Status and Future of CINT

Reflex as Reflection Database
Object-Oriented CINT
Multi-Threading

Masaharu Goto, Agilent • Philippe Canal, Fermilab • Stefan Roiser, CERN
Paul Russo, Fermilab • Axel Naumann*, CERN

(*) 100% CINT

Status and Future of CINT

What is it? How does ROOT rely on it?

CINT's current status

CINT and Reflex: the new CINT7

CINT's future:

- Dictionary developments
- Object oriented design
- Multi-threading support

What is CINT?

- Reflection data manager
- Dictionary generator
- C++ Parser
- Code and library manager
- Interpreter
- Byte-code compiler

Started in 1991 by Masaharu Goto, originally in C

>300k *real* LOC (excluding comments / empty lines)

ROOT is major "customer" of CINT

What is CINT? Reflection

CINT manages reflection data (type information):

1. Which types are defined?

   Use case: THtml generates doc for all known classes

```
root [0] THtml h
root [1] h.MakeAll(kTRUE)
...
   346 htmldoc/TAxis.html
   345 htmldoc/TBaseClass.html
   344 htmldoc/TBenchmark.html
   343 htmldoc/TBits.html
   342 htmldoc/TBox.html
   341 htmldoc/TBrowser.html
   340 htmldoc/TBtree.html
   339 htmldoc/TBuffer.html
...
```

What is CINT? Reflection

CINT manages reflection data (type information):

2. Which members do they have?
3. Where are they? (Member offset from object address)

Use case: I/O writes all members to file

```
root [0] TH1::Class()->GetStreamerInfo()->ls()
StreamerInfo for class: TH1, version=5
...
  Short_t         fBarOffset    offset=656
  Short_t         fBarWidth     offset=658
  Double_t        fEntries      offset=664
  Double_t        fTsumw        offset=672
  Double_t        fTsumw2       offset=680
```

What is CINT? Reflection

CINT manages reflection data (type information):

4. Which functions does TNeuron have?

   Use case: function lookup in interpreter

```
root [0] TNeuron neuron
root [1] neuron.MoreCoffee()
Error: Can't call TNeuron::MoreCoffee()
```

What is CINT? Reflection

CINT manages reflection data (type information):

5. Call a function
   Use case: Signal / Slot mechanism in GUI,
   e.g. sort TBrowser entries by name if name column header is clicked

```
Connect("Clicked()", "TRootBrowser", fBrowser,
        Form("SetSortMode(=%d)", kViewArrangeByName));
```

What is CINT? Reflection

CINT manages reflection data (type information):

1. Which types are defined?
2. Which members do they have?
3. Where are they?
4. Which functions does TNeuron have?
5. Call a function

What Is CINT?

- Reflection data manager
- Dictionary generator
- C++ Parser
- Code and library manager
- Interpreter

ROOT's dictionary generator rootcint
~~is based on CINT~~
(genreflex+GCCXML is an alternative)

What Is CINT?

- Reflection data manager
- Dictionary generator
- C++ Parser
- Code and library manager
- Interpreter

CINT remembers which macros, libraries were loaded; can re-parse for template instantiations

What Is CINT?

- Reflection data manager
- Dictionary generator
- C++ Parser
- Code and library manager
- Interpreter

ROOT prompt

```
.x Macro.C
gROOT->ProcessLine(...)
```

Current Status

Major developments since last workshop:

- Many limitations removed, e.g. concerning array vs. scalar, auto-loading
- Many new features, e.g. AMD64, MS VisualC++ 2005 support
- Reduced memory footprint (-10MB when running ROOT's benchmarks.C)
- New build system both for CINT itself (configure) and ROOT's CINT (cintdlls-Makefile)
- Bug fixes

CINT, Reflex

Reflex: package to store reflection (=type info) data

Clean C++ API: Type, Scope, Member, e.g.

```
Type::GetDeclaringScope()
Scope::SubTypes_Begin()
Member::GetType()
```

To replace CINT's C structs:

- Easier to maintain due to grouping of functionality (object oriented database), access control, etc
- Fully dynamic, thus smaller memory footprint

CINT + Reflex = CINT7

Needs (more than expected) changes:

- in CINT, e.g. due to non-modular code
- in Reflex, because CINT is a challenging user (unloading of macros, delayed dictionary initialization, etc)

Preview (30% done) in **`$ROOTSYS/cint7`**

Work in progress, parallel to maintenance of CINT5/6

Will become default once stable

CINT's Future

Detailed plan of work:

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

Dictionary Size

Dictionary mainly consists of

- call wrappers: translate string **`"TObject::GetName()"`** to function call

- Function calls to setup dictionary: add **`"TObject"`**, add its function **`"GetName()"`** etc

- Public re-definition of classes to inspect their (otherwise private) members

Dictionary Size

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

Dictionary mainly consists of

- cal
  `"TO`

Extract address of "TObject::GetName()" from library, forward calls directly to that address

- Function calls to setup dictionary: add **`"TObject"`**, add its function **`"GetName()"`** etc

- Public re-definition of classes to inspect their (otherwise private) members

Dictionary Size

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems

Dictionary mainly consists of

- cal
  `"TO`

Extract address of "TObject::GetName()" from library, forward calls directly to that address

- Fu
  ad

Store dictionary data (Reflex objects) in ROOT file, ROOT I/O instead of compiled dictionary

- Public re-definition of classes to inspect their (otherwise private) members

Dictionary Size

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems

Dictionary mainly consists of

- cal
  `"TO`

Extract address of "TObject::GetName()" from library, forward calls directly to that address

- Fu
  ad

Store dictionary data (Reflex objects) in ROOT file, ROOT I/O instead of compiled dictionary

- Pu
  (ot

Calculate member inspection data on the fly or examine (compiler dependent) memory layout

On-Demand Dictionary

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems

Currently: dictionaries for ***all*** types

On-demand: automatically generate and cache only ***needed*** dictionaries

On-demand dictionary example:

1. access `MyClass<int>` but no dictionary yet
2. automatically parse MyClass's header
3. create dictionary for `MyClass<int>`
4. compile (ACLiC) / load dictionary

Great for templates: no 100 dicts for 100 template specializations "just in case"

Object-Oriented CINT

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

For ease of maintenance:

Refactor code to convert current C-based to object oriented design

Started planning of new layout:

- interpreter
- parser
- code manager
- dictionary database: Reflex

- dictionary generator
- byte-code compiler

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

Multi-Threading Support

For ease of maintenance and thread safety:

Refactor code to convert current C-based to object oriented design

Started planning of new layout:

- interpreter: can have multiple per thread
- parser: one for each interpreter
- code manager: libraries / macros
- dictionary database: Reflex (+ thread safety); libraries / macros
- dictionary generator: "static"
- byte-code compiler: "static"

1/process: shared libs,
1/interpreter: macros

Byte-Code Compiler

1. finish CINT7 (Reflex)
2. minimize dictionaries (direct lib calls, dict.root)
3. on-the-fly dictionaries (template dicts)
4. object-oriented CINT (class G__Interpreter)
5. multi-threading support
6. byte-code compiler (loop, scoping problems)

Byte-code:

custom format for efficient execution of macros

Byte-code compiler:

converts macros into byte-code, optimizes it

Problems with scope resolution, loops;

difficult to maintain

Status: re-implementation by Masa (CINT6)

Summary

- CINT amazingly stable: very few changes needed, virtually no API changes
- Well maintained
- Shortcomings known, remedy: long list of planned improvements

Development plans are long scale, with continuous flow of improvements

Benefits arrive on a regular bases!