == JoyLoLCallback_UserPath) {
90      Symbol *userPath = getSymbolOption(lstate, "userPath");
91      return (void*)userPath;
92    } else if (resourceId == JoyLoLCallback_LocalPath) {
93      Symbol *localPath = getSymbolOption(lstate, "localPath");
94      return (void*)localPath;
95    } else if (resourceId == JoyLoLCallback_SystemPath) {
96      Symbol *systemPath = getSymbolOption(lstate, "systemPath");
97      return (void*)systemPath;
98    }
99    return NULL;
```

```
}

int luaopen_joylol_core_lua (lua_State *lstate) {
  setJoyLoLCallbackFrom(lstate, coreLuaCallback);
  lua_getglobal(lstate, "require");
  lua_pushstring(lstate, "joylol.jInterps");
  if (lua_pcall(lstate, 1, 1, 0)) {
    return luaL_error(lstate,
      "Failed to load [joylol.jInterps]\nERROR:\n%s\n",
      lua_tostring(lstate, -1)
    );
  }
  getJoyLoLInterpInto(lstate, jInterp);
  lua_pushstring(lstate, "core");
  lua_createtable(lstate,0, 1); // joylol.core
  lua_pushstring(lstate, "lua");
  luaL_newlib(lstate, lua_core_lua);
  lua_settable(lstate, -3);
  lua_settable(lstate, -3);
  return 1;
}
```

5.12.3 Conclusions

CHeader : public

CHeader : private

CCode : default

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <readline/readline.h>
#include <readline/history.h>
#include <joylol/jInterps.h>
#include <joylol/cFunctions.h>
#include <joylol/signals.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
#include <joylol/dictionaries.h>
#include <joylol/texts.h>
#include <joylol/parsers.h>
#include <joylol/assertions.h>
```

```
#include <joylol/contexts.h>
#include <joylol/loaders.h>
#include <joylol/core/lua.h>
#include <joylol/core/lua-private.h>
// dictionary
// printer
```

Lmsfile : default

```
local joylolTarget = makePath{getEnv('HOME'), 'bin', 'joylol'}
local joylolDep    = makePath{'buildDir', 'joylol.lua'}
tInsert(installTargets, target{
  target       = joylolTarget,
  dependencies = { joylolDep },
  command      = tConcat({'install -T', joylolDep, joylolTarget }, ' ')
})
```

Lmsfile : default

Lmsfile : default

5.13 The core JoyLoL embedded in Textadept

636

636
636

5.14 Overview

This is the overview.

5.14.1 Textadept lexer for JoyLoL

LuaCode : lexer

```
-- joyLoL LPeg lexer.

local lexer = require('lexer')
local token, word_match = lexer.token, lexer.word_match
local P, R, S = lpeg.P, lpeg.R, lpeg.S

local lex = lexer.new('joylol')

-- Whitespace.
lex:add_rule('whitespace', token(lexer.WHITESPACE, lexer.space^1))

-- Keywords.
lex:add_rule('keyword', token(lexer.KEYWORD, word_match[[  CoAlgebra EndCoAlgebra
  Invariant EndInvariant
  Lexer EndLexer
  Parser EndParser
  Structure EndStructure
  Method EndMethod
  PreDataStack EndPreDataStack
  PreProcessStack EndPreProcessStack
  PreCondition EndPreCondition
  RMCode EndRMCode
  PostDataStack EndPostDataStack
  PostProcessStack EndPostProcessStack
  PostCondition EndPostCondition
]]))

-- Identifiers.
lex:add_rule('identifier', token(lexer.IDENTIFIER, lexer.word))

-- Strings.
lex:add_rule('string', token(lexer.STRING, lexer.delimited_range("'")
+
                                           lexer.delimited_range('"')))

```

```
36  -- Comments.
37  local line_comment = '//' * lexer.nonnewline_esc^0
38  local block_comment = '/*' * (lexer.any - '*/')^0 * P('*/')^-1
39
40  lex:add_rule('comment', token(lexer.COMMENT, line_comment + block_comment))
41  --lex:add_rule('comment', token(lexer.COMMENT, '#' * lexer.nonnewline^0))
42
43  -- Numbers.
44  lex:add_rule('number', token(lexer.NUMBER, lexer.float + lexer.integer))
45
46  -- Operators.
47  lex:add_rule('operator', token(lexer.OPERATOR, S('+-*/%^=<>,.{}[]()')))
48
49  -- Fold points.
50  -- lex:add_fold_point(lexer.KEYWORD, 'start', 'end')
51  lex:add_fold_point(lexer.KEYWORD, 'CoAlgebra',         'EndCoAlgebra')
52  lex:add_fold_point(lexer.KEYWORD, 'Invariant',         'EndInvariant')
53  lex:add_fold_point(lexer.KEYWORD, 'Lexer',             'EndLexer')
54  lex:add_fold_point(lexer.KEYWORD, 'Parser',            'EndParser')
55  lex:add_fold_point(lexer.KEYWORD, 'Structure',         'EndStructure')
56  lex:add_fold_point(lexer.KEYWORD, 'Method',            'EndMethod')
57  lex:add_fold_point(lexer.KEYWORD, 'PreDataStack',      'EndPreDataStack')
58  lex:add_fold_point(lexer.KEYWORD, 'PreProcessStack',   'EndPreProcessStack')
59  lex:add_fold_point(lexer.KEYWORD, 'PreCondition',      'EndPreCondition')
60  lex:add_fold_point(lexer.KEYWORD, 'RMCode',            'EndRMCode')
61  lex:add_fold_point(lexer.KEYWORD, 'PostDataStack',     'EndPostDataStack')
62  lex:add_fold_point(lexer.KEYWORD, 'PostProcessStack', 'EndPostProcessStack')
63  lex:add_fold_point(lexer.KEYWORD, 'PostCondition',     'EndPostCondition')
64
65  lex:add_fold_point(lexer.OPERATOR, '{', '}')
66  lex:add_fold_point(lexer.OPERATOR, '(', ')')
67  lex:add_fold_point(lexer.OPERATOR, '[', ']')
68  lex:add_fold_point(lexer.COMMENT, '//', lexer.fold_line_comments('//'))
69  lex:add_fold_point(lexer.COMMENT, '/*', '*/')
70  return lex
```

5.14.2 JoyLoL load options

LuaCode : options

```
1  local jOpts = { }
2
3  joylol         = joylol         or { }
```

```
joylol.options = joylol.options or { }

local options  = joylol.options
```

LuaCode : options

```
local gitVersion = {
  authorName      = "Stephen Gaito",
  commitDate      = "2018-12-03",
  commitShortHash = "38e0564",
  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
  subject         = "updated textadept lexer for JoyLoL",
  notes           = ""
}
```

LuaCode : options

```
options.gitVersion = gitVersion
```

LuaCode : options

```
-- joylol loader options

options.verbose = false

function jOpts.beVerbose()
  options.verbose = true
end

options.debug = false

function jOpts.debug()
  options.debug = true
end

options.tracing = false

function jOpts.trace()
  options.tracing = true
end

function jOpts.beQuiet()
  options.verbose = false
  options.debug   = false
  options.tracing = false
```

```
25  end
26
27  options.configFile = 'config'
28
29  function jOpts.setConfigFile(aConfigFile)
30    options.configFile = aConfigFile
31  end
32
33  function jOpts.noConfiguration()
34    options.configFile = nil
35  end
36
37  options.userPath = os.getenv('HOME')..'/.joylol'
38
39  function jOpts.setUserPath(aUserPath)
40    options.userPath = aUserPath
41  end
42
43  options.localPath = '/usr/local/lib/joylol'
44
45  function jOpts.setLocalPath(aLocalPath)
46    options.localPath = aLocalPath
47  end
48
49  options.systemPath = '/usr/lib/joylol'
50
51  function jOpts.setSystemPath(aSystemPath)
52    options.systemPath = aSystemPath
53  end
54
55  return jOpts
```

5.14.3 Textadept installation for JoyLoL language

LuaCode : install

```
1  -- Ensure files whose extensions are either
2  -- 'joy' or 'joylol' are interpreted as JoyLoL files.
3  textadept.file_types.extensions['joy']    = 'joylol'
4  textadept.file_types.extensions['joylol'] = 'joylol'
5
6  -- Ensure files whose first lines contain the word
7  -- 'JoyLoL' are interpreted as JoyLoL files.
```

```
textadept.file_types.patterns['%s[Jj][Oo][Yy][Ll][Oo][Ll]%s'] = 'joylol'
```

Textadept installation for JoyLoL language 5.14.3

5.15 Textadept language module initialization

LuaCode : init

```lua
local gitVersion = {
  authorName      = "Stephen Gaito",
  commitDate      = "2018-12-03",
  commitShortHash = "38e0564",
  commitLongHash  = "38e0564bfc658bcd3257d07cc085a247a396c83f",
  subject         = "updated textadept lexer for JoyLoL",
  notes           = ""
}
```

LuaCode : init

```lua
-- joylol command line interpreter

-- load the options
joylol             = joylol         or { }
joylol.options     = joylol.options or { }
local options      = joylol.options

options.verbose    =
  options.verbose    or false
options.debug      =
  options.debug      or false
options.configFile =
  options.configFile or 'config'
options.userPath   =
  options.userPath   or os.getenv('HOME')..'/.joylol'
options.localPath  =
  options.localPath  or '/usr/local/lib/joylol'
options.systemPath =
  options.systemPath or '/usr/lib/joylol'

-- Start by adding the standard joylol CoAlg locations to the Lua search
-- paths

local joylolPaths = {
  options.userPath..'/?.lua',
  options.localPath..'/?.lua',
  options.systemPath..'/?.lua',
  package.path
}
```

```
30  package.path = table.concat(joylolPaths, ';')
31
32  local joylolCPaths = {
33    options.userPath..'/?.so',
34    options.localPath..'/?.so',
35    options.systemPath..'/?.so',
36    package.path
37  }
38  package.cpath = table.concat(joylolCPaths, ';')
39
40  if options.verbose then print('loading [joylol.core.textadept]') end
41  joylol = require 'joylol.core.textadept'
42  if options.verbose then print('loaded [joylol.core.textadept]\n') end
43
44  joylol.options = options
45
46  --joylol.setVerbose(options.verbose)
47  --joylol.setDebugging(options.debug)
48
49  if (options.configFile) then
50    joylol.loadFile(options.configFile)
51  end
52
53  -- Initialization for the JoyLoL language
54
55  -- add in luatex specific key codes
56  keys['joylol'] = keys.context or {}
57  -- keys.joylol.cg = require('context/ctags').goto_symbol    -- Ctrl-g
58  --keys.joylol[not OSX and (GUI and 'cR' or 'cmr') or 'mR'] = require('common/messageBuffer')
59  keys.joylol['cR'] = require('common/messageBuffer').clearRunCompile
60
61  -- tell textadept how to compile/run joylol files
62  textadept.run.compile_commands.joylol = 'joylol "%f"'
63  textadept.run.run_commands.joylol     = 'joylol "%f"'
64  -- add comment string for context
65  textadept.editing.comment_string.joylol = ';'
66
67  require 'joylol.repl'
68
69  return { }
```

5.15.1 JoyLoL REPL

LuaCode : repl

```lua
-- A JoyLoL REPL

-- Based upon Mitchell's Lua REPL
-- see: https://foicica.com/wiki/lua-repl
-- on: 2017-02-06

local function newJoyLoLREPL()
  buffer.new()._type = '[JoyLoL REPL]'
  buffer:set_lexer('joylol')
  buffer:set_text(';; JoyLoL REPL')
  buffer:document_end()
  buffer:new_line()
  buffer:new_line()
  buffer:set_save_point()
end

local function evaluateJoyLoLREPL()
  local selStart = buffer.selection_start
  local selEnd   = buffer.selection_end
  local code
  local lastLine
  if selStart ~= selEnd then
    local startLine = buffer:line_from_position(selStart)
    selStart = buffer:position_from_line(startLine)
    local endLine   = buffer:line_from_position(selEnd)
    if buffer.column[selEnd] > 0 then
      selEnd = buffer:position_from_line(endLine+1)
    end
    code = buffer:text_range(selStart, selEnd)
  else
    code = buffer:get_cur_line()
  end

  buffer:document_end()
  buffer:new_line()
  joylol.evalString(code)
  buffer:new_line()
  buffer:set_save_point()
end

```

```
keys.joylol = keys.joylol or { }
keys.joylol['c>'] = newJoyLoLREPL
keys.joylol['c\n'] = function()
  if buffer._type ~= '[JoyLoL REPL]' then
    -- propagate key event to next handler
    return false
  end
  evaluateJoyLoLREPL()
end

-- add creation of a JoyLoL-REPL to context
keys.context = keys.context or { }
keys.context['c>'] = newJoyLoLREPL
```

5.15.2 Lua interface

CCode : default

```
static const KeyValues gitVersionKeyValues[] = {
  { "authorName",      "Stephen Gaito"},
  { "commitDate",      "2018-12-03"},
  { "commitShortHash", "38e0564"},
  { "commitLongHash",  "38e0564bfc658bcd3257d07cc085a247a396c83f"},
  { "subject",         "updated textadept lexer for JoyLoL"},
  { "notes",           ""},
  { NULL,                NULL}
};
```

CCode : default

```
static int lua_core_textadept_getGitVersion (lua_State *lstate) {
  const char* aKey   = lua_tostring(lstate, 1);
  if (aKey) {
    getGitVersionInto(gitVersionKeyValues, aKey, aValue);
    lua_pushstring(lstate, aValue);
  } else {
    lua_pushstring(lstate, "no valid key provided");
  }
  return 1;
}

static const struct luaL_Reg lua_core_textadept [] = {
  { "gitVersion",   lua_core_textadept_getGitVersion },
```

```
14    {NULL, NULL}
15  };
16
17  static void coreTextadeptWriteStdOut(
18    JoyLoLInterp *jInterp,
19    Symbol       *aMessage
20  ) {
21    assert(jInterp);
22    lua_State *lstate = jInterp->lstate;
23    assert(lstate);
24
25    lua_getglobal(lstate, "buffer");
26    lua_getfield(lstate, -1, "append_text");
27    lua_getglobal(lstate, "buffer");
28    lua_pushstring(lstate, aMessage);
29    if (lua_pcall(lstate, 2, 0, 0)) {
30      /*return*/ luaL_error(lstate,
31        "Failed to append message [%s] to current buffer\nERROR:\n%s\n",
32        aMessage,
33        lua_tostring(lstate, -1)
34      );
35    }
36    lua_pop(lstate, 1);
37  }
38
39  static void coreTextadeptWriteStdErr(
40    JoyLoLInterp *jInterp,
41    Symbol       *aMessage
42  ) {
43    assert(jInterp);
44    lua_State *lstate = jInterp->lstate;
45    assert(lstate);
46
47    lua_getglobal(lstate, "ui");
48    lua_getfield(lstate, -1, "print");
49    lua_pushstring(lstate, aMessage);
50    if (lua_pcall(lstate, 1, 0, 0)) {
51      /*return*/ luaL_error(lstate,
52        "Failed to append message [%s] to Message Buffer\nERROR:\n%s\n",
53        aMessage,
54        lua_tostring(lstate, -1)
55      );
56    }
```

```
57    lua_pop(lstate, 1);
58  }
59
60  static Boolean getBooleanOption(
61    lua_State *lstate,
62    Symbol    *optionName
63  ) {
64    lua_getglobal(lstate, "joylol");
65    lua_getfield(lstate, -1, "options");
66    lua_getfield(lstate, -1, optionName);
67    Boolean aBool = (Boolean)lua_toboolean(lstate, -1);
68    lua_pop(lstate, 3);
69    return aBool;
70  }
71
72  static Symbol *getSymbolOption(
73    lua_State *lstate,
74    Symbol    *optionName
75  ) {
76    lua_getglobal(lstate, "joylol");
77    lua_getfield(lstate, -1, "options");
78    lua_getfield(lstate, -1, optionName);
79    Symbol *aSymbol = (Symbol*)lua_tostring(lstate, -1);
80    lua_pop(lstate, 3);
81    return aSymbol;
82  }
83
84  static void *coreTextadeptCallback(
85    lua_State *lstate,
86    size_t resourceId
87  ) {
88    if (resourceId == JoyLoLCallback_StdOutMethod) {
89      StdOutputMethod *coreWriteStdOut =
90        coreTextadeptWriteStdOut;
91      return (void*)coreWriteStdOut;
92    } else if (resourceId == JoyLoLCallback_StdErrMethod) {
93      StdOutputMethod *coreWriteStdErr =
94        coreTextadeptWriteStdErr;
95      return (void*)coreWriteStdErr;
96    } else if (resourceId == JoyLoLCallback_Verbose) {
97      Boolean verbose = getBooleanOption(lstate, "verbose");
98      return (void*)verbose;
99    } else if (resourceId == JoyLoLCallback_Trace) {
```

```
    Boolean tracing = getBooleanOption(lstate, "tracing");
    return (void*)tracing;
  } else if (resourceId == JoyLoLCallback_Debug) {
    Boolean debug = getBooleanOption(lstate, "debug");
    return (void*)debug;
  } else if (resourceId == JoyLoLCallback_Quiet) {
    Boolean quiet = getBooleanOption(lstate, "quiet");
    return (void*)quiet;
  } else if (resourceId == JoyLoLCallback_ConfigFile) {
    Symbol *configFile = getSymbolOption(lstate, "configFile");
    return (void*)configFile;
  } else if (resourceId == JoyLoLCallback_UserPath) {
    Symbol *userPath = getSymbolOption(lstate, "userPath");
    return (void*)userPath;
  } else if (resourceId == JoyLoLCallback_LocalPath) {
    Symbol *localPath = getSymbolOption(lstate, "localPath");
    return (void*)localPath;
  } else if (resourceId == JoyLoLCallback_SystemPath) {
    Symbol *systemPath = getSymbolOption(lstate, "systemPath");
    return (void*)systemPath;
  }
  return NULL;
}

int luaopen_joylol_core_textadept (lua_State *lstate) {
  printf("luaopen_joylol_core_textadept start\n");
  setJoyLoLCallbackFrom(lstate, coreTextadeptCallback);
  lua_getglobal(lstate, "require");
  lua_pushstring(lstate, "joylol.jInterps");
  if (lua_pcall(lstate, 1, 1, 0)) {
    return luaL_error(lstate,
      "Failed to load [joylol.jInterps]\nERROR:\n%s\n",
      lua_tostring(lstate, -1)
    );
  }
  getJoyLoLInterpInto(lstate, jInterp);
  lua_pushstring(lstate, "core");
  lua_createtable(lstate,0, 1); // joylol.core
  lua_pushstring(lstate, "textadept");
  luaL_newlib(lstate, lua_core_textadept);
  lua_settable(lstate, -3);
  lua_settable(lstate, -3);
  printf("luaopen_joylol_core_textadept done\n");
```

```
  return 1;
}
```

5.15.3 Conclusions

CHeader : public

CHeader : private

CCode : default

```c
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <readline/readline.h>
#include <readline/history.h>
#include <joylol/jInterps.h>
#include <joylol/cFunctions.h>
#include <joylol/stringBuffers.h>
#include <joylol/dictNodes.h>
#include <joylol/texts.h>
#include <joylol/parsers.h>
#include <joylol/assertions.h>
#include <joylol/contexts.h>
#include <joylol/loaders.h>
#include <joylol/core/textadept.h>
#include <joylol/core/textadept-private.h>
// dictionary
// printer
```

Lmsfile : default

```
local joylolTarget =
  makePath{getEnv('HOME'), '.textadept', 'lexers', 'joylol.lua'}
local joylolDep    = makePath{'buildDir', 'joylol.lua'}
tInsert(installTargets, target{
  target       = joylolTarget,
  dependencies = { joylolDep },
  command      = tConcat({'install -T', joylolDep, joylolTarget }, ' ')
})

local joylolModuleDir =
  makePath{getEnv('HOME'), '.textadept', 'modules', 'joylol'}
tInsert(installTargets, target{
```

```
13    target       = joylolModuleDir,
14    dependencies = { },
15    command      = tConcat({'install --directory', joylolModuleDir }, '
16  ')
17  })
18
19  local installTarget =
20    makePath{ joylolModuleDir, 'installJoyLoLLangauge.lua'}
21  local installDep    = makePath{'buildDir', 'installJoyLoLLanguage.lua'}
22  tInsert(installTargets, target{
23    target       = installTarget,
24    dependencies = { installDep },
25    command      = tConcat({'install -T', installDep, installTarget }, '
26  ')
27  })
28
29  local initTarget =
30    makePath{ joylolModuleDir, 'init.lua'}
31  local initDep    = makePath{'buildDir', 'init.lua'}
32  tInsert(installTargets, target{
33    target       = initTarget,
34    dependencies = { initDep },
35    command      = tConcat({'install -T', initDep, initTarget }, ' ')
36  })
37
38  local optsTarget =
39    makePath{ joylolModuleDir, 'options.lua'}
40  local optsDep    = makePath{'buildDir', 'options.lua'}
41  tInsert(installTargets, target{
42    target       = optsTarget,
43    dependencies = { optsDep },
44    command      = tConcat({'install -T', optsDep, optsTarget }, ' ')
45  })
46
47  local replTarget =
48    makePath{ joylolModuleDir, 'repl.lua'}
49  local replDep    = makePath{'buildDir', 'repl.lua'}
50  tInsert(installTargets, target{
51    target       = replTarget,
52    dependencies = { replDep },
53    command      = tConcat({'install -T', replDep, replTarget }, ' ')
54  })
```

Lmsfile : default

Lmsfile : default

653

6 ConTeXt

653
653

654

654
654

6.1 JoyLoL CoAlgebraic Extensions ConTeXt module

656

656
656

6.2 Overview

6.2.1 Implementation

6.2.1.1 Bridging the semantic gap

JoyLoL *is explicitly defined to be* a fixed point of the formal semantic functor, making JoyLoL its own formal semantic definition. However there is, currently, no existing computational device which *implements* the JoyLoL language. That is, there is no computational device which 'runs' JoyLoL code natively.

The objective of this document is to provide an implementation of JoyLoL in as transparently correct way as possible. As discussed in, [Gai17], the formal definition of any computational language has two distinct components: one *deductive* and the other *inductive*. While we can rigorously check any deductive proofs of correctness, we can only ever hope to falsify any inductive tests of correctness. Any formally correct implementation of a computational language needs to be explicitly clear where the line between the deductively provable and the inductively testable is located.

The desired goal of any rigorous implementation is to keep as much as possible of the code deductively provable. Conversely any rigorous implementation needs to keep any code which is only inductively testable as clear and simple as possible. However how and were we draw the line between the deductively provable and the merely inductively testable implementation, will have profound impact upon the *performance* of all resulting JoyLoL computations run using this implementation. Provable correctness *and* performance are *both* critically important.

To obtain the correct balance of correctness, (potential) performance, and simplicity, JoyLoL has been designed as a 'trampolining' interpreter, written in ANSI-C, but meta-compiled from Literate sources written in ConTEXt/LuaTEX which are transcribed into ANSI-C source before being compiled to an executable on a given platform by an appropriately chosen ANSI-C compiler.

Finally since JoyLoL is meant to form a foundation for Mathematics, and, as such, the basis of mathematical proof, we need to ensure the JoyLoL language is accessible within the most common tool, TEX, used by mathematicians to communicate their proofs. To do this we wrap JoyLoL in a simple Lua interface. By wrapping the ANSI-C JoyLoL libraries in a Lua interface, we allow the JoyLoL libraries to be used, in particular, inside LuaTEX and hence inside LATEX and ConTEXt documents. At the moment, LATEX does not make integral use of LuaTEX's Lua subsystem. Instead we make use of ConTEXt for most of our documentation and mathematical writing, since ConTEXt does make integral use of LuaTEX's Lua subsystem.

6.2.1.2 Literate Sources

The literate sources, provide human readable documentation and justifications for each JoyLoL Co-Algebraic extension, complete with formal semantic definitions of each axiomatic word in JoyLoL.

6.2.1.3 ANSI-C system code

We build the lowest level system code for JoyLoL using ANSI-C with a few "standard" POSIX extension libraries. We have chosen ANSI-C for its:

- **portability**: There are a large number of ANSI-C compatible C compilers which target almost *all* computers currently in existence.
- **inter-working**: There are a large number of code libraries implementing useful algorithms which can be 'loaded' into the runtime image of an ANSI-C compiled program.
- **performance**: *If* desired, the overall JoyLoL interpreter can be compiled using any of the modern ANSI-C compilers' optimization modes. Since ANSI-C is so heavily used, the optimizing modes of most compilers are realatively well 'understood', tested and stable.
- **transparency**: The semantic gap between ANSI-C and the 'assembler' / 'machine-code' of almost any computer is small enough that a *large number* of skilled programmers could, if needed, hand code any C code directly into a given machine-code. For our needs, this means that there is no obscure mapping between the short pieces of JoyLoL implementation code and a given CPU's machine-code. This ensures that what JoyLoL does when running is 'relatively' easy to understand for most programmers.
- **familiarity**: While programmers are only a small part of our target audience, given we are explicitly dealing with the mathematics of computation, the programming community is an important part of the audience. More importantly 'most' programmers have a 'working' familiarity with the subset of ANSI-C used to implement the lowest levels of JoyLoL.

6.2.1.4 Interpreter structure

Since all JoyLoL words explicitly manage the context's data and process stacks, there is, in theory, no need for the ANSI-C call stack. The typical C-like language uses the call stack to hold both local data, any call parameters, as well as the process location to which to 'return'. Because data and process information are mixed on the call stack, to keep the call stack from growing without bounds, the explicit

expectation of any C-like language is that calls 'return' in the *strict* reverse order in which they are called, and that, more importantly, there is a finite limit on the number of calls a process might make.

When using JoyLoL as a foundation of Mathematics, we will find that there are many processes which do not naturally follow this strict return in reverse order pattern. Keeping the data and process stacks separate ensures that JoyLoL does not need to be enforce this strict call pattern. Instead JoyLoL implements a 'continuation passing style' of programming, see, for example, [SW00], [Gor79, section 5.1), [Ten81, chapter 10) [FW08, chapters 5 and 6).

In typical programming languages, this continuation passing style is implemented using either explicit 'jumps'/'gotos', see [Ten81, chapter 10), or, alternatively, using 'tail calls', see [Pro01, chapter 2), or [FW08, section 6.2). The use of explicit computed gotos, which are implemented as non-ANSI-C standard *extensions*, requires the use of global variables to pass the data and process stacks. Unfortunately this use of global variables inhibits most standard C compiler optimizations[7].

In the best of all worlds, we could implement JoyLoL's lowest levels using a *systems programming language* with native 'tail calls'. Since the data and process stacks can now be passed as 'normal' procedure parameters, standard C compiler optimizations will not be inhibited. Unfortunately no widely used systems programming language currently implements tail calls. While the functional languages such as Haskell, and Lisp/Scheme have native tail calls, they do not map sufficiently cleanly onto the underlying machine-language of a given computer's CPU. All C-like languages, who do typically map reasonably cleanly onto a given CPU's architecture, do not have a tail call friendly call structure.

To solve this problem, following [FW08, Section 5.2), we use a 'trampolining' interpreter as the main 'eval loop' for JoyLoL. The use of trampolining, ensures that the C-call stack never grows very large. JoyLoL words are implemented as simple C procedures keeping the structure of the resulting C-code simple. Trampolining also allows the use of external libraries which expect C-call stacks. For each cycle around the JoyLoL eval loop, the top of the process stack is used to determine which C procedure (JoyLoL word) to call next.

Unfortunately, while providing simply structured C-code, trampolining of small C procedures, is not as performant as a system which makes use of native tail calls. Instead by using the ConTEXt/LuaTEX based meta-compiler we can pre-compile any complex JoyLoL word definitions as explicit C procedures which *can allow* a given C compiler's optimization mode to produce performant code. This means

[7] With considerably more effort, we *could* arrange to keep the data and process stacks in 'local' variables in 'simulated' 'C-call stacks'. While this *might* improve performance, the use of such simulated C-call stacks, being so non-standard, would seriously reduce the number of programmers who could *easily* understand the resulting C-code.

that the JoyLoL system itself can be written in JoyLoL, allowing it to be proven deductively correct, yet still be performant.

6.2.1.5 Call structure

JoyLoL is a Forth-like language which manipulates 'stacks'. Almost all existing general purpose computational devices are 'register based'. Cleanly and performantly implementing stack based languages on a register based computational device has been previously explored in Ertl's thesis, [Ert96].

6.2.1.6 Bootstrapping JoyLoL

Since we want your tool set to be as rigorous as possible, we ultimately need to use JoyLoL to deductively prove its own correctness. Unfortunately, at least initially, most users do not have a running version of JoyLoL. In order to obtain the *first* running version, we need to 'bootstrap' the tool set by building an initial version of JoyLoL which is not rigorously checked.

Since we assume that any serious user of JoyLoL will be using JoyLoL to develop mathematical arguments, and hence will be using ConTEXt, we will provide this 'bootstrapped' JoyLoL using Lua. To do this each JoyLoL CoAlgebra will contain a highly simplified version of itself as pure Lua using the `MinJoyLoL` environment.

6.2.2 Organization

While we assert that all of the CoAlgebraic extensions provided in this document are conservative extensions over JoyLoL provided with only Lists of Lists, it is useful, for 'bears of *very* little brains' such as myself, to work, at least initially, with the extra structure provided by these CoAlgebraic extensions. We will show in a subsequent paper that all of the CoAlgebraic extensions provided in this document, are conservative extensions over JoyLoL provided with only lists of lists.

6.3 Code Manipulation

In this chapter we define the ConTEXt tools we will use to define the JoyLoL language.

The JoyLoL CoAlgebra ConTEXt module provides the tools required to fully describe the formal semantics of a particular JoyLoL CoAlgebraic extension including any defined JoyLoL words. It consists of literate documentation of the actual source code produced to implement the JoyLoL CoAlgebraic extension.

1 Implementation

In this section we load the Syntax Highlighter modules used by the code display commands (below). We also load the ConTests module used to test the JoyLoL CoAlgebra module itself. We then load the lua code associated with the `t-joylol` module.

MkIVCode : default

```
1 \writestatus{loading}{ConTeXt User Module / JoyLoL CoAlgebra Extensions}
2
3 \usemodule[t-literate-progs]
4 \usemodule[t-high-lisp]
5 \usemodule[t-contests]
6 \usemodule[t-joylol]
7
8 \ctxloadluafile{t-joylol-coalg}
```

2 Test Suite: JoyLoLCoAlg environment

The JoyLoLCoAlg environment provides a highly structured environment in which to describe the formal semantics and implementation of a particular JoyLoL CoAlgebraic extension.

A typical JoyLoLCoAlg environment consists of a collection of JoyLoL words. This includes a 'global' word which defines any global code required by the CoAlgegraic extension as a whole.

3 Examples

```
\startJoyLoLCoAlg[title=List of Lists][lists]
```

The first argument provides the arguments to an embedded `\startchapter` command.

The second argument provides the arguments to an embedded `\startcomponent` command. It also provides the base file name of all of the automatically generated code fragments.

The second argument also determines the name of any JoyLoL, ANSI-C, or Lua source file artefacts produced by this literate code documentation.

MkIVCode : default

```
\def\declareJoyLoLCoAlg[#1]{
  \directlua{thirddata.joylolCoAlgs.newCoAlg('#1')}
}

\let\startJoyLoLCoAlg=\declareJoyLoLCoAlg

\def\stopJoyLoLCoAlg{\relax}
```

LuaCode : repl

```
--local function newCoAlg(coAlgName)
--  local lCoAlg        = setDefs(theCoAlg, coAlgName)
--  lCoAlg.name         = coAlgName
--  lCoAlg.words        = lCoAlg.words or {}
--  lCoAlg.words.global = {}
--end

local function newCoAlg(coAlgName)
  texio.write_nl('newCoAlg: ['..coAlgName..']')
  theCoAlg               = {}
  theCoAlg.name          = coAlgName
  theCoAlg.ctx           = nil --joylol.newContext()
  theCoAlg.hasJoyLoLCode = false;
  theCoAlg.hasLuaCode    = false;
  theCoAlg.hasCHeader    = false;
  theCoAlg.hasCCode      = false;
  build.coAlgsToBuild = build.coAlgsToBuild or {}
  tInsert(build.coAlgsToBuild, coAlgName)
  build.coAlgDependencies = build.coAlgDependencies or {}
end

coAlgs.newCoAlg = newCoAlg
```

3.1 Implementation: Start: Tests

Test case

should do something

```
\mockContextMacro{startcomponent}{1}
\mockContextMacro{startchapter}{1}
\startJoyLoLCoAlg[title=List of Lists][lists]
\assertMacroNthArgumentOnMthExpansionMatches%
  {startcomponent}{1}{1}{lists}{}
\assertMacroNthArgumentOnMthExpansionMatches%
  {startchapter}{1}{1}{title=List of Lists}{}
\startLuaConTest
  local theCoAlg = thirddata.joylolCoAlgs.theCoAlg
  assert.isTable(theCoAlg)
  assert.hasKey(theCoAlg, 'lists')
  local lists = theCoAlg.lists
  assert.isTable(lists)
  assert.hasKey(lists, 'name')
  assert.matches(lists.name, 'lists')
  assert.hasKey(lists, 'words')
  local words = lists.words
  assert.hasKey(words, 'global')
\stopLuaConTest
```

[lists] ConTest FAILED:
Expected [startcomponent] to have been expanded
in file: /home/stg/ExpositionGit/tools/conTeXt/joylol-c/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 ConTest FAILED:
Expected [startchapter] to have been expanded
in file: /home/stg/ExpositionGit/tools/conTeXt/joylol-c/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 LuaTest FAILED:
Could not execute the LuaTest.
Expected table: 0x9622ef0 to have the key lists.
in file: /home/stg/ExpositionGit/tools/conTeXt/joylol-c/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118 ConTest FAILED:
LuaConTest failed
expected LuaConTest [local theCoAlg = thirddata.joylolCoAlgs.theCoAlgassert.isTable(theCoAlg)assert.hasKey(theCoAlg, 'lists')local lists = theCoAlg.listsassert.isTable(lists)assert.hasKey(lists, 'name')assert.matches(lists.name, 'lists')assert.hasKey(lists, 'words')local words = lists.wordsassert.hasKey(words, 'global')] to succeed

in file: /home/stg/ExpositionGit/tools/conTeXt/joylol-c/module/t-joylol-coalg/doc/context/third/joyLoLCoAlg/codeManipulation.tex between lines 96 and 118

4 Implementation: Stop

MkIVCode : default

```
\def\stopJoyLoLCoAlg{
  \directlua{thirddata.joylolCoAlgs.createCoAlg()}
  \stopchapter
  \stopcomponent
}
```

LuaCode : repl

```
local function createCoAlg()
end

coAlgs.createCoAlg = createCoAlg
```

6.3.1 Source licenses

6.3.1.1 Examples

6.3.1.2 Implementation

MkIVCode : default

```
\unexpanded\def\srcCopyrightCCBYSA{}
```

6.3.2 Target licenses

6.3.2.1 Examples

6.3.2.2 Implementation

MkIVCode : default

```
\unexpanded\def\targetCopyrightMIT{}
```

6.3.3 Describing CoAlgebraic dependencies

6.3.3.1 Examples

6.3.3.2 Implementation

MkIVCode : default

```
\def\dependsOn[#1]{
  \directlua{thirddata.joylolCoAlgs.addDependency('#1')}
}
```

LuaCode : repl

```
local function addDependency(dependencyName)
  build.coAlgDependencies = build.coAlgDependencies or {}
  tInsert(build.coAlgDependencies, dependencyName)
end

coAlgs.addDependency = addDependency
```

6.3.4 JoyLoL stack action: In

A JoyLoL stack action (either in or out) contains one or more sections of *implementation* code, either ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks. These stack actions provide the only allowed interface between an implementation language's 'local' variables and the JoyLoL stack context. 'In' actions take a data structure in a local variable and place it on either the data or process stacks.

6.3.4.1 Examples

6.3.4.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLStackActionIn[#1]{
  \directlua{thirddata.joylolCoAlgs.newStackActionIn('#1')}
}
```

LuaCode : repl

```
local function newStackActionIn(aWord)
```

```
end

coAlgs.newStackActionIn = newStackActionIn
```

6.3.4.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLStackActionIn{
  \directlua{thirddata.joyloCoAlgs.endStackActionIn()}
}
```

LuaCode : repl

```
local function endStackActionIn()
end

coAlgs.endStackActionIn = endStackActionIn
```

6.3.5 JoyLoL stack action: Out

A JoyLoL stack action (either in or out) contains one or more sections of *implementation* code, either ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks. These stack actions provide the only allowed interface between an implementation language's 'local' variables and the JoyLoL stack context. 'Out' actions take an item on either the data or process stack and place it into a data structure in a local variable and *possibly* 'removing' it from the appropriate stack.

6.3.5.1 Examples

6.3.5.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLStackActionOut[#1]{
  \directlua{thirddata.joylolCoAlgs.newStackActionOut('#1')}
}
```

LuaCode : repl

```
local function newStackActionOut(aWord)
```

```
end

coAlgs.newStackActionOut = newStackActionOut
```

6.3.5.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLStackActionOut{
  \directlua{thirddata.joylolCoAlgs.endStackActionOut()}
}
```

LuaCode : repl

```
local function endStackActionOut()
end

coAlgs.endStackActionOut = endStackActionOut
```

6.3.6 Describing the data stack

6.3.6.1 Examples

6.3.6.2 Implementation

MkIVCode : default

```
\def\preDataStack[#1][#2]{
  \directlua{thirddata.joylolCoAlgs.addPreDataStackDescription('#1', '#2')}
}

\def\postDataStack[#1]{
  \directlua{thirddata.joylolCoAlgs.addPostDataStackDescription('#1')}
}
```

LuaCode : repl

```
local function addPreDataStackDescription(arg1, arg2)
end

coAlgs.addPreDataStackDescription = addPreDataStackDescription

```

```
local function addPostDataStackDescription(arg1, arg2)
end

coAlgs.addPostDataStackDescription = addPostDataStackDescription
```

6.3.7 Describing the process stack

6.3.7.1 Examples

6.3.7.2 Implementation

MkIVCode : default

```
\def\preProcessStack[#1][#2]{
  \directlua{thirddata.joylolCoAlgs.addPreProcessStackDescription('#1',
'#2')}
}

\def\postProcessStack[#1]{
  \directlua{thirddata.joylolCoAlgs.addPostProcessStackDescription('#1')}
}
```

LuaCode : repl

```
local function addPreProcessStackDescription(arg1, arg2)
end

coAlgs.addPreProcessStackDescription = addPreProcessStackDescription

local function addPostProcessStackDescription(arg1, arg2)
end

coAlgs.addPostProcessStackDescription = addPostProcessStackDescription
```

6.4 JoyLoL

QUESTION: How do we load a *.joy file? Where do we put this command?

6.4.1 JoyLoL code environment

6.4.1.1 Examples

6.4.1.2 Implementation

We begin by registering the `JoylolCode` code type with the build srcTypes system. This will ensure the `\createJoylolCodeFile` macro (and corresponding lua code) knows how to deal with files of `JoylolCode`.

LuaCode : repl

```
build.srcTypes = build.srcTypes or { }
build.srcTypes['JoylolCode'] = 'joylolCode'
```

MkIVCode : default

```
\defineLitProgs
  [JoylolCode]
  [ option=lisp, numbering=line,
    before={\noindent\startLitProgFrame}, after=\stopLitProgFrame
  ]

\setLitProgsOriginMarker[JoylolCode][markJoylolCodeOrigin]
```

LuaCode : repl

```
local function markJoylolCodeOrigin()
  local codeType       = setDefs(code, 'JoylolCode')
  local codeStream     = setDefs(codeType, 'curCodeStream', 'default')
  codeStream           = setDefs(codeType, codeStream)
  return sFmt(';; from file: %s after line: %s',
    codeStream.fileName,
    toStr(
      mFloor(
        codeStream.startLine/code.lineModulus
      )*code.lineModulus
    )
  )
```

```
13 end
14
15 litProgs.markJoylolCodeOrigin = markJoylolCodeOrigin
```

6.4.2 Lua Make System files

In this section we add the code required to produce Lua Make System files which know how to compile JoyLoL CoAlgebraic extensions as shared libraries which can be loaded into a Lua implementation.

MkIVCode : default

```
1 \def\addJoyLoLTargets#1{%
2   \directlua{
3     thirddata.joylolCoAlgs.addJoyLoLTargets('#1')
4   }
5 }
```

LuaCode : repl

```
1  local function addJoyLoLTargets(aCodeStream)
2    litProgs.setCodeStream('Lmsfile', aCodeStream)
3    litProgs.markCodeOrigin('Lmsfile')
4    local lmsfile = {}
5    tInsert(lmsfile, "require 'lms.joyLoL'\n")
6    tInsert(lmsfile, "joylol.targets(lpTargets, {")
7    tInsert(lmsfile, "  coAlgs = {")
8    for i, aCoAlg in ipairs(build.coAlgsToBuild) do
9      tInsert(lmsfile, "    '"..aCoAlg.."',")
10   end
11   tInsert(lmsfile, "  },")

12   build.srcTargets = build.srcTargets or { }
13   local srcTargets = build.srcTargets

14   srcTargets.cHeader = srcTargets.cHeader or { }
15   local cHeader      = srcTargets.cHeader
16   tInsert(lmsfile, "  cHeaderFiles = {")
17   for i, aSrcFile in ipairs(cHeader) do
18     tInsert(lmsfile, "    '"..aSrcFile.."',")
19   end
20   tInsert(lmsfile, "  },")

21   srcTargets.cCode = srcTargets.cCode or { }
```

```
  local cCode      = srcTargets.cCode
  tInsert(lmsfile, "  cCodeFiles = {")
  for i, aSrcFile in ipairs(cCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  srcTargets.joylolCode = srcTargets.joylolCode or { }
  local joylolCode      = srcTargets.joylolCode
  tInsert(lmsfile, "  joylolCodeFiles = {")
  for i, aSrcFile in ipairs(joylolCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  if build.cCodeLibDirs then
    tInsert(lmsfile, "  cCodeLibDirs = {")
    for i, aLibDir in ipairs(build.cCodeLibDirs) do
      tInsert(lmsfile, "    '"..aLibDir.."',")
    end
    tInsert(lmsfile, "  },")
  end
  if build.cCodeLibs then
    tInsert(lmsfile, "  cCodeLibs = {")
    for i, aLib in ipairs(build.cCodeLibs) do
      tInsert(lmsfile, "    '"..aLib.."',")
    end
    tInsert(lmsfile, "  },")
  end

  tInsert(lmsfile, "  coAlgLibs = {")
  for i, aCoAlgDependency in ipairs(build.coAlgDependencies) do
    tInsert(lmsfile, "    '"..aCoAlgDependency.."',")
  end
  tInsert(lmsfile, "  },")
  tInsert(lmsfile, "})")
  litProgs.setPrepend('Lmsfile', aCodeStream, true)
  litProgs.addCode.default('Lmsfile', tConcat(lmsfile, '\n'))
end

coAlgs.addJoyLoLTargets = addJoyLoLTargets
```

MkIVCode : default

```
\def\addCTestJoyLoLCallbacks#1{%
  \directlua{
    thirddata.joylolCoAlgs.addCTestJoyLoLCallbacks('#1')
  }
}
```

LuaCode : repl

```
local function addCTestJoyLoLCallbacks(aCodeStream)
  local contests      = setDefs(thirddata, 'contests')
  local tests         = setDefs(contests, 'tests')
  local methods       = setDefs(tests, 'methods')
  local setup         = setDefs(methods, 'setup')
  local cTests        = setDefs(setup, 'cTests')
  aCodeStream         = aCodeStream         or 'default'
  cTests[aCodeStream] = cTests[aCodeStream] or { }
  tInsert(cTests[aCodeStream], [=[void ctestsWriteStdOut(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  fprintf(stdout, "%s", aMessage);
}

void ctestsWriteStdErr(
  JoyLoLInterp *jInterp,
  Symbol       *aMessage
) {
  fprintf(stderr, "%s", aMessage);
}
void *ctestsCallback(
  lua_State *lstate,
  size_t resourceId
) {
  if (resourceId == JoyLoLCallback_StdOutMethod) {
    return (void*)ctestsWriteStdOut;
  } else if (resourceId == JoyLoLCallback_StdErrMethod) {
    return (void*)ctestsWriteStdErr;
  } else if (resourceId == JoyLoLCallback_Verbose) {
    return (void*)FALSE;
  } else if (resourceId == JoyLoLCallback_Debug) {
    return (void*)FALSE;
  }
  return NULL;
}
```

```
]=])
  setup                = setDefs(tests, 'setup')
  cTests               = setDefs(setup, 'cTests')
  cTests[aCodeStream] = cTests[aCodeStream] or { }
  tInsert(cTests[aCodeStream], [=[setJoyLoLCallbackFrom(lstate, ctestsCallback);
]=])
end

coAlgs.addCTestJoyLoLCallbacks = addCTestJoyLoLCallbacks
```

MkIVCode : default

```
\def\setJoylolVerboseOn{%
  \directlua{thirddata.joylol.setVerbose(true)}
}

\def\setJoylolVerboseOff{%
  \directlua{thirddata.joylol.setVerbose(false)}
}

\def\setJoylolDebuggingOn{%
  \directlua{thirddata.joylol.setDebugging(true)}
}

\def\setJoylolDebuggingOff{%
  \directlua{thirddata.joylol.setVDebugging(false)}
}

\def\setJoylolTracingOn{%
  \directlua{thirddata.joylol.setTracing(true)}
}

\def\setJoylolTracingOff{%
  \directlua{thirddata.joylol.setTracing(false)}
}

\def\setJoylolShowStackOn{%
  \directlua{thirddata.joylol.setShowStack(true)}
}

\def\setJoylolShowStackOff{%
  \directlua{thirddata.joylol.setShowStack(false)}
}

```

```
\def\setJoylolShowSpecificationsOn{%
  \directlua{thirddata.joylol.setShowSpecifications(true)}
}

\def\setJoylolShowSpecificationsOff{%
  \directlua{thirddata.joylol.setShowSpecifications(false)}
}

\def\setJoylolCheckingOn{%
  \directlua{thirddata.joylol.setChecking(true)}
}

\def\setJoylolCheckingOff{%
  \directlua{thirddata.joylol.setChecking(false)}
}
```

LuaCode : repl

```
function showStack(aMessage)
  texio.write_nl('-----------------------------------------------')
  if aMessage and type(aMessage) == 'string' and 0 < #aMessage then
    texio.write_nl(aMessage)
  end
  dataStack    = joylol.showData()
  processStack = joylol.showProcess()
  texio.write_nl("Data:")
  texio.write_nl(dataStack)
  texio.write_nl("Process:")
  texio.write_nl(processStack)
  texio.write_nl('AT: '..status.filename..'::'..status.linenumber)
  texio.write_nl('-----------------------------------------------')

end

contests.showStack = showStack
```

6.5 JoyLoL Tests

QUESTION: How do we load a *.joy file? Where do we put this command?

1 JoylolTests

see ConTests LuaTests.tex file

To integrate into ConTests inside ConTeXt runner we need to create something like:

local function runCurLuaTestCase(suite, case) runALuaTest(case.lua, suite, case) end

contests.testRunners.runCurLuaTestCase = runCurLuaTestCase

Anything in the testRunners table must be a function taking two arguments as above.

MkIVCode : default

```
\definetyping[JoylolTest]
\setuptyping[JoylolTest][option=lisp]

\let\oldStopJoylolTest=\stopJoylolTest
\def\stopJoylolTest{%
  \oldStopJoylolTest%
  \directlua{thirddata.contests.addJoylolTest('_typing_')}
}

\def\showJoylolTest{%
  \directlua{thirddata.contests.showJoylolTest()}
}

\def\setJoylolTestStage#1#2{%
  \directlua{%
    thirddata.contests.setJoylolTestStage('#1', '#2')
  }
}

\def\JoylolTestsMethodSetup{%
  \setJoylolTestStage{Methods}{Setup}
}

\def\JoylolTestsMethodTeardown{%
  \setJoylolTestStage{Methods}{Teardown}
}
```

```
27
28  \def\JoylolTestsSetup{%
29    \setJoylolTestStage{Global}{Setup}
30  }
31
32  \def\JoylolTestsTeardown{%
33    \setJoylolTestStage{Global}{Teardown}
34  }
35
36  \def\JoylolTestSuiteSetup{%
37    \setJoylolTestStage{TestSuite}{Setup}
38  }
39
40  \def\JoylolTestSuiteTeardown{%
41    \setJoylolTestStage{TestSuite}{Teardown}
42  }
43
44  \def\setJoylolTestStream#1{%
45    \directlua{
46      thirddata.contests.setJoylolTestStream('#1')
47    }
48  }
49
50  \def\addJoylolTestInclude#1{%
51    \directlua{
52      thirddata.contests.addJoylolTestInclude('#1')
53    }
54  }
55
56  \def\addJoylolTestLibDir#1{%
57    \directlua{
58      thirddata.contests.addJoylolTestLibDir('#1')
59    }
60  }
61
62  \def\addJoylolTestLib#1{%
63    \directlua{
64      thirddata.contests.addJoylolTestLib('#1')
65    }
66  }
67
68  \def\createJoylolTestFile#1#2#3{%
69    \directlua{
```

```
    thirddata.contests.createJoylolTestFile('#1', '#2', '#3')
  }
}

\def\addJoylolTestTargets#1{%
  \directlua{
    thirddata.contests.addJoylolTestTargets('#1')
  }
}
```

LuaCode : repl

```
local function addJoylolTest(bufferName)
  local bufferContents = buffers.getcontent(bufferName):gsub("\13", "\n")
  local methods        = setDefs(tests, 'methods')
  local suite          = setDefs(tests, 'curSuite')
  local case           = setDefs(suite, 'curCase')
  local joylolTests    = setDefs(case,  'joylolTests')
  local curStage       = tests.stage:lower()
  if curStage:find('global') then
    if curStage:find('up') then
      local setup      = setDefs(tests,    'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(tests,    'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  elseif curStage:find('suite') then
    if curStage:find('up') then
      local setup      = setDefs(suite,    'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(suite,    'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  elseif curStage:find('method') then
    if curStage:find('up') then
      local setup      = setDefs(methods,  'setup')
      joylolTests      = setDefs(setup,    'joylolTests')
    elseif curStage:find('down') then
      local teardown   = setDefs(methods,  'teardown')
      joylolTests      = setDefs(teardown, 'joylolTests')
    end
  end
```

```
33    tests.stage            = ''
34    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
35    joylolTestStream       = setDefs(joylolTests, joylolTestStream)
36    tInsert(joylolTestStream, bufferContents)
37  end
38
39  contests.addJoylolTest = addJoylolTest
40
41  local function setJoylolTestStage(suiteCase, setupTeardown)
42    tests.stage = suiteCase..'-'..setupTeardown
43  end
44
45  contests.setJoylolTestStage = setJoylolTestStage
46
47  local function setJoylolTestStream(aCodeStream)
48    if type(aCodeStream) ~= 'string'
49      or #aCodeStream < 1 then
50      aCodeStream = 'default'
51    end
52    tests.curJoylolTestStream = aCodeStream
53  end
54
55  contests.setJoylolTestStream = setJoylolTestStream
56
57  local function addJoylolTestInclude(anInclude)
58    local joylolIncludes   = setDefs(tests, 'joylolIncludes')
59    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
60    joylolTestStream       = setDefs(joylolIncludes, joylolTestStream)
61    tInsert(joylolTestStream, anInclude)
62  end
63
64  contests.addJoylolTestInclude = addJoylolTestInclude
65
66  local function addJoylolTestLibDir(aLibDir)
67    local joylolLibDirs    = setDefs(tests, 'joylolLibDirs')
68    local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
69    joylolTestStream       = setDefs(joylolLibDirs, joylolTestStream)
70    tInsert(joylolTestStream, aLibDir)
71  end
72
73  contests.addJoylolTestLibDir = addJoylolTestLibDir
74
75  local function addJoylolTestLib(aLib)
```

```
  local joylolLibs       = setDefs(tests, 'joylolLibs')
  local joylolTestStream = setDefs(tests, 'curJoylolTestStream', 'default')
  joylolTestStream       = setDefs(joylolLibs, joylolTestStream)
  tInsert(joylolTestStream, aLib)
end

contests.addJoylolTestLib = addJoylolTestLib
```

LuaCode : repl

```
local function buildJoylolChunk(joylolChunk, curSuite, curCase)
  if type(joylolChunk) == 'table' then
    joylolChunk = tConcat(joylolChunk, '\n')
  end

  if type(joylolChunk) ~= 'string' then
    return nil
  end

  if joylolChunk:match('^%s*$') then
    return nil
  end

  return [=[(
]=]..joylolChunk..[=[
)
(
 "]=]..curCase.desc..[=["
  ]=]..curCase.fileName..[=[
  ]=]..curCase.startLine..[=[
  ]=]..status.linenumber..[=[
)
runTestCase
showStack
true
]=]
end

contests.buildJoylolChunk = buildJoylolChunk

local function showJoylolTest()
  local curSuite = setDefs(tests, 'curSuite')
  local curCase  = setDefs(curSuite, 'curCase')
  texio.write_nl('================================================')
```

```
  local joylolChunk =
    buildJoylolChunk(curCase.joylol, curSuite, curCase)
  if joylolChunk then
    texio.write_nl('Joylol Test: ')
    texio.write_nl('-----------------------------------------------')
    texio.write_nl(joylolChunk)
    texio.write_nl('-----------------------------------------------')
  else
    texio.write_nl('NO Joylol Test could be built')
  end
  texio.write_nl('AT: '..status.filename..'::'..status.linenumber)
  texio.write_nl('===============================================')
end

contests.showJoylolTest = showJoylolTest
```

LuaCode : repl

```
local function runAJoylolTest(joylolTest, suite, case)
  case.passed = case.passed or true
  local joylolChunk = buildJoylolChunk(joylolTest, suite, case)
  if not joylolChunk then
    -- nothing to test
    return true
  end

  local caseStats = tests.stats.joylol.cases
  caseStats.attempted = caseStats.attempted + 1
  tex.print("\\starttyping")
  joylol.evalString(joylolChunk)
  tex.print("\\stoptyping")
  local testResult = joylol.popData()
  if not testResult then
    local errObj = joylol.popData()
    local failure = logFailure(
      "LuaTest FAILED",
      suite.desc,
      case.desc,
      errObj.message,
      toStr(errObj[1]),
      sFmt("in file: %s between lines %s and %s",
        case.fileName, toStr(case.startLine), toStr(case.lastLine))
      )
    reportFailure(failure, false)
```

```
    tInsert(tests.failures, failure)
    return false
  end

  -- all tests passed
  caseStats.passed = caseStats.passed + 1
  tex.print("\\noindent{\\green PASSED}")
  return true
end

contests.runAJoylolTest = runAJoylolTest

local function runCurJoylolTestCase(suite, case)
  runAJoylolTest(case.joylol, suite, case)
end

contests.testRunners.runCurJoylolTestCase = runCurJoylolTestCase
```

MkIVCode : default

```
\def\createJoylolTestFile#1#2#3{%
  \directlua{
    thirddata.contests.createJoylolTestFile('#1', '#2', '#3')
  }
}
```

LuaCode : repl

```
local function createJoylolTestFile(
  aCodeStream, aFilePath, aFileHeader
)
  texio.write("\n-------------------------------\n")
  texio.write("aCodeStream = ".. aCodeStream.."\n")
  texio.write("aFilePath   = ".. aFilePath.."\n")
  texio.write("\n-------------------------------\n")

  if not build.buildDir then
    texio.write('\nERROR: document directory NOT yet defined\n')
    texio.write('       NOT creating code file ['..aFilePath..']\n\n')
    return
  end

  if type(aFilePath) ~= 'string'
    or #aFilePath < 1 then
```

```
    texio.write('\nERROR: no file name provided for joylolTests\n\n')
    return
  end

  build.joylolTestTargets = build.joylolTestTargets or { }
  local aTestExec = aFilePath:gsub('%..+$','')
  tInsert(build.joylolTestTargets, aTestExec)

  aFilePath = build.buildDir .. '/buildDir/' .. aFilePath
  local outFile = io.open(aFilePath, 'w')
  if not outFile then
    return
  end

  texio.write('creating JoylolTest file: ['..aFilePath..']\n')

  if type(aFileHeader) == 'string'
    and 0 < #aFileHeader then
    outFile:write(aFileHeader)
    outFile:write('\n\n')
  end

  tests.suites = tests.suites or { }

  if type(aCodeStream) ~= 'string'
    or #aCodeStream < 1 then
    aCodeStream = 'default'
  end

  outFile:write(';; A JoylolTest file\n\n')

  outFile:write(';;-------------------------------------------------------\n')
  outFile:write(';; global setup\n')
  outFile:write(';;-------------------------------------------------------\n\n')
  local joylolIncludes = setDefs(tests, 'joylolIncludes')

  joylolIncludes[aCodeStream] = joylolIncludes[aCodeStream] or { }

  for i, anInclude in ipairs(joylolIncludes[aCodeStream]) do
    outFile:write(anInclude..'\n')
    outFile:write('load \n\n')
  end
  outFile:write('\n')
```

```
57
58    tests.methods = tests.methods or { }
59    local methods = tests.methods
60    methods.setup = methods.setup or { }
61    local mSetup  = methods.setup
62    mSetup.joylolTests = mSetup.joylolTests or { }
63    msJoylolTests      = mSetup.joylolTests
64
65    --msJoylolTests[aCodeStream] = msJoylolTests[aCodeStream] or { }

66    if msJoylolTests and
67      msJoylolTests[aCodeStream] then
68      local setupCode = tConcat(msJoylolTests[aCodeStream],'\n')
69      setupCode       = litProgs.splitString(setupCode)
70      outFile:write(tConcat(setupCode, '\n'))
71      outFile:write('\n')
72    end
73    outFile:write('\n')
74
75    outFile:write(';;-------------------------------------------------------\n')
76    outFile:write(';; all tests\n')
77    outFile:write(';;-------------------------------------------------------\n')
78
79    outFile:write('(\n')
80    outFile:write('  (\n')
81    tests.setup = tests.setup or { }
82    if tests.setup.joylolTests and
83      tests.setup.joylolTests[aCodeStream] then
84      local setupCode = tConcat(tests.setup.joylolTests[aCodeStream],'\n')
85      setupCode       = litProgs.splitString(setupCode)
86      outFile:write('  '..tConcat(setupCode, '\n  '))
87      outFile:write('\n')
88    end
89    outFile:write('  ) ;; JoylolTests setup\n')
90    outFile:write('  tests.defineTestsSetup\n\n')
91
92    outFile:write('  (\n')
93    tests.teardown = tests.teardown or { }
94    if tests.teardown.joylolTests and
95      tests.teardown.joylolTests[aCodeStream] then
96      local teardownCode =tConcat(tests.teardown.joylolTests[aCodeStream],'\n
97  ')
98      teardownCode = litProgs.splitString(teardownCode, '\n')
```

```
99      outFile:write('  '..tConcat(teardownCode, '\n  '))
100   end
101   outFile:write('  ) ;; JoylolTests teardown\n')
102   outFile:write('  tests.defineTestsTeardown\n\n')
103
104   for i, aTestSuite in ipairs(tests.suites) do
105     aTestSuite.cases = aTestSuite.cases or { }
106     local suiteCaseBuf = { }
107
108     for j, aTestCase in ipairs(aTestSuite.cases) do
109       local joylolTests     = setDefs(aTestCase, 'joylolTests')
110       if aTestCase.desc and
111         aTestCase.fileName and
112         aTestCase.startLine and
113         aTestCase.lastLine and
114         joylolTests[aCodeStream] then
115         tInsert(suiteCaseBuf, '    ;;-------------------------------------------------------
116         tInsert(suiteCaseBuf, '    ;; jTC: '..aTestCase.desc..'\n')
117         tInsert(suiteCaseBuf, '    ;;-------------------------------------------------------
118         tInsert(suiteCaseBuf, '    (\n')
119         tInsert(suiteCaseBuf, '      (\n')
120         tInsert(suiteCaseBuf, '        "'..aTestCase.desc..'"\n')
121         tInsert(suiteCaseBuf, '        "'..aTestCase.fileName..'"\n')
122         tInsert(suiteCaseBuf, '        '..toStr(aTestCase.startLine)..'\n')
123         tInsert(suiteCaseBuf, '        '..toStr(aTestCase.lastLine)..'\n')
124         tInsert(suiteCaseBuf, '      ) ;; test case details\n')
125         tInsert(suiteCaseBuf, '      tests.recordTestCaseDetails\n\n')
126         local joylolTestsCode = tConcat(joylolTests[aCodeStream], '\n')
127         joylolTestsCode       = litProgs.splitString(joylolTestsCode)
128         tInsert(suiteCaseBuf, '    '..tConcat(joylolTestsCode, '\n    '))
129         tInsert(suiteCaseBuf, '\n    ) ;; test case\n')
130         tInsert(suiteCaseBuf, '    tests.runTestCase\n\n')
131       elseif (not aTestCase.desc or
132         not aTestCase.fileName or
133         not aTestCase.startLine or
134         not aTestCase.lastLine) and
135         joylolTests[aCodeStream] then
136         texio.write("\nERROR missing \\startTestCase\n")
137         texio.write("near:\n")
138         texio.write(tConcat(joylolTests[aCodeStream], '\n'))
139         texio.write('\n')
140       end
141     end
```

```
142
143     if aTestSuite.desc and (0 < #suiteCaseBuf) then
144       outFile:write('  ;;-----------------------------------------------------\n')
145       outFile:write('  ;; jTS:'..aTestSuite.desc..'\n')
146       outFile:write('  ;;-----------------------------------------------------\n')
147       outFile:write('  (\n')
148       outFile:write('    (\n')
149       outFile:write('      "'..aTestSuite.desc..'"\n')
150       outFile:write('    ) ;; test suite details\n')
151       outFile:write('    tests.recordTestSuiteDetails\n\n')
152
153       outFile:write('    (\n')
154       aTestSuite.setup = aTestSuite.setup or { }
155       if aTestSuite.setup.joylolTests and
156         aTestSuite.setup.joylolTests[aCodeStream] then
157         local setupCode = tConcat(aTestSuite.setup.joylolTests[aCodeStream],'\n
158 ')
159         setupCode = litProgs.splitString(setupCode, '\n')
160         outFile:write('    '..tConcat(setupCode, '\n    '))
161       end
162       outFile:write('    ) ;; test suite setup\n')
163       outFile:write('    tests.defineTestSuiteSetup\n\n')
164
165       outFile:write('    (\n')
166       aTestSuite.teardown = aTestSuite.teardown or { }
167       if aTestSuite.teardown.joylolTests and
168         aTestSuite.teardown.joylolTests[aCodeStream] then
169         local teardownCode = tConcat(aTestSuite.teardown.joylolTests[aCodeStream],'\n
170 ')
171         teardownCode = litProgs.splitString(teardownCode, '\n')
172         outFile:write('    '..tConcat(teardownCode, '\n    '))
173       end
174       outFile:write('    ) ;; test suite teardown\n')
175       outFile:write('    tests.defineTestSuiteTeardown\n\n')
176
177       outFile:write(tConcat(suiteCaseBuf))
178
179       outFile:write('  )\n')
180       outFile:write('  tests.runTestSuite\n\n')

181     elseif not aTestSuite.desc and (0 < #suiteCaseBuf) then
182       texio.write("\nERROR missing \\startTestSuite\n")
183       texio.write("near:\n")
```

```
184       texio.write(tConcat(suiteCaseBuf, '\n'))
185       texio.write('\n')
186     end
187   end

188   outFile:write(')\n')
189   outFile:write('tests.runAllTests\n\n')
190
191   outFile:write(';;-------------------------------------------------------\n')
192   outFile:write(';; global teardown\n')
193   outFile:write(';;-------------------------------------------------------\n\n')

194   methods.teardown      = methods.teardown or { }
195   local mTeardown       = methods.teardown
196   mTeardown.joylolTests = mTeardown.joylolTests or { }
197   mtJoylolTests         = mTeardown.joylolTests
198
199   --mtJoylolTests[aCodeStream] = mtJoylolTests[aCodeStream] or { }
200
201   if mtJoylolTests and
202     mtJoylolTests[aCodeStream] then
203     local teardownCode = tConcat(mtJoylolTests[aCodeStream],'\n')
204     teardownCode       = litProgs.splitString(teardownCode)
205     outFile:write('  '..tConcat(teardownCode, '\n  '))
206     outFile:write('\n')
207   end
208   outFile:write('\n')
209   outFile:write(';;-------------------------------------------------------\n')
210
211   outFile:close()
212 end
213
214 contests.createJoylolTestFile = createJoylolTestFile
```

6.5.1 Lua Make System files

In this section we add the code required to produce Lua Make System files which know how to compile JoyLoL Tests.

MkIVCode : default

```
1 \def\addJoyLoLTestTargets#1{%
2   \directlua{
```

```
    thirddata.joylolCoAlgs.addJoylolTestTargets('#1')
  }
}
```

LuaCode : repl

```
local function addJoylolTestTargets(aCodeStream)
  litProgs.setCodeStream('Lmsfile', aCodeStream)
  litProgs.markCodeOrigin('Lmsfile')
  local lmsfile = {}
  tInsert(lmsfile, "require 'lms.joylolTests'\n")
  tInsert(lmsfile, "joylolTests.targets(lpTargets, {")
  tInsert(lmsfile, "  testExecs = {")
  for i, aTestExec in ipairs(build.joylolTestTargets) do
    tInsert(lmsfile, "    '"..aTestExec.."',")
  end
  tInsert(lmsfile, "  },")

  build.srcTargets = build.srcTargets or { }
  local srcTargets = build.srcTargets

  srcTargets.cHeader = srcTargets.cHeader or { }
  local cHeader      = srcTargets.cHeader
  tInsert(lmsfile, "  cHeaderFiles = {")
  for i, aSrcFile in ipairs(cHeader) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  srcTargets.cCode = srcTargets.cCode or { }
  local cCode      = srcTargets.cCode
  tInsert(lmsfile, "  cCodeFiles = {")
  for i, aSrcFile in ipairs(cCode) do
    tInsert(lmsfile, "    '"..aSrcFile.."',")
  end
  tInsert(lmsfile, "  },")

  if build.cCodeLibDirs then
    tInsert(lmsfile, "  cCodeLibDirs = {")
    for i, aLibDir in ipairs(build.cCodeLibDirs) do
      tInsert(lmsfile, "    '"..aLibDir.."',")
    end
    tInsert(lmsfile, "  },")
  end
```

```
36    if build.cCodeLibs then
37      tInsert(lmsfile, "  cCodeLibs = {")
38      for i, aLib in ipairs(build.cCodeLibs) do
39        tInsert(lmsfile, "    '"..aLib.."',")
40      end
41      tInsert(lmsfile, "  },")
42    end
43
44    tInsert(lmsfile, "  coAlgLibs = {")
45    for i, aCoAlgDependency in ipairs(build.coAlgDependencies) do
46      tInsert(lmsfile, "    '"..aCoAlgDependency.."',")
47    end
48    tInsert(lmsfile, "  },")
49    tInsert(lmsfile, "})")
50    litProgs.setPrepend('Lmsfile', aCodeStream, true)
51    litProgs.addCode.default('Lmsfile', tConcat(lmsfile, '\n'))
52  end
53
54  coAlgs.addJoylolTestTargets = addJoylolTestTargets
```

6.6 Rules

1 Test Suite: rule environment

MkIVCode : default

```
\let\stopRule\relax

\def\stopRuleDone{
  \directlua{thirddata.joylolCoAlgs.stopRule()}
}

\def\startRule[#1]{
  \directlua{thirddata.joylolCoAlgs.startRule('#1')}
  \buff_pickup{_rules_buffer_}%
    {startRule}{stopRule}%
    {\relax}{\stopRuleDone}\plusone%
}
```

LuaCode : repl

```
local function startRule(ruleName)
  texio.write_nl("starting rule: ["..ruleName.."]")
end

coAlgs.startRule = startRule

local sectionHeaders = tConcat({
  'arguments',
  'returns',
  'preDataStack',
  'preProcessStack',
  'preConditions',
  'postDataStack',
  'postProcessStack',
  'postConditions'
}, '|'):lower()

local function stopRule()
  local rulesBody  = buffers.getcontent('_rules_buffer_'):gsub("\13",
"\n")
  local rules      = { }
  local lines      = { }
  local curSection = 'ignore'
```

```
  for aLine in rulesBody:gmatch("[^\r\n]+") do
    local aMatch = aLine:match("^%s*\\(%a+)%s*$")
    if aMatch and
      sectionHeaders:find(aMatch:lower(), 1, true)
    then
      rules[curSection] = lines
      lines             = { }
      curSection        = aMatch
    else
      tInsert(lines, aLine)
    end
  end
  rules[curSection] = lines

  texio.write_nl('---------rules-buffer-------------')
  texio.write_nl(lpPP(rules))
  texio.write_nl('---------rules-buffer-------------')
end

coAlgs.stopRule = stopRule
```

Test case

should manipulate buffers

```
  \startRule[testRule]
    \arguments
      some argument content
    \returns
      some returns content
    \preDataStack
      some preDataStack content
    \preProcessStack
      some preProcessStack content
    \preConditions
      some preConditions content
    \postDataStack
      some postDataStack content
    \postProcessStack
      some postProcessStack content
    \postConditions
      some postConditions content
  \stopRule
```

692

692
692

6.7 JoyLoL code fragments

6.7.1 JoyLoL implementation fragment

6.7.1.1 Examples

6.7.1.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLFragment[#1]{
  \directlua{thirddata.joylolCoAlgs.newFragment('#1')}
}
```

LuaCode : repl

```
local function newFragment(fragmentName)
  local curFragment = setDefs(theCoAlg, 'curFragment')
  curFragment.name  = fragmentName
  setDefs(curFragment, 'code')
end

coAlgs.newFragment = newFragment
```

6.7.1.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLFragment{
  \directlua{thirddata.joylolCoAlgs.endFragment()}
}
```

LuaCode : repl

```
local function endFragment()
  local curFragment =
    shouldExist(theCoAlg, 'curFragment', {
      '\\stopJoyLoLFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })

  texio.write_nl('---------joylol-fragment-------------')
```

```
  texio.write_nl(lpPP(curFragment))
  texio.write_nl('---------joylol-fragment-------------')

  local wordName =
    shouldExist(curFragment, 'name',
      'joylol fragment not named'
    )
  local codeVersions =
    shouldExist(curFragment, 'code',
      'incorrectly setup joylol fragment'
    )

  local numCodeVersions = 0
  for fragmentType, fragmentBody in pairs(codeVersions) do
--    joylol.crossCompilers.addFragment(
--      fragmentType,
--      wordName,
--      fragmentBody
--    )
    numCodeVersions = numCodeVersions + 1
  end
  if numCodeVersions < 1 then
    error(tConcat({
      'no \\startFragment environment used ',
      'inside a \\startJoyLoLFragment environment'
    }))
  end
end

coAlgs.endFragment = endFragment
```

6.7.2 fragment definition environment

MkIVCode : default

```
\let\stopFragment\relax

\def\stopFragmentDone{
  \directlua{thirddata.joylolCoAlgs.stopFragment()}
}

\def\startFragment[#1]{
  \directlua{thirddata.joylolCoAlgs.startFragment('#1')}
```

```
  \buff_pickup{_fragment_buffer_}%
    {startFragment}{stopFragment}%
    {\relax}{\stopFragmentDone}\plusone%
}
```

LuaCode : repl

```
local function startFragment(fragmentType)
  local curFragment =
    shouldExist(theCoAlg, 'curFragment', {
      '\\startFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })
  curFragment.curType  = fragmentType
end

coAlgs.startFragment = startFragment

local function stopFragment()
  local curFragment  =
    shouldExist(theCoAlg, 'curFragment', {
      '\\stopFragment used outside of ',
      '\\startJoyLoLFragment environment'
    })
  local codeVersions =
    shouldExist(curFragment, 'code',
      'incorrectly setup joylol fragment - missing code'
    )
  local curType =
    shouldExist(curFragment, 'curType',
      'incorrectly setup fragment - missing curType'
    )
  local fragmentBody =
    buffers.getcontent('_fragment_buffer_'):gsub("\13", "\n")
  codeVersions[curType] = fragmentBody

  tex.sprint("\\starttyping")
  tex.print(fragmentBody)
  tex.sprint("\\stoptyping")
end

coAlgs.stopFragment = stopFragment
```

6.8 JoyLoL words

6.8.1 JoyLoL word environment

A JoyLoL word contains one or more sections of code, either JoyLoL, ANSI-C or Lua, together with a collection of descriptors of the JoyLoL {pre, post} {data, process} stacks.

6.8.1.1 Examples

6.8.1.2 Implementation: start

MkIVCode : default

```
\def\startJoyLoLWord[#1]{
  \directlua{thirddata.joylolCoAlgs.newWord('#1')}
}
```

LuaCode : repl

```
local function newWord(wordName)
  local curWord = setDefs(theCoAlg, 'curWord')
  curWord.name = wordName
  setDefs(curWord, 'code')
end

coAlgs.newWord = newWord
```

6.8.1.3 Implementation: stop

MkIVCode : default

```
\def\stopJoyLoLWord{
  \directlua{thirddata.joylolCoAlgs.endWord()}
}
```

LuaCode : repl

```
local function endWord()
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\stopJoyLoLWord used outside of ',
      '\\startJoyLoLWord environment'
```

```
    })

  texio.write_nl('---------joylol-word-------------')
  texio.write_nl(lpPP(curWord))
  texio.write_nl('---------joylol-word-------------')

  local wordName =
    shouldExist(curWord, 'name',
      'joylol word not named'
    )
  local codeVersions =
    shouldExist(curWord, 'code',
      'incorrectly setup joylol word'
    )

  local numCodeVersions = 0
  for implType, implBody in pairs(codeVersions) do
--    joylol.crossCompilers.addImplementation(
--      implType,
--      wordName,
--      implBody
--    )
    numCodeVersions = numCodeVersions + 1
  end
  if numCodeVersions < 1 then
    error(tConcat({
      'no \\startImplementation environment used ',
      'inside a \\startJoyLoLWord environment'
    }))
  end
end

coAlgs.endWord = endWord
```

6.8.2 JoyLoL word implementation

MkIVCode : default

```
\let\stopImplementation\relax

\def\stopImplementationDone{
  \directlua{thirddata.joylolCoAlgs.stopImplementation()}
}
```

```
\def\startImplementation[#1]{
  \directlua{thirddata.joylolCoAlgs.startImplementation('#1')}
  \buff_pickup{_implementation_buffer_}%
    {startImplementation}{stopImplementation}%
    {\relax}{\stopImplementationDone}\plusone%
}
```

LuaCode : repl

```
local function startImplementation(implType)
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\startImplementation used outside of ',
      '\\startJoyLoLWord environment'
    })
  curWord.curType = implType
end

coAlgs.startImplementation = startImplementation

local function stopImplementation()
  local curWord =
    shouldExist(theCoAlg, 'curWord', {
      '\\stopImplementation used outside of ',
      '\\startJoyLoLWord environment'
    })
  local codeVersions =
    shouldExist(curWord, 'code',
      'incorrectly setup joylol word - missing code'
    )
  local curType =
    shouldExist(curWord, 'curType',
      'incorrectly setup joylol word - missing curType'
    )
  local implBody =
    buffers.getcontent('_implementation_buffer_'):gsub("\13", "\n")
  codeVersions[curType] = implBody

  tex.sprint("\\starttyping")
  tex.print(implBody)
  tex.sprint("\\stoptyping")
end

```

JoyLoL word implementation 6.8.2

```
35 coAlgs.stopImplementation = stopImplementation
```

6.9 Preamble

MkIVCode : default

```
%D \module
%D   [     file=t-joylol-coalg,
%D      version=2017.05.10,
%D        title=\CONTEXT\ User module,
%D     subtitle=The JoyLoL CoAlgebraic Extensions \ConTeXt\ module,
%D       author=Stephen Gaito,
%D         date=\currentdate,
%D    copyright=PerceptiSys Ltd (Stephen Gaito),
%D        email=stephen@perceptisys.co.uk,
%D      license=MIT License]

%C Copyright (C) 2017 PerceptiSys Ltd (Stephen Gaito)
%C
%C Permission is hereby granted, free of charge, to any person obtaining a
%C copy of this software and associated documentation files (the
%C "Software"), to deal in the Software without restriction, including
%C without limitation the rights to use, copy, modify, merge, publish,
%C distribute, sublicense, and/or sell copies of the Software, and to
%C permit persons to whom the Software is furnished to do so, subject to
%C the following conditions:
%C
%C The above copyright notice and this permission notice shall be included
%C in all copies or substantial portions of the Software.
%C
%C THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
%C OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
%C MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
%C IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
%C CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
%C TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
%C SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

% begin info
%
% title   : JoyLoL CoAlgebra definitions
% comment : Provides structured document and code generation
% status  : under development, mkiv only
%
% end info
```

```

\unprotect
```

MkIVCode : default

```
\protect \endinput
```

LuaCode : default

```
-- This is the lua code associated with t-joylol-coalg.mkiv

if not modules then modules = { } end modules ['t-joylol-coalg'] = {
    version   = 1.000,
    comment   = "joylol coalgegraic extensions - lua",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
    copyright = "PerceptiSys Ltd (Stephen Gaito)",
    license   = "MIT License"
}

thirddata         = thirddata        or {}
thirddata.joylol  = thirddata.joylol or {}

local joylol      = thirddata.joylol

thirddata.joylolCoAlgs = thirddata.joylolCoAlgs or {}
local coAlgs      = thirddata.joylolCoAlgs
coAlgs.theCoAlg   = {}
local theCoAlg    = coAlgs.theCoAlg

thirddata.literateProgs = thirddata.literateProgs or {}
local litProgs    = thirddata.literateProgs
litProgs.code     = litProgs.code or {}
local code        = litProgs.code
local setDefs     = litProgs.setDefs
local shouldExist = litProgs.shouldExist
local build       = setDefs(litProgs, 'build')

local contests    = setDefs(thirddata, 'contests')
local initStats   = contests.initStats
local tests       = setDefs(contests, 'tests')
                    setDefs(tests, 'suites')
                    setDefs(tests, 'failures')
local assert      = setDefs(contests, 'assert')
                    setDefs(contests, 'testRunners')
```

```
local expInfo     = setDefs(contests, 'expInfo')

                        setDefs(tests, 'stats')
tests.stats.joylol     = initStats()
local joylolStats      = tests.stats.joylol
local joylolAssertions = joylolStats.assertions

local tInsert = table.insert
local tConcat = table.concat
local tRemove = table.remove
local tSort   = table.sort
local sFmt    = string.format
local sMatch  = string.match
local toStr   = tostring
local mFloor  = math.floor
local lpPP    = litProgs.prettyPrint

--local pushData, pushProcess = joylol.pushData, joylol.pushProcess
--local pushProcessQuoted = joylol.pushProcessQuoted
--local popData, popProcess   = joylol.popData, joylol.popProcess
--local newList, newDictionary = joylol.newList, joylol.newDictionary
--local jEval = joylol.eval

if joylol.core then
  interfaces.writestatus(
    "joyLoL",
    joylol.core.context.gitVersion('commitDate')
  )
else
  interfaces.writestatus(
    "joyLoL",
    "partially loaded"
  )
end

interfaces.writestatus('joyLoLCoAlg', "loaded JoyLoL CoAlgs")
```

LuaTemplate : default

```
if not modules then modules = { } end modules ['t-joylol-coalg-templates']
= {
    version   = 1.000,
    comment   = "JoyLoL CoAlgebraic extensions module - templates",
    author    = "PerceptiSys Ltd (Stephen Gaito)",
```

```
    copyright = "PerceptiSys Ltd (Stephen Gaito)",
    license   = "MIT License"
}

thirddata              = thirddata              or {}
thirddata.joylolCoAlgs = thirddata.joylolCoAlgs or {}

local coAlgs     = thirddata.joylolCoAlgs

local templates  = { }

templates.cHeader = [=[This is the start of a cHeader template
{{ lookupInDict 'coAlgName }}
This is the end of a cHeader template
]=]

templates.cCode = [=[This is the start of a cCode template
{{ lookupInDict 'coAlgName }}
This is the end of the cCode template
]=]

templates.joyLoLCode = [=[This is the start of a joyLoLCode template
{{ lookupInDict 'coAlgName }}
This is the end of the joyLoLCode template
]=]

templates.luaCode = [=[-- A Lua file (automatically generated)
{{ lookupInDict 'coAlgName }}
This is the end of the luaCode template
]=]

local joyLoL = coAlgs.joyLoL
local pushData, pushProcess = joyLoL.pushData, joyLoL.pushProcess
local pushProcessQuoted = joyLoL.pushProcessQuoted
local popData, popProcess   = joyLoL.popData, joyLoL.popProcess
local newList, newDictionary = joyLoL.newList, joyLoL.newDictionary
local jEval = joyLoL.eval

-------------------------------------------------------------------------
-- NOTE the following uses raw JoyLoL code to load the templates into the
-- context provided.

-- To understand this code.... **think categorically**
```

```
49
50  -- In JoyLoL a particular object in the category *is* the structure of the
51  -- data stack, while a particular arrow in the category *is* the process
52  -- stack.
53
54  -- To understand what these arrows are doing... you read the JoyLoL code
55  -- in reverse order (from a 'jEval' up).
56  ------------------------------------------------------------------------
57
58  function coAlgs.loadTemplates(aCtx)
59    pushProcess(aCtx, 'addToDict')
60    for aKey, aValue in pairs(templates) do
61      pushProcess(aCtx, 'addToDict')
62      pushProcessQuoted(aCtx, aValue)
63      pushProcessQuoted(aCtx, aKey)
64    end
65    newDictionary(aCtx)
66    pushProcessQuoted(aCtx, 'templates')
67    jEval(aCtx)
68  end
69
70  interfaces.writestatus('joyLoLCoAlg', 'loaded JoyLoL CoAlg templates')
```

706

706
706

6.10 Conclusion

Lmsfile : default

Bibliography

AV80 report: [author: Apt, Krzysztof R and Van Emden, M H] [institution: Department of Computer Science, University of Waterloo] [number: CS-80-12] [pagetotal: 54] [title: Contributions to the theory of Logic Programming] [type: Research report] [url: https://cs.uwaterloo.ca/research/tr/1980/CS-80-12.pdf] [year: 1980]

ACV13 Awodey, S., Coquand, T., & Voevodsky, V. (Eds.) (2013). *Homotopy Type Theory: Univalent Foundation of Mathematics*. The Univalent Foundations Program, Institute for Advanced Study. Retrieved from http://homotopytypetheory.org/book/ (first-edition-974-g068cbba)

Bil90 Billaud, M. (1990). Simple operational and denotational semantics for Prolog with cut. doi:10.1016/0304-3975(90)90197-P

Ert96 Ertl, M. A. (1996). *Implementation of Stack-Based Languages on Register Machines*. (Dissertation). Retrieved from http://www.complang.tuwien.ac.at/papers/ertl96diss.ps.gz

FW08 Friedman, D. P. & Wand, M. (2008). *Essentials of Programming Languages* (, Ed.). (3rd ed.). MIT Press. Retrieved from http://www.eopl3.com/

Gai17 report: [author: Gaito, Stephen T] [institution: PerceptiSys Ltd] [title: JoyLoL] [type: Technical Report] [year: 2017]

Gia02 Giaquinto, M. (2002). *The search for certainty: A Philosophical Account of Foundations of Mathematics*. Oxford University Press.

Gor79 Gordon, M. J. C. (1979). *The Denotational description of Programming Languages: An introduction* (, Ed.). Springer-Verlag.

Gun92 Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. MIT Press. Retrieved from https://mitpress.mit.edu/books/semantics-programming-languages

Hat82 Hatcher, W. S. (1982). *The Logical Foundations of Mathematics*. Pergamon Press.

Lak76 Lakatos, I. (1976). *Proofs and Refutations* (J. Worrall & E. Zahar, Eds.). Cambridge University Press.

Mac01 MacKenzie, D. (2001). *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press.

MAE+65 McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., & Levin, M. I. (1965). *LISP 1.5 Programmer's Manual*. (2nd Ed ed.). MIT Press.

Pro01 Probst, M. (2001). *Proper Tail Recursion in C*. (Diplomarbeit). Retrieved from https://www.complang.tuwien.ac.at/schani/diplarb.ps

Sha00 Shapiro, S. (2000). *Thinking about Mathematics: The philosophy of mathematics*. Oxford University Press.

SW00 Strachey, C. & Wadsworth, C. P. (2000). Continuations: A Mathematical Semantics for handling full jumps. Retrieved from http://link.springer.com/article/10.1023/A (originally published in 1974, as technical report, PRG-11, Programming Research Group, University of Oxford)

Ten81 Tennent, R. D. (1981). *Principles of Programming Languages* (, Ed.). Prentice-Hall.

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [pagetotal: 14] [title: Mathematical Foundations of Joy] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j02maf.html] [year: 1994]

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Overview of the language JOY] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j00ovr.html] [year: 1994]

Thu94 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Rationale for Joy, a functional language] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j00rat.html] [year: 1994]

Thu95 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Atomic Programs of Joy] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/j03atm.html] [year: 1995]

Thu05 report: [author: von Thun, Manfred] [institution: Philosophy Department, La Trobe University] [title: Joy website] [type: preprint] [url: http://www.kevinalbrecht.com/code/joy-mirror/joy.html] [year: 2005]

Thu11 online: [author: von Thun, Manfred] [title: Joy Programming Language] [url: http://www.latrobe.edu.au/humanities/research/research-projects/past-projects/joy-programming-language] [year: 2011]

Win93 Winskel, G. (1993). *The Formal Semantics of Programming Languages: An Introduction*. Massachusetts Institute of Technology.