kaldi-error.h

Go to the documentation of this file.

```
// base/kaldi-error.h

// Copyright 2009-2011  Microsoft Corporation;  Ondrej Glembek;  Lukas Burget;
//                      Saarland University

// See ../../COPYING for clarification regarding multiple authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
// WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
// MERCHANTABLITY OR NON-INFRINGEMENT.
// See the Apache 2 License for the specific language governing permissions and
// limitations under the License.

#ifndef KALDI_BASE_KALDI_ERROR_H_
#define KALDI_BASE_KALDI_ERROR_H_ 1

#include <stdexcept>
#include <string>
#include <cstring>
#include <sstream>
#include <cstdio>

#if _MSC_VER >= 1900 || (!defined(_MSC_VER) && __cplusplus > 199711L) ||
 defined(__GXX_EXPERIMENTAL_CXX0X__)
#define NOEXCEPT(Predicate) noexcept((Predicate))
#else
#define NOEXCEPT(Predicate)
#endif

#include "base/kaldi-types.h"
#include "base/kaldi-utils.h"

/* Important that this file does not depend on any other kaldi headers. */


namespace kaldi {


extern int32 g_kaldi_verbose_level;

extern const char *g_program_name;

inline int32 GetVerboseLevel() { return g_kaldi_verbose_level; }

inline void SetVerboseLevel(int32 i) { g_kaldi_verbose_level = i; }

// Class KaldiLogMessage is invoked from the  KALDI_WARN, KALDI_VLOG and
// KALDI_LOG macros. It prints the message to stderr.  Note: we avoid
// using cerr, due to problems with thread safety.  fprintf is guaranteed
// thread-safe.

// class KaldiWarnMessage is invoked from the KALDI_WARN macro.
class KaldiWarnMessage {
 public:
  inline std::ostream &stream() { return ss; }
  KaldiWarnMessage(const char *func, const char *file, int32 line);
  ~KaldiWarnMessage()  { fprintf(stderr, "%s\n", ss.str().c_str()); }
 private:
  std::ostringstream ss;
};

```

```
78  // class KaldiLogMessage is invoked from the KALDI_LOG macro.
79  class KaldiLogMessage {
80   public:
81    inline std::ostream &stream() { return ss; }
82    KaldiLogMessage(const char *func, const char *file, int32 line);
83    ~KaldiLogMessage() { fprintf(stderr, "%s\n", ss.str().c_str()); }
84   private:
85    std::ostringstream ss;
86  };
87
88  // Class KaldiVlogMessage is invoked from the KALDI_VLOG macro.
89  class KaldiVlogMessage {
90   public:
91    KaldiVlogMessage(const char *func, const char *file, int32 line,
92                     int32 verbose_level);
93    inline std::ostream &stream() { return ss; }
94    ~KaldiVlogMessage() { fprintf(stderr, "%s\n", ss.str().c_str()); }
95   private:
96    std::ostringstream ss;
97  };
98
99
100 // class KaldiErrorMessage is invoked from the KALDI_ERROR macro.
101 // The destructor throws an exception.
102 class KaldiErrorMessage {
103  public:
104   KaldiErrorMessage(const char *func, const char *file, int32 line);
105   inline std::ostream &stream() { return ss; }
106   ~KaldiErrorMessage() NOEXCEPT(false);  // defined in kaldi-error.cc
107  private:
108   std::ostringstream ss;
109 };
110
111
112
113 #ifdef _MSC_VER
114 #define __func__ __FUNCTION__
115 #endif
116
117 // Note on KALDI_ASSERT and KALDI_PARANOID_ASSERT
118 // The original (simple) version of the code was this
119 //
120 // #define KALDI_ASSERT(cond) if (!(cond)) kaldi::KaldiAssertFailure_(__func__,
    __FILE__, __LINE__, #cond);
121 //
122 // That worked well, but we were concerned that it
123 // could potentially cause a performance issue due to failed branch
124 // prediction (best practice is to have the if branch be the commonly
125 // taken one).
126 // Therefore, we decided to move the call into the else{} branch.
127 // A single block {} around if /else  does not work, because it causes
128 // syntax error (unmatched else block) in the following code:
129 //
130 // if (condition)
131 //   KALDI_ASSERT(condition2);
132 // else
133 //   SomethingElse();
134 //
135 // do {} while(0)  -- note there is no semicolon at the end! --- works nicely
136 // and compilers will be able to optimize the loop away (as the condition
137 // is always false).
138 #ifndef NDEBUG
139 #define KALDI_ASSERT(cond) \
140   do { if ((cond)) ; else kaldi::KaldiAssertFailure_(__func__, __FILE__, __LINE__,
    #cond);} while(0)
141 #else
142 #define KALDI_ASSERT(cond)
143 #endif
144 // also see KALDI_COMPILE_TIME_ASSERT, defined in base/kaldi-utils.h,
145 // and KALDI_ASSERT_IS_INTEGER_TYPE and KALDI_ASSERT_IS_FLOATING_TYPE,
146 // also defined there.
147 #ifdef KALDI_PARANOID // some more expensive asserts only checked if this defined
148 #define KALDI_PARANOID_ASSERT(cond) \
149   do { if ((cond)) ; else kaldi::KaldiAssertFailure_(__func__, __FILE__, __LINE__,
    #cond);} while(0)
```

```
#else
#define KALDI_PARANOID_ASSERT(cond)
#endif


#define KALDI_ERR kaldi::KaldiErrorMessage(__func__, __FILE__, __LINE__).stream()
#define KALDI_WARN kaldi::KaldiWarnMessage(__func__, __FILE__, __LINE__).stream()
#define KALDI_LOG kaldi::KaldiLogMessage(__func__, __FILE__, __LINE__).stream()

#define KALDI_VLOG(v) if (v <= kaldi::g_kaldi_verbose_level)     \
           kaldi::KaldiVlogMessage(__func__, __FILE__, __LINE__, v).stream()

inline bool IsKaldiError(const std::string &str) {
  return(!strncmp(str.c_str(), "ERROR ", 6));
}

void KaldiAssertFailure_(const char *func, const char *file,
                         int32 line, const char *cond_str);


}  // namespace kaldi

#endif  // KALDI_BASE_KALDI_ERROR_H_
```