TOPPERS C++ Wrapper Library

高木 信尚 （TOPPERSプロジェクト）

Version 0.9.2

2004年01月22日

目次

TOPPERS C++ Wrapper Library v
特徴 v
開発環境 v
制約事項 v
モジュール索引 vii
名前空間索引 viii
階層索引 ix
構成索引 xi
ファイル索引 xiv
ページ索引 xv
モジュールの解説 2
サービスコールスタブ 2
カーネルオブジェクト管理 3
メモリアロケータ 14
プログラム診断 15
同期機構 17
toppers 20
クラスの解説 33
toppers::alarmhandler 33
toppers::alarmhandler_body 36
toppers::allocator 39
toppers::allocator::rebind 42
toppers::cpuexception 43
toppers::cpuexception_body 45
toppers::cpulock 47
toppers::cyclichandler 49
toppers::cyclichandler_body 52
toppers::dataqueue 55
toppers::dispatcher 58
toppers::dispatcher::pending 59
toppers::error_handler_diagnostics 60
toppers::eventflag 62
toppers::fixed_new_pool 65
toppers::fixed_simple_pool 66
toppers::fixed_sized_allocator 68
toppers::fixed_sized_new 69
toppers::fmempool 72
toppers::fncode_stub 74
toppers::has_obj 89
toppers::has_svc 90
toppers::interrupt 91
toppers::interrupthandler 92
toppers::interrupthandler_body 94
toppers::isr 96
toppers::isr_body 98
toppers::kernelstatus 100

toppers::kernelstatus::context 102
toppers::lock 103
toppers::mailbox 105
toppers::messagebuffer 107
toppers::mutex 109
toppers::non_task_context_body 112
toppers::null_diagnostics 114
toppers::null_tracer 116
toppers::object_controller 117
toppers::object_id 118
toppers::outline_stub 119
toppers::overrunhandler 135
toppers::overrunhandler_body 137
toppers::polymorphic_diagnostics 140
toppers::raw_stub 143
toppers::readyqueue 158
toppers::recursive_lock 159
toppers::rendezvous 161
toppers::rendezvousport 163
toppers::semaphore 166
toppers::syslog_tracer 169
toppers::task 172
toppers::task_body 176
toppers::task_body::terminator 181
toppers::task_context_body 182
toppers::taskexception 185
toppers::taskexception_body 188
toppers::variable_new_pool 192
toppers::variable_simple_pool 193
toppers::variable_sized_new 195
toppers::vmempool 198
ファイルの解説 200
toppers/alarmhandler.hpp 200
toppers/allocator.hpp 202
toppers/assert.hpp 204
toppers/complement.hpp 206
toppers/config.hpp 208
toppers/context.hpp 210
toppers/cpuexception.hpp 212
toppers/cyclichandler.hpp 214
toppers/dataqueue.hpp 216
toppers/diagnostics.hpp 218
toppers/eventflag.hpp 220
toppers/fmempool.hpp 222
toppers/fncode_stub.hpp 224
toppers/interrupt.hpp 226
toppers/kernel.hpp 228
toppers/kernelstatus.hpp 230
toppers/mailbox.hpp 232
toppers/messagebuffer.hpp 234
toppers/mutex.hpp 236
toppers/object.hpp 238

toppers/outline_stub.hpp 240
toppers/overrunhandler.hpp 241
toppers/raw_stub.hpp 243
toppers/rendezvous.hpp 245
toppers/semaphore.hpp 247
toppers/sync.hpp 249
toppers/task.hpp 251
toppers/taskexception.hpp 253
toppers/trace.hpp 255
toppers/vmempool.hpp 257
ページの解説 259
ライセンス 259
コーディング規約 259
チュートリアル 260
環境設定 260
Hello, World! 261
TODO一覧 262
索引 263

TOPPERS C++ Wrapper Library

`TOPPERS C++ Wrapper Library' はμITRON 4.0仕様に準拠したTOPPERSカーネルの機能を ラッピングした C++ のテンプレートライブラリである。

組み込みシステムでの利用を目的とするため、実行速度とプログラムサイズの両面において、 効率の良いコードが生成されることを目指して設計している。

本ライブラリは TOPPERS カーネルの薄いラッパーであり、カーネルに対して何らかの機能を 追加することを目的とするものではない。 しかしながら、本ライブラリを使用することで、より上位のライブラリを実装する上での強力な 基盤を得ることができる。

特徴

- テンプレートをフル活用したジェネリックライブラリ
- 原則として仮想関数を使用せず、コンパイル時に型を解決する
- コンテキストや待機方法をパラメータ化しつつも、静的な処理選択が可能
- 簡潔かつ安全なコーディングの支援
- カーネルごとの機能のサポート状況を自動判別
- 例外処理に絡む空間効率の低下を最小限に抑える

開発環境

- Binutils 2.13以降
- GCC 3.2以上 （TOPPERS対応パッチが必要）
- Newlib 1.11.0以降
- TOPPERS/JSPカーネル 1.4以降、 または TOPPERS/FI4カーネル 1.0以降

制約事項

- Embedded C++には対応していない
- TOPPERSカーネル以外のμITRONカーネルの場合、ある程度のポーティングが必要

日付:

Date
2004/01/22 06:23:46

作者:

高木信尚 （TOPPERSプロジェクト）

TOPPERS C++ Wrapper Library モジュール索引

TOPPERS C++ Wrapper Library モジュール

すべてのモジュールの一覧です。

サービスコールスタブ 2
カーネルオブジェクト管理 3
メモリアロケータ 14
プログラム診断 15
同期機構 17

TOPPERS C++ Wrapper Library 名前空間索引

TOPPERS C++ Wrapper Library 名前空間一覧

名前空間の一覧です。

toppers (ライブラリが使用するトップレベルの名前空間) 20

TOPPERS C++ Wrapper Library 階層索引

TOPPERS C++ Wrapper Library クラス階層

この継承一覧はおおまかにはソートされていますが、完全にアルファベット順でソートされてはいません。

toppers::alarmhandler< Id, Diagnostics, Stub > 33
toppers::allocator< T, Pool > 39
toppers::allocator< T, Pool >::rebind< T1 > 42
toppers::cpuexception< Diagnostics, Stub > 43
toppers::cpuexception_body< Diagnostics, Stub > 45
toppers::cpulock< Diagnostics, Stub > 47
toppers::cyclichandler< Id, Diagnostics, Stub > 49
toppers::dataqueue< Id, Diagnostics, Stub > 55
toppers::dispatcher< Diagnostics, Stub > 58
toppers::dispatcher< Diagnostics, Stub >::pending 59
toppers::error_handler_diagnostics< T > 60
    toppers::polymorphic_diagnostics< T > 140
toppers::eventflag< Id, Diagnostics, Stub > 62
toppers::fixed_new_pool< Size > 65
toppers::fixed_simple_pool< N, Size > 66
toppers::fixed_sized_allocator< Size, Pool > 68
toppers::fixed_sized_new< Host, Pool, Allocator > 69
toppers::fmempool< Id, Diagnostics, Stub > 72
toppers::fncode_stub< Stub > 74
toppers::has_obj< ObjType > 89
toppers::has_svc< Fncd > 90
toppers::interrupt< Diagnostics, Stub > 91
toppers::interrupthandler< Diagnostics, Stub > 92
toppers::isr< Id, Diagnostics, Stub > 96
toppers::kernelstatus< Diagnostics, Stub > 100
toppers::kernelstatus< Diagnostics, Stub >::context 102
toppers::lock< LockObj, SenseKernel > 103
toppers::mailbox< Id, Diagnostics, Stub > 105
toppers::messagebuffer< Id, Diagnostics, Stub > 107
toppers::mutex< Id, Diagnostics, Stub > 109
toppers::non_task_context_body< Diagnostics, Stub > 112
    toppers::alarmhandler_body< Diagnostics, Stub > 36
    toppers::cyclichandler_body< Diagnostics, Stub > 52
    toppers::interrupthandler_body< Diagnostics, Stub > 94
    toppers::isr_body< Diagnostics, Stub > 98

toppers::overrunhandler_body< Diagnostics, Stub > 137
toppers::null_diagnostics 114
toppers::null_tracer 116
toppers::object_controller< Id, Object, Stub > 117
toppers::object_id< Id > 118
toppers::outline_stub 119
toppers::overrunhandler< Diagnostics, Stub > 135
toppers::raw_stub< Exist > 143
toppers::readyqueue< Diagnostics, Stub > 158
toppers::recursive_lock< LockObj, SenseKernel > 159
toppers::rendezvous< Diagnostics, Stub > 161
toppers::rendezvousport< Id, Diagnostics, Stub > 163
toppers::semaphore< Id, Diagnostics, Stub > 166
toppers::syslog_tracer< Count, BufferSize, Diagnostics > 169
toppers::task< Id, Diagnostics, Stub > 172
toppers::task_body< Diagnostics, Stub >::terminator 181
toppers::task_context_body< Diagnostics, Stub > 182
toppers::task_body< Diagnostics, Stub > 176
toppers::taskexception_body< Diagnostics, Stub > 188
toppers::taskexception< Diagnostics, Stub > 185
toppers::variable_new_pool 192
toppers::variable_simple_pool< TotalSize > 193
toppers::variable_sized_new< Host, Pool > 195
toppers::vmempool< Id, Diagnostics, Stub > 198

TOPPERS C++ Wrapper Library 構成索引

TOPPERS C++ Wrapper Library 構成

クラス、構造体、共用体の解説です。

toppers::alarmhandler< Id, Diagnostics, Stub > (アラームハンドラ機能をカプセル化したテンプレートクラス) 33

toppers::alarmhandler_body< Diagnostics, Stub > (アラームハンドラの本体動作を定義する基底クラス) 36

toppers::allocator< T, Pool > (メモリプールを使用したSTL互換のアロケータクラス) 39

toppers::allocator< T, Pool >::rebind< T1 > (異なる型のオブジェクトを割り付け可能にするためのテンプレートクラス) 42

toppers::cpuexception< Diagnostics, Stub > (CPU例外処理の機能をカプセル化したテンプレートクラス) 43

toppers::cpuexception_body< Diagnostics, Stub > (CPU例外処理ハンドラの本体動作を定義する基底クラス) 45

toppers::cpulock< Diagnostics, Stub > (CPUロック状態の制御をカプセル化したテンプレートクラス) 47

toppers::cyclichandler< Id, Diagnostics, Stub > (周期ハンドラの機能をカプセル化したテンプレートクラス) 49

toppers::cyclichandler_body< Diagnostics, Stub > (周期ハンドラの本体動作を定義する基底クラス) 52

toppers::dataqueue< Id, Diagnostics, Stub > (データキューの機能をカプセル化したテンプレートクラス) 55

toppers::dispatcher< Diagnostics, Stub > (ディスパッチャの制御をカプセル化したテンプレートクラス) 58

toppers::dispatcher< Diagnostics, Stub >::pending (ディスパッチ保留状態参照クラス) 59

toppers::error_handler_diagnostics< T > (エラーハンドラ関数を呼び出す診断クラス) 60

toppers::eventflag< Id, Diagnostics, Stub > (イベントフラグの機能をカプセル化したテンプレートクラス) 62

toppers::fixed_new_pool< Size > (グローバルな new および delete 演算子による固定長メモリプール) 65

toppers::fixed_simple_pool< N, Size > (排他制御のない固定長メモリプール) 66

toppers::fixed_sized_allocator< Size, Pool > (固定サイズのアロケータを定義するためのヘルパークラス) 68

toppers::fixed_sized_new< Host, Pool, Allocator > (指定メモリプールを用いた new および delete 演算子) 69

toppers::fmempool< Id, Diagnostics, Stub > (固定長メモリプールの機能をカプセル化したテンプレートクラス) 72

toppers::fncode_stub< Stub > (機能コードを用いたサービスコールスタブの定義) 74
toppers::has_obj< ObjType > (カーネルが指定したカーネルオブジェクトをサポートしているかどうかを判別する) 89
toppers::has_svc< Fncd > (カーネルが指定したサービスコールをサポートしているかどうかを判別する) 90
toppers::interrupt< Diagnostics, Stub > (割り込みを管理・制御するテンプレートクラス) ..91
toppers::interrupthandler< Diagnostics, Stub > (割込みハンドラを管理・制御するテンプレートクラス) 92
toppers::interrupthandler_body< Diagnostics, Stub > (割込みハンドラの本体動作を定義する基底クラス) 94
toppers::isr< Id, Diagnostics, Stub > (割込みサービスルーチンの機能をカプセル化したテンプレートクラス) 96
toppers::isr_body< Diagnostics, Stub > (割込みサービスルーチンの本体動作を定義する基底クラス) 98
toppers::kernelstatus< Diagnostics, Stub > (カーネル状態に関する操作をカプセル化したテンプレートクラス) 100
toppers::kernelstatus< Diagnostics, Stub >::context (実行中コンテキスト参照クラス) 102
toppers::lock< LockObj, SenseKernel > (クリティカルセクションを生成するためのテンプレートクラス) 103
toppers::mailbox< Id, Diagnostics, Stub > (メールボックスの機能をカプセル化したテンプレートクラス) 105
toppers::messagebuffer< Id, Diagnostics, Stub > (メッセージバッファの機能をカプセル化したテンプレートクラス) 107
toppers::mutex< Id, Diagnostics, Stub > (ミューテックスの機能をカプセル化したテンプレートクラス) 109
toppers::non_task_context_body< Diagnostics, Stub > (非タスクコンテキストのハンドラ本体定義用基底クラス) 112
toppers::null_diagnostics (特に何も行わない診断クラス) 114
toppers::null_tracer (何も出力しないデータトレースクラス) 116
toppers::object_controller< Id, Object, Stub > (カーネルオブジェクトの生成・削除・状態参照を制御するクラス) 117
toppers::object_id< Id > (カーネルオブジェクトのID番号を管理するクラス) 118
toppers::outline_stub (外部関数を用いたサービスコールスタブの定義) 119
toppers::overrunhandler< Diagnostics, Stub > (オーバーランハンドラの機能をカプセル化したテンプレートクラス) 135
toppers::overrunhandler_body< Diagnostics, Stub > (オーバーランハンドラの本体動作を定義する基底クラス) 137
toppers::polymorphic_diagnostics< T > (各処理を仮想関数として提供する診断クラス) 140
toppers::raw_stub< Exist > (最も単純なサービスコールスタブ) 143
toppers::readyqueue< Diagnostics, Stub > (レディキューの操作をカプセル化したテンプレートクラス) 158

toppers::recursive_lock< LockObj, SenseKernel > (ネスト可能なクリティカルセクションを生成するためのテンプレートクラス) 159
toppers::rendezvous< Diagnostics, Stub > (ランデブ番号をカプセル化したテンプレートクラス) 161
toppers::rendezvousport< Id, Diagnostics, Stub > (ランデブポートの機能をカプセル化したテンプレートクラス) 163
toppers::semaphore< Id, Diagnostics, Stub > (セマフォの機能をカプセル化したテンプレートクラス) 166
toppers::syslog_tracer< Count, BufferSize, Diagnostics > (Syslogを用いたデータトレースクラス) 169
toppers::task< Id, Diagnostics, Stub > (タスク機能をカプセル化したテンプレートクラス) 172
toppers::task_body< Diagnostics, Stub > (タスク本体のデータおよび動作を定義する基底クラス) 176
toppers::task_body< Diagnostics, Stub >::terminator (自タスクを終了させる場合にスローする例外クラス) 181
toppers::task_context_body< Diagnostics, Stub > (タスクコンテキストの本体定義用基底クラス) 182
toppers::taskexception< Diagnostics, Stub > (タスク例外処理機能をカプセル化したテンプレートクラス) 185
toppers::taskexception_body< Diagnostics, Stub > (タスク例外処理ルーチンの本体動作を定義する基底クラス) 188
toppers::variable_new_pool (グローバルな **new** および **delete** 演算子による可変長メモリプール) 192
toppers::variable_simple_pool< TotalSize > (排他制御のない可変長メモリプール) 193
toppers::variable_sized_new< Host, Pool > (指定メモリプールを用いた **new** および **delete** 演算子) 195
toppers::vmempool< Id, Diagnostics, Stub > (可変長メモリプールの機能をカプセル化したテンプレートクラス) 198

TOPPERS C++ Wrapper Library ファイル索引

TOPPERS C++ Wrapper Library ファイル一覧

これはファイル一覧です。

toppers/alarmhandler.hpp (アラームハンドラを管理・制御するクラスの定義) 200
toppers/allocator.hpp (メモリアロケータに関するクラスの定義　このヘッダファイルでは以下のクラスを定義している。) 202
toppers/assert.hpp (アサーション機能のためのマクロ定義) 204
toppers/complement.hpp (カーネル間の差異を補完・吸収するための宣言・定義) 206
toppers/config.hpp (使用するカーネルやコンパイラ等の環境に依存する設定) 208
toppers/context.hpp (タスクおよび非タスクコンテキストの本体動作に関する基本定義) .210
toppers/cpuexception.hpp (CPU例外ハンドラを管理・制御するクラスの定義) 212
toppers/cyclichandler.hpp (周期ハンドラを管理・制御するクラスの定義) 214
toppers/dataqueue.hpp (データキューを管理・制御するクラスの定義) 216
toppers/diagnostics.hpp (プログラム診断クラスの定義) 218
toppers/eventflag.hpp (イベントフラグの管理・制御を行うクラスの定義) 220
toppers/fmempool.hpp (固定長メモリプールを管理・制御するクラスの定義) 222
toppers/fncode_stub.hpp (機能コードを用いたサービスコールスタブの定義) 224
toppers/interrupt.hpp (割込み管理に関するクラスの定義) 226
toppers/kernel.hpp (カーネル機能をライブラリから使用するための補助的な定義) 228
toppers/kernelstatus.hpp (カーネル状態を管理・制御するクラスの定義) 230
toppers/mailbox.hpp (メールボックスを管理・制御するクラスの定義) 232
toppers/messagebuffer.hpp (メッセージバッファを管理・制御するクラスの定義) 234
toppers/mutex.hpp (ミューテックスを管理・制御するクラスの定義) 236
toppers/object.hpp (各カーネルオブジェクトで共通に利用するクラス定義) 238
toppers/outline_stub.hpp (外部関数を用いたサービスコールスタブの定義) 240
toppers/overrunhandler.hpp (オーバーランハンドラを管理・制御するクラスの定義) 241
toppers/raw_stub.hpp (最も単純なサービスコールスタブの定義) 243
toppers/rendezvous.hpp (ランデブを管理・制御するためのクラスの定義) 245
toppers/semaphore.hpp (セマフォを管理・制御するクラスの定義) 247
toppers/sync.hpp (同期機構に関するクラスの定義) 249
toppers/task.hpp (タスクを管理・制御するためのクラス定義) 251
toppers/taskexception.hpp (タスク例外処理を管理・制御するクラスの定義) 253
toppers/trace.hpp (実行時トレースに関する宣言・定義) 255
toppers/vmempool.hpp (可変長メモリプールを管理・制御するクラスの定義) 257

TOPPERS C++ Wrapper Library ページ索引

TOPPERS C++ Wrapper Library 関連ページ

関連ページの一覧です。

ライセンス 259
コーディング規約 259
チュートリアル 260
TODO一覧 262

TOPPERS C++ Wrapper Library モジュールの解説

サービスコールスタブ

μITRON 4.0仕様のサービスコールを間接的に呼び出すためのスタブクラス

構成

- class **fncode_stub**
  機能コードを用いたサービスコールスタブの定義

- class **outline_stub**
  外部関数を用いたサービスコールスタブの定義

- class **raw_stub**
  最も単純なサービスコールスタブ

解説

サービスコールスタブ モジュールは、TOPPERS カーネルのサービスコールの呼出し方法をパラメータ化 するための機構である。 ユーザ独自のサービスコールスタブを定義することで、μITRON 以外のカーネルや、μITRON の異なる バージョンのカーネルを使用することも、原理的には可能である。

カーネル間の差異の吸収:

主として、サポートされるサービスコールの違いを意識することなくライブラリを構築できる。 存在しないサービスコールを呼出そうとした場合、コンパイル時にエラーにすることも、実行時に エラーにすることもできる。また、ユーザの手によって、不足したサービスコールを補完することも 可能である。

使用するサービスコールスタブをテンプレート引数にすることで、該当するサービスコールを実際に （テンプレートではなく実体として）呼出すようなコードを記述するまで、コンパイルエラーになる ことを回避することができる。

引数や返却値の型の統一:

μITRON 仕様のカーネルは、仕様を満たしていたとしても、サービスコールの引数の型が 微妙に異なる場合がある。また、同一のカーネルでもターゲットによって異なる可能性もある。

++では、そうした微妙な型の差異が多重定義等に思わぬ障害を引き起こす可能性があるため、 サービスコールスタブによって型を統一することができる。

例外指定の付加:

μITRON カーネルのヘッダは、主としてC言語で使用することを想定して実装されているため、C++の 例外指定がない場合がある。

サービスコールスタブを用いることで、各サービスコールに例外指定を追加することができる。 これにより、例外処理に関連した不要なコードが生成されることを抑止し、プログラムの効率化を図る ことができる。

カーネルオブジェクト管理

μITRONのカーネルオブジェクトを管理するためのテンプレートクラス群

構成

- class **alarmhandler**
  アラームハンドラ機能をカプセル化したテンプレートクラス

- class **alarmhandler_body**
  アラームハンドラの本体動作を定義する基底クラス

- class **cpuexception**
  CPU例外処理の機能をカプセル化したテンプレートクラス.

- class **cpuexception_body**
  CPU例外処理ハンドラの本体動作を定義する基底クラス.

- class **cpulock**
  CPUロック状態の制御をカプセル化したテンプレートクラス.

- class **cyclichandler**
  周期ハンドラの機能をカプセル化したテンプレートクラス

- class **cyclichandler_body**
  周期ハンドラの本体動作を定義する基底クラス

- class **dataqueue**
  データキューの機能をカプセル化したテンプレートクラス

- class **dispatcher**
  ディスパッチャの制御をカプセル化したテンプレートクラス

- struct **dispatcher::pending**
  ディスパッチ保留状態参照クラス

- class **eventflag**
  イベントフラグの機能をカプセル化したテンプレートクラス

- class **fmempool**
  固定長メモリプールの機能をカプセル化したテンプレートクラス

- struct **has_obj**
  カーネルが指定したカーネルオブジェクトをサポートしているかどうかを判別する

- struct **has_svc**
  カーネルが指定したサービスコールをサポートしているかどうかを判別する

- class **interrupt**
  割り込みを管理・制御するテンプレートクラス

- class **interrupthandler**
  割込みハンドラを管理・制御するテンプレートクラス

- class **interrupthandler_body**
  割込みハンドラの本体動作を定義する基底クラス

- class **isr**
  割込みサービスルーチンの機能をカプセル化したテンプレートクラス

- class **isr_body**
  割込みサービスルーチンの本体動作を定義する基底クラス

- class **kernelstatus**
  カーネル状態に関する操作をカプセル化したテンプレートクラス

- struct **kernelstatus::context**
  実行中コンテキスト参照クラス

- class **mailbox**
  メールボックスの機能をカプセル化したテンプレートクラス

- class **messagebuffer**
  メッセージバッファの機能をカプセル化したテンプレートクラス

- class **mutex**
  ミューテックスの機能をカプセル化したテンプレートクラス

- class **non_task_context_body**
  非タスクコンテキストのハンドラ本体定義用基底クラス

- class **object_controller**
  カーネルオブジェクトの生成・削除・状態参照を制御するクラス

- class **object_id**
  カーネルオブジェクトのID番号を管理するクラス

- class **overrunhandler**
  オーバーランハンドラの機能をカプセル化したテンプレートクラス

- class **overrunhandler_body**
  オーバーランハンドラの本体動作を定義する基底クラス

- class **readyqueue**
  レディキューの操作をカプセル化したテンプレートクラス

- class **rendezvous**
  ランデブ番号をカプセル化したテンプレートクラス

- class **rendezvousport**
  ランデブポートの機能をカプセル化したテンプレートクラス

- class **semaphore**
  セマフォの機能をカプセル化したテンプレートクラス

- class **task**
  タスク機能をカプセル化したテンプレートクラス

- class **task_body**
  タスク本体のデータおよび動作を定義する基底クラス

- class **task_body::terminator**
  自タスクを終了させる場合にスローする例外クラス

- class **task_context_body**

タスクコンテキストの本体定義用基底クラス

- class **taskexception**
  タスク例外処理機能をカプセル化したテンプレートクラス

- class **taskexception_body**
  タスク例外処理ルーチンの本体動作を定義する基底クラス

- class **vmempool**
  可変長メモリプールの機能をカプセル化したテンプレートクラス

マクロ定義

- #define
  TOPPERS_BIND_HANDLER(func) ::toppers::detail::bind_context_handler<__typeof__(&(func)), &(func)>(&(func))
  任意型の関数のハンドラ関数への結びつけ

関数

- template<class Body> void **toppers::alarmhandler_entry** (VP_INT)
  アラームハンドラのテンプレート

- template<class Body> void **toppers::cpuexception_entry** (VP p_exinf)
  CPU例外ハンドラのテンプレート.

- template<class Body> void **toppers::cyclichandler_entry** (VP_INT)
  周期ハンドラのテンプレート

- template<class Body> void **toppers::interrupthandler_entry** ()
  割込みハンドラのテンプレート

- template<class Body> void **toppers::isr_entry** (VP_INT)
  割込みサービスルーチンのテンプレート

- template<bool Sense> bool **toppers::sense_kernel** ()
  カーネルが動作中かどうかを判別する。

- template<class Body> void **toppers::overrunhandler_entry** (ID, VP_INT)

オーバーランハンドラのテンプレート

- template<class Body> void **toppers::task_entry** (VP_INT)
  タスクの起動番地として指定するためのテンプレート関数

- template<class Body> void **toppers::taskexception_entry** (TEXPTN texptn, VP_INT exinf)
  タスク例外処理ルーチンのテンプレート

解説

カーネルオブジェクト管理モジュールは、TOPPERS カーネルの機能の大部分をラッピングするための クラス群であり、本ライブラリの中核となるモジュールにあたる。

本モジュールでは、カーネルを管理・制御するための要素の多くをパラメータ化している。以下に、 それらのパラメータを記載する。

共通のテンプレート引数

本モジュールに属している主要なテンプレートクラスは、そのテンプレート引数として、以下に挙げる ものの一部または全部を有している。

カーネルオブジェクトID番号（ID *Id* ）:

本ライブラリでは、ID番号は各クラスのテンプレート引数として指定することで、型の一部に 埋め込まれる。これにより、型を指定することでカーネルオブジェクトを特定することが可能に なり、カーネルオブジェクト自体を静的なパラメータとすることを可能にする。

テンプレート引数*Id* に0を指定（デフォルトは0）した場合、インスタンス内部にID番号を保持 することができる。これにより、実行時にインスタンスとID番号を関連付けることができる。

負のID番号には対応していない。ただし、テンプレート引数*Id* を0にし、動的に負のID番号に 関連付けることはできる。

多くの場合、*Id* のデフォルト値は0である。

プログラム診断ポリシー（class *Diagnostics* ）:

サービスコールの呼出しに失敗した場合等の動作を規定する。

テンプレート引数に*Diagnostics* を持つクラスは、*Diagnostics* から private 継承している。 そのため、*Diagnostics* は、プログラム診断という本来の用途を超えて、各クラスの動作を細かく カスタマイズすることができる。例えば、*Diagnostics* に仮想デストラクタやその他の仮想関数を 定義することで、動的多態性を持たせることも可能である。

多くの場合、*Diagnostics* のデフォルト値には*TOPPERS_DEFAULT_DIAGNOSIS* が指定されている。 *TOPPERS_DEFAULT_DIAGNOSIS* は、ユーザが独自に定義しない限り、*ignorant_diagnosis* である。

サービスコールスタブポリシー（class *Stub* ）:

クラスのメンバ関数がカーネルにアクセスする際にしようするスタブを指定する。

多くの場合、*Stub* のデフォルト値には*TOPPERS_DEFAULT_STUB* が指定されている。*TOPPERS_DEFAULT_STUB* は、ユーザが独自に定義しない限り、*raw_stub*<> である。

ボディ（*Body* ）:

タスク、およびハンドラや処理ルーチンが必要となるカーネルオブジェクトでは、ボディと呼ばれる クラスを定義することで、個別のデータや動作を定義する。

ボディクラスはハンドラ等に用いる関数のテンプレート引数として指定する他、カーネルオブジェクトの 生成時に*create* メンバ関数のテンプレート引数として指定することもできる。

共通のメソッド

本モジュールに属しているクラスの多くは、以下に挙げる共通のメソッドを有している。 こうした共通のメソッドを提供することにより、ライブラリの習得を容易にし、かつ上位のライブラリの ジェネリックな設計を支援する。

カーネルオブジェクトの生成（*create* ）:

カーネルオブジェクトの生成には*create* という名称のメンバ関数を使用する。

e create メンバ関数は*creation* 型へのポインタを引数として受け取るが、これは μITRON 4.0仕様の*T_CYYY* 構造体に相当する。

e create メンバ関数は、それが属しているクラスのテンプレート引数*Id* が0以外の 場合には、対応する*cre_yyy* サービスコールを呼出す。*Id* が0の場合には、対応する *acre_yyy* サービスコールを呼出し、自動的に割付けられたID番号をインスタンスに格納 する。

e create メンバ関数は、*creation* 型へのポインタを受け取る以外に、*creation* 型に含まれる各フィールドを個別に渡すこともできる。

また、タスクの他、ハンドラや処理ルーチンを必要とするカーネルオブジェクトでは、*Body* をテンプレート引数で指定する方法もある。

カーネルオブジェクトの削除（*destroy* ）:

カーネルオブジェクトの削除には@ destroy という名称のメンバ関数を使用する。 これは内部的に*del_yyy* サービスコールを呼出すが、名称が*delete* ではないのは、 C++の予約語と衝突するためである。

クラスのテンプレート引数*Id* が0の場合、*destroy* メンバ関数によってカーネル オブジェクトが削除されると、それまで保持していたID番号がリセットされる。

カーネルオブジェクトの状態参照（*refer* ）:

カーネルオブジェクトの状態参照には*refer* という名称のメンバ関数を使用する。
e refer メンバ関数は*reference* 型へのポインタを引数として受け取るが、これは μITRON 4.0仕様の*T_RYYY* 構造体に相当する。

カーネルオブジェクトの定義（*define* ）：

カーネルオブジェクトの定義には*define* という名称のメンバ関数を使用する
e define メンバ関数は*definition* 型へのポインタを引数として受け取るが、これは μITRON 4.0仕様の*T_DYYY* 構造体に相当し、内部的に*def_yyy* サービスコールが 呼出される。

e define メンバ関数は*create* に類似するが、以下の点が異なる。

- 静的メンバ関数である。
- インスタンスとID番号の関連付け操作は行われない。

カーネルオブジェクトの定義取り消し（*undefine* ）：

カーネルオブジェクトの定義取り消しには*undefine* という名称のメンバ関数を使用する
このメンバ関数は、*def_yyy* サービスコールに*NULL* ポインタを渡すことで実現している。

生のID番号の取り出し（*get_unsafe_id* ）：

カーネルオブジェクトID番号を管理するクラスのインスタンスから、生のID番号を取り出すには *get_unsafe_id* という名称のフリー関数（非メンバ関数）を使用する。
生のID番号を取得できると便利なことが多いが、反面クラスによるカプセル化を破綻させる危険性 も併せ持っている。 本ライブラリでは、ユーザにそうした危険性に対する注意を喚起する意味で、*unsafe* という 語を使用するとともに、メンバ関数ではなくフリー関数として提供している。

コンテキストの指定

サービスコールによっては、タスクコンテキストと非タスクコンテキストの両方から使用できる。 どちらのコンテキスト用のサービスコールを呼出すかを指定するには、専用の列挙定数を用いる。

- タスクコンテキスト指定（*task_context* ）
- 非タスクコンテキスト指定（*non_task_context* ）
- コンテキスト非依存（*context_independent* ） なお、*task_context_body* または *non_task_context_body* テンプレートクラスから継承した ボディクラスでは、*context_independent* 定数が、*task_context* または*non_task_context* のいずれかに定義されているため、ボディクラスのメンバ関数内では、*context_independent* を 指定した場合でも効率の良いコード生成が期待できる。

待機方法の指定

μITRON では、タスクを待ち状態にする可能性のあるサービスコールでは、その待機方法として、 永久待ち、ポーリング、およびタイムアウト指定を選択することができる。

カーネル本来の機能では、待機方法をパラメータ化するには、TMO 型の値指定によって実行時に 選択していたが、本ライブラリでは、下記の列挙定数を指定することでコンパイル時に選択することが 可能である。

- 永久待ち（*forever* ）
- ポーリング（*polling* ）
- タイムアウト指定（*TMO* 型のタイムアウト値）

マクロ定義の解説

#define TOPPERS_BIND_HANDLER(func) ::toppers::detail::bind_context_handler<__typeof__(&(func)), &(func)>(&(func))

引数:

func 関数へのポインタ（外部関数は使用不可）

返却値:

ハンドラへのポインタ

TODO:

将来のバージョンでは可能な限りGCC以外のコンパイラにも対応させる。

型の異なる関数を、*VP_INT* の拡張情報を受取るハンドラ関数に結びつける。 結びつけることができる関数は、スカラー型または*VP_INT* からの変換コンストラクタを 持つクラスを唯一の引数とする関数でなければならない。

```
#include <toppers/cyclichandler.hpp>
#include <toppers/raw_stub.hpp>

//  周期的に指定したタスクを起動する。
toppers::cyclichandler<> create_activator(ID tskid, RELTIM cyctim = 1)
{
   using namespace toppers;
   cyclichandler<> cyc;

   ER ercd = cyc.create(TA_STA, VP_INT(tskid),
                     TOPPERS_BIND_HANDLER(raw_stub<>::stub_iact_tsk),
                     cyctim);
   return cyc;
}
```

関数の解説

template<class Body> void toppers::alarmhandler_entry (VP_INT *exinf*) [inline]

参照:

alarmhandler_body

テンプレート引数*Body* として指定したボディクラスを本体とするアラームハンドラを定義する。 この関数を*T_CALM* 構造体の*almhdr* フィールドに設定することができる。

注意:

CRE_ALM （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void cpuexception_entry (VP *p_exinf*) [inline]

参照:

cpuexception_body

テンプレート引数*Body* として指定したボディクラスを本体とするCPU例外ハンドラを定義する。 この関数を*T_DEXC* 構造体の*exchdr* フィールドに設定することができる。

注意:

DEF_EXC （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void toppers::cyclichandler_entry (VP_INT *exinf*) [inline]

参照:

cyclichandler_body

テンプレート引数*Body* として指定したボディクラスを本体とする周期ハンドラを定義する。この関数を*T_CCYC* 構造体の*cychdr* フィールドに設定することができる。

注意:

CRE_CYC （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void toppers::interrupthandler_entry () [inline]

参照:

interrupthandler_body

テンプレート引数*Body* として指定したボディクラスを本体とする割込みハンドラを定義する。 この関数を*T_DINH* 構造体の*inthdr* フィールドに設定することができる。

注意:

DEF_INH （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void toppers::isr_entry (VP_INT *exinf*) [inline]

参照:

isr_body

テンプレート引数*Body* として指定したボディクラスを本体とする割込みサービスルーチンを定義する。 この関数を*T_CISR* 構造体の*isr* フィールドに設定することができる。

注意:

CRE_ISR （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void toppers::overrunhandler_entry (ID *tskid*, VP_INT *exinf*) [inline]

参照:

overrunhandler_body

テンプレート引数*Body* として指定したボディクラスを本体とするオーバーランハンドラを定義する。 この関数を*T_DOVR* 構造体の*ovrhdr* フィールドに設定することができる。

注意:

DEF_OVR （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<bool Sense> bool sense_kernel () [inline]

vsns_ini サービスコールを呼び出して、カーネルが動作中かどうかを判別する。

この関数は、テンプレート引数に*false* を指定すると常に*false* を返すため、必要に応じてカーネルの 動作状態判別を省略する場合に使用する。

template<class Body> void toppers::task_entry (VP_INT *exinf*) [inline]

参照:

task_body

テンプレート引数*Body* として指定したボディクラスを本体とするタスクを定義する。 この関数を*T_CTSK* 構造体の*task* フィールドに設定することができる。

注意:

CRE_TSK （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

template<class Body> void toppers::taskexception_entry (TEXPTN *texptn*, VP_INT *exinf*) [inline]

参照:

taskexception_body

テンプレート引数*Body* として指定したボディクラスを本体とするタスク例外処理ルーチンを定義する。 この関数を*T_DTEX* 構造体の*texrtn* フィールドに設定することができる。

注意:

DEF_TEX （静的 API ）に対して使用する場合は、Cリンケージの関数で一旦ラッピングすること。

メモリアロケータ

インスタンスの生成と削除に関するカスタマイズ機能

構成

- class **allocator**
  メモリプールを使用したSTL互換のアロケータクラス

- class **fixed_new_pool**
  グローバルな new および delete 演算子による固定長メモリプール

- class **fixed_simple_pool**
  排他制御のない固定長メモリプール

- class **fixed_sized_allocator**
  固定サイズのアロケータを定義するためのヘルパークラス

- class **fixed_sized_new**
  指定メモリプールを用いた new および delete 演算子

- class **variable_new_pool**
  グローバルな new および delete 演算子による可変長メモリプール

- class **variable_simple_pool**
  排他制御のない可変長メモリプール

- class **variable_sized_new**
  指定メモリプールを用いた new および delete 演算子

解説

STL用のアロケータ:

≠ｱコンテナは、テンプレート引数としてアロケータを指定するようになっているため、それに適合する 任意のメモリプールを指定できるアロケータを提供する。

new および delete 演算子のカスタマイズ:

任意のメモリプールを指定できる new および delete 演算子を定義した既定クラスを提供する。

プログラム診断

エラー処理、整合性診断、および実行時トレース機能

構成

- class **error_handler_diagnostics**
  エラーハンドラ関数を呼び出す診断クラス

- struct **has_obj**
  カーネルが指定したカーネルオブジェクトをサポートしているかどうかを判別する

- struct **has_svc**
  カーネルが指定したサービスコールをサポートしているかどうかを判別する

- class **null_diagnostics**
  特に何も行わない診断クラス

- class **null_tracer**
  何も出力しないデータトレースクラス

- class **polymorphic_diagnostics**
  各処理を仮想関数として提供する診断クラス

- class **syslog_tracer**
  syslogを用いたデータトレースクラス

マクロ定義

- #define **TOPPERS_ASSERT**(expr) ((void)((expr) ? 0 : ::toppers::detail::assertion_failed(#expr, __FILE__, __LINE__)))
  実行時アサーション

- #define **TOPPERS_STATIC_ASSERT**(expr) typedef int TOPPERS_JOIN(toppers_diagnostics_, __LINE__)[toppers::static_assertion<!!(expr)>::okay]
  静的アサーション

- #define **TOPPERS_TYPENAME**(type) toppers::type_name<__typeof__(type)>::get()

型名を表す文字列定数の取得

解説

プログラム診断モジュールには、本ライブラリの大多数のクラスの基底となる *Diagnostics* ポリシーの他、 動的および静的アサーション、実行時のデータトレース機能などが含まれている。これらの機能は本ライブラリを 実装する上で多用されているだけでなく、ユーザがアプリケーションを実装する上でも有用である。

マクロ定義の解説

#define TOPPERS_ASSERT(expr) ((void)((expr) ? 0 : ::toppers::detail::assertion_failed(#expr, __FILE__, __LINE__)))

引数:

expr 評価対象の式

標準ライブラリの*assert* と同じだが、同一の翻訳単位内で再定義することはできない。

＜Jーネル、またはNEWLIB等の*assert* をサポートするライブラリを使用する場合は、単に *assert* に置換される。 *assert* がサポートされない環境の場合、*toppers::detail::assertion_failed* 関数を環境に合わせて実装する必要がある。

```
   int toppers::detail::assertion_failed(const char* expr, const char* file, int line)
throw();
```

引数の*expr* , *file* , *line* は、それぞれ評価式文字列、ファイル名文字列、行番号である。返却値は何でもよい。

注意:

標準ライブラリが*assert* をサポートしている場合は、toppers/assert.hpp より先にインクルードする必要がある。

#define TOPPERS_STATIC_ASSERT(expr) typedef int TOPPERS_JOIN(toppers_diagnostics_, __LINE__)[toppers::static_assertion<!!(expr)>::okay]

引数:

expr: 評価対象の定数式

e expr で指定した式が真に評価される場合は何も起こらないが、偽に評価された場合はコンパイルエラーを引き起こす。 コンパイルエラー発生時のメッセージはコンパイラによって異なるが、GCC 3.xの場合は以下のようなメッセージが出力される。

```
ファイル名:行番号: error: `okay' is not a member of type `toppers::static_assertion<false>'
```

同期機構

排他制御機能、および排他制御付きの演算機能を提供する。

構成

- class **lock**
  クリティカルセクションを生成するためのテンプレートクラス

- class **recursive_lock**
  ネスト可能なクリティカルセクションを生成するためのテンプレートクラス

関数

- template<class LockObj, typename T> T & **toppers::sync_increment** (T &x)
  同期指定付きのインクリメント

- template<class LockObj, typename T> T & **toppers::sync_decrement** (T &x)
  同期指定付きのデクリメント

- template<class LockObj, typename T> void **toppers::sync_swap** (T &x, T &y)
  同期指定付きのスワップ（値の入れ替え）

解説

μITRON 4.0仕様には、クリティカルセクションを作るための機能がいくつかある。 セマフォやミューテックス、CPUロックやディスパッチ禁止等である。 そうした機能はクリティカルセクションの最初と最後でサービスコールを呼出す必要がある。 しかし、C++ では C に比べて（多くの場合、例外が原因となって）実

行パスが非常に複雑であり、 クリティカルセクションから抜けるさいに確実にサービスコールを呼出すことは困難である。

通常、C++ では特定のスコープから抜ける場合に確実に処理を行わせる方法として、 デストラクタを使用する。 同期機構モジュールでは、インスタンスの生存期間がそのままクリティカルセクションとなるような、 排他制御のためのクラスを提供する。

例によって、クリティカルセクションを作るための機構は、テンプレート引数によって選択することができる。

関数の解説

template<class LockObj, typename T> T& sync_decrement (T & *x*) [inline]

参照:

sync_increment sync_swap

引数:

x: デクリメントするオブジェクト

返却値:

x を返す。

テンプレート引数*LockObj* で指定した同期オブジェクト（*semaphore* 、 *mutex* 、 *cpulock* 等）によって生成されたクリティカルセクションの中で、*x* を -- 演算子によってデクリメントする。*LockObj* の詳細については**lock** を参照のこと。

意見:

-- 演算子を多重定義することで、単純なデクリメント以外の動作を実現できる。

注意:

この関数はカーネル非動作状態では使用できない。

template<class LockObj, typename T> T& sync_increment (T & *x*) [inline]

参照:

sync_decrement sync_swap

引数:

x: インクリメントするオブジェクト

返却値:

x を返す。

テンプレート引数*LockObj* で指定した同期オブジェクト（*semaphore* 、 *mutex* 、 *cpulock* 等）によって生成されたクリティカルセクションの中で、*x* を ++ 演算子によってインクリメントする。*LockObj* の詳細については**lock** を参照のこと。

意見:

++ 演算子を多重定義することで、単純なインクリメント以外の動作を実現できる。

注意:

この関数はカーネル非動作状態では使用できない。

template<class LockObj, typename T> void sync_swap (T & *x*, T & *y*) [inline]

参照:

sync_increment sync_decrement

引数:

x: スワップするする一方のオブジェクト

y: スワップするするもう一方のオブジェクト

テンプレート引数*LockObj* で指定した同期オブジェクト（*semaphore* 、 *mutex* 、 *cpulock* 等）によって生成されたクリティカルセクションの中で、*x* と *y* の値を入れ替える。*LockObj* の詳細については**lock** を参照のこと。

意見:

コピーコンストラクタと代入演算子を用いて実装しているため、 それらの定義次第では、単純な値の入れ替え以外の動作を実現できる。

注意:

この関数はカーネル非動作状態では使用できない。

TOPPERS C++ Wrapper Library 名前空間の解説

名前空間 toppers の解説

ライブラリが使用するトップレベルの名前空間

構成

- class **alarmhandler**
  アラームハンドラ機能をカプセル化したテンプレートクラス

- class **alarmhandler_body**
  アラームハンドラの本体動作を定義する基底クラス

- class **allocator**
  メモリプールを使用したSTL互換のアロケータクラス

- struct **allocator::rebind**
  異なる型のオブジェクトを割り付け可能にするためのテンプレートクラス

- class **cpuexception**
  CPU例外処理の機能をカプセル化したテンプレートクラス.

- class **cpuexception_body**
  CPU例外処理ハンドラの本体動作を定義する基底クラス.

- class **cpulock**
  CPUロック状態の制御をカプセル化したテンプレートクラス.

- class **cyclichandler**
  周期ハンドラの機能をカプセル化したテンプレートクラス

- class **cyclichandler_body**
  周期ハンドラの本体動作を定義する基底クラス

- class **dataqueue**
  データキューの機能をカプセル化したテンプレートクラス

- class **dispatcher**
  ディスパッチャの制御をカプセル化したテンプレートクラス

- struct **dispatcher::pending**
  ディスパッチ保留状態参照クラス

- class **error_handler_diagnostics**
  エラーハンドラ関数を呼び出す診断クラス

- class **eventflag**
  イベントフラグの機能をカプセル化したテンプレートクラス

- class **fixed_new_pool**
  グローバルな new および delete 演算子による固定長メモリプール

- class **fixed_simple_pool**
  排他制御のない固定長メモリプール

- class **fixed_sized_allocator**
  固定サイズのアロケータを定義するためのヘルパークラス

- class **fixed_sized_new**
  指定メモリプールを用いた new および delete 演算子

- class **fmempool**
  固定長メモリプールの機能をカプセル化したテンプレートクラス

- class **fncode_stub**
  機能コードを用いたサービスコールスタブの定義

- struct **has_obj**
  カーネルが指定したカーネルオブジェクトをサポートしているかどうかを判別する

- struct **has_svc**
  カーネルが指定したサービスコールをサポートしているかどうかを判別する

- class **interrupt**
  割り込みを管理・制御するテンプレートクラス

- class **interrupthandler**

割込みハンドラを管理・制御するテンプレートクラス

- class **interrupthandler_body**
  割込みハンドラの本体動作を定義する基底クラス

- class **isr**
  割込みサービスルーチンの機能をカプセル化したテンプレートクラス

- class **isr_body**
  割込みサービスルーチンの本体動作を定義する基底クラス

- class **kernelstatus**
  カーネル状態に関する操作をカプセル化したテンプレートクラス

- struct **kernelstatus::context**
  実行中コンテキスト参照クラス

- class **lock**
  クリティカルセクションを生成するためのテンプレートクラス

- class **mailbox**
  メールボックスの機能をカプセル化したテンプレートクラス

- class **messagebuffer**
  メッセージバッファの機能をカプセル化したテンプレートクラス

- class **mutex**
  ミューテックスの機能をカプセル化したテンプレートクラス

- class **non_task_context_body**
  非タスクコンテキストのハンドラ本体定義用基底クラス

- class **null_diagnostics**
  特に何も行わない診断クラス

- class **null_tracer**
  何も出力しないデータトレースクラス

- class **object_controller**
  カーネルオブジェクトの生成・削除・状態参照を制御するクラス

- class **object_id**
  カーネルオブジェクトのID番号を管理するクラス

- class **outline_stub**
  外部関数を用いたサービスコールスタブの定義

- class **overrunhandler**
  オーバーランハンドラの機能をカプセル化したテンプレートクラス

- class **overrunhandler_body**
  オーバーランハンドラの本体動作を定義する基底クラス

- class **polymorphic_diagnostics**
  各処理を仮想関数として提供する診断クラス

- class **raw_stub**
  最も単純なサービスコールスタブ

- class **readyqueue**
  レディキューの操作をカプセル化したテンプレートクラス

- class **recursive_lock**
  ネスト可能なクリティカルセクションを生成するためのテンプレートクラス

- class **rendezvous**
  ランデブ番号をカプセル化したテンプレートクラス

- class **rendezvousport**
  ランデブポートの機能をカプセル化したテンプレートクラス

- class **semaphore**
  セマフォの機能をカプセル化したテンプレートクラス

- class **syslog_tracer**
  syslogを用いたデータトレースクラス

- class **task**
  タスク機能をカプセル化したテンプレートクラス

- class **task_body**
  タスク本体のデータおよび動作を定義する基底クラス

- class **task_body::terminator**
  自タスクを終了させる場合にスローする例外クラス

- class **task_context_body**
  タスクコンテキストの本体定義用基底クラス

- class **taskexception**
  タスク例外処理機能をカプセル化したテンプレートクラス

- class **taskexception_body**
  タスク例外処理ルーチンの本体動作を定義する基底クラス

- class **variable_new_pool**
  グローバルな new および delete 演算子による可変長メモリプール

- class **variable_simple_pool**
  排他制御のない可変長メモリプール

- class **variable_sized_new**
  指定メモリプールを用いた new および delete 演算子

- class **vmempool**
  可変長メモリプールの機能をカプセル化したテンプレートクラス

列挙体

- enum **function_code_type**
  機能コード

- enum **object_type**
  カーネルオブジェクトの種別

- enum **task_context_tag** { **task_context** = 1 }
  タスクコンテキスト用のサービスコールを要求するためのタグ

- enum **non_task_context_tag** { **non_task_context** = 2 }

非タスクコンテキスト用のサービスコールを要求するためのタグ

- enum **context_independent_tag** { **context_independent** = 0 }
  コンテキストに応じたサービスコールを要求するためのタグ

- enum **forever_tag** { **forever** = TMO_FEVR }
  永久待ち用のサービスコールを要求するためのタグ

- enum **polling_tag** { **polling** = TMO_POL }
  ポーリング用のサービスコールを要求するためのタグ

- enum **normal_tag** { **normal** = 0 }
  通常用（強制実行ではない）のサービスコールを要求するためのタグ

- enum **forced_tag** { **forced** = -3 }
  強制実行用のサービスコールを要求するためのタグ

関数

- template<class Body> void **alarmhandler_entry** (VP_INT)
  アラームハンドラのテンプレート

- template<class Body> void **cpuexception_entry** (VP p_exinf)
  CPU例外ハンドラのテンプレート.

- template<class Body> void **cyclichandler_entry** (VP_INT)
  周期ハンドラのテンプレート

- template<class Body> void **interrupthandler_entry** ()
  割込みハンドラのテンプレート

- template<class Body> void **isr_entry** (VP_INT)
  割込みサービスルーチンのテンプレート

- template<bool Sense> bool **sense_kernel** ()
  カーネルが動作中かどうかを判別する。

- template<class Body> void **overrunhandler_entry** (ID, VP_INT)
  オーバーランハンドラのテンプレート

- template<class LockObj, typename T> T & **sync_increment** (T &x)
  同期指定付きのインクリメント

- template<class LockObj, typename T> T & **sync_decrement** (T &x)
  同期指定付きのデクリメント

- template<class LockObj, typename T> void **sync_swap** (T &x, T &y)
  同期指定付きのスワップ（値の入れ替え）

- template<class Body> void **task_entry** (VP_INT)
  タスクの起動番地として指定するためのテンプレート関数

- template<class Body> void **taskexception_entry** (TEXPTN texptn, VP_INT exinf)
  タスク例外処理ルーチンのテンプレート

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **endl** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  改行およびフラッシュ操作用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **boolalpha** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  論理文字列指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **unboolalpha** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  論理文字列指定解除用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **showbase** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  基数接頭辞付加指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **unshowbase** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  基数接頭辞付加指定解除用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **uppercase** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  大文字指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **lowercase** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)

小文字指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **dec** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  10進数指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **oct** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  8進数指定用マニピュレータ

- template<std::size_t Count, std::size_t BufferSize, class Diagnostics> **syslog_tracer**< Count, BufferSize, Diagnostics > & **hex** (**syslog_tracer**< Count, BufferSize, Diagnostics > &tr)
  16進数指定用マニピュレータ

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, Tracer &(*manip)(Tracer &))
  引数なしマニピュレータの指定

- detail::setfill_helper **setfill** (char c)
  充填文字設定用マニピュレータ

- detail::setwidth_helper **setw** (int width)
  最小フィールド幅設定用マニピュレータ

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, bool x)
  論理型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, char x)
  文字のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const char *x)
  文字列のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, long x)
  long 型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, unsigned long x)
  unsigned long 型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, int x)
  int 型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, unsigned int x)
  unsigned int 型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const volatile void *x)
  ポインタ型のトレース出力

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_msg &x)
  T_MSG 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_msg_pri &x)
  T_MSG_PRI 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_ctsk &x)
  T_CTSK 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rtsk &x)
  T_RTSK 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rtst &x)
  T_RTST 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_dtex &x)
  T_DTEX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rtex &x)
  T_RTEX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_csem &x)
  T_CSEM 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rsem &x)
  T_RSEM 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cflg &x)
  T_CFLG 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rflg &x)
  T_RFLG 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cdtq &x)

T_CDTQ 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rdtq &x)
  T_RDTQ 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cmbx &x)
  T_CMBX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rmbx &x)
  T_RMBX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cmtx &x)
  T_CMTX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rmtx &x)
  T_RMTX 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cmbf &x)
  T_CMBF 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rmbf &x)
  T_RMBF 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cpor &x)
  T_CPOR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rpor &x)
  T_RPOR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rrdv &x)
  T_RRDV 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cmpf &x)
  T_CMPF 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rmpf &x)
  T_RMPF 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cmpl &x)
  T_CMPL 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rmpl &x)
  T_RMPL 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_ccyc &x)
  T_CCYC 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rcyc &x)
  T_RCYC 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_calm &x)
  T_CALM 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_ralm &x)
  T_RALM 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_dovr &x)
  T_DOVR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rovr &x)
  T_ROVR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rsys &x)
  T_RSYS 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_dinh &x)
  T_DINH 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_cisr &x)
  T_CISR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_risr &x)
  T_RISR 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_dsvc &x)
  T_DSVC 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_dexc &x)
  T_DEXC 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rcfg &x)
  T_RCFG 型のトレース出力.

- template<class Tracer> Tracer & **operator<<** (Tracer &tr, const t_rver &x)
  T_RVER 型のトレース出力.

解説

この名前空間は、ライブラリが提供する全機能を収めるトップレベルの名前空間である。

列挙体の解説

enum toppers::context_independent_tag

列挙定数:

context_independent コンテキスト非依存を指定する定数

enum toppers::forced_tag

列挙定数:

forced 強制実行用を指定する定数

enum toppers::forever_tag

列挙定数:

forever 永久待ちを指定する定数

enum toppers::non_task_context_tag

列挙定数:

non_task_context 非タスクコンテキストを指定する定数

enum toppers::normal_tag

列挙定数:

normal 通常用を指定する定数

enum toppers::polling_tag

列挙定数:

polling ポーリングを指定する定数

enum toppers::task_context_tag

列挙定数:

task_context タスクコンテキストを指定する定数

TOPPERS C++ Wrapper Library クラスの解説

クラス テンプレートtoppers::alarmhandler< Id, Diagnostics, Stub > の解説

アラームハンドラ機能をカプセル化したテンプレートクラス

```
#include <toppers/alarmhandler.hpp>
```

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

- typedef **alarmhandler_body**< Diagnostics, Stub > **body**
  ボディクラス

Public メンバ関数

コンストラクタ

- **alarmhandler** ()
  デフォルトコンストラクタ

- **alarmhandler** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **alarmhandler** (const **alarmhandler**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  アラームハンドラの生成

- ER **create** (ATR almatr, VP_INT exinf, void(*almhdr)(VP_INT))
  アラームハンドラの生成（個別のフィールド指定）

- template<class Body> ER **create** (ATR almatr=0, VP_INT exinf=0)
  アラームハンドラの生成（ボディ指定）

- ER **destroy** ()
  アラームハンドラの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  アラームハンドラの状態参照

起動と停止

- ER **start** (RELTIM almtim) const
  アラームハンドラの起動

- ER **stop** () const
  アラームハンドラの停止

フレンド

- ID **get_unsafe_id** (const **alarmhandler**< Id, Diagnostics, Stub > &**alarmhandler**)
  アラームハンドラID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::alarmhandler< Id, Diagnostics, Stub >

参照:

alarmhandler_body

alaramhandler_body テンプレートクラスはアラームハンドラに対する操作をカプセル化している。

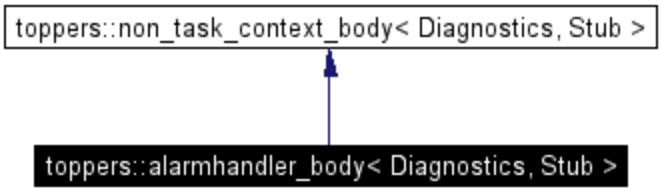
task 等と同様、ハンドラ内の処理は **alaramhandler_body** テンプレートクラスの派生クラスとして 記述する必要がある。

クラス テンプレートtoppers::alarmhandler_body< Diagnostics, Stub > の解説

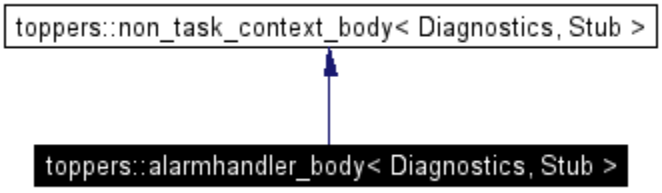
アラームハンドラの本体動作を定義する基底クラス

`#include <toppers/alarmhandler.hpp>`

toppers::alarmhandler_body< Diagnostics, Stub >に対する継承グラフ



toppers::alarmhandler_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **alarmhandler_body** (VP_INT exinf)
  コンストラクタ

- VP_INT **get_extended_info** () const
  拡張情報の取得

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **~alarmhandler_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::alarmhandler_body< Diagnostics, Stub >

参照:

alarmhandler alarmhandler_entry

アラームハンドラを作成する場合、**alarmhandler_body** テンプレートクラスから public 継承したクラスを定義し、その型を **alarmhandler_entry** テンプレート関数のテンプレート引数に渡す。

クラス テンプレートtoppers::allocator< T, Pool > の解説

メモリプールを使用したSTL互換のアロケータクラス

`#include <toppers/allocator.hpp>`

Public 型

- typedef std::size_t **size_type**
  オブジェクトのサイズ型

- typedef std::ptrdiff_t **different_type**
  イテレータ（この場合はポインタ）間の距離型

- typedef T * **pointer**
  ポインタ型

- typedef const T * **const_pointer**
  定数ポインタ型

- typedef T & **reference**
  参照型

- typedef const T & **const_reference**
  定数参照型

- typedef T **value_type**
  オブジェクトの型

Public メンバ関数

- **allocator** () throw ()
  コンストラクタ

- **allocator** (const Pool &pool) throw ()
  メモリプール指定のコンストラクタ

- **allocator** (const **allocator** &) throw ()
  コピーコンストラクタ

- template<typename T1> **allocator** (const **allocator**< T1, Pool > &) throw ()
  テンプレート引数違いの変換コンストラクタ

- **~allocator** () throw ()
  デストラクタ

- **pointer address** (**reference** x) const
  指定オブジェクトのアドレスを返す。

- **const_pointer address** (**const_reference** x) const
  指定オブジェクトのアドレスを返す。

- T * **allocate** (**size_type** n, const void *=0)
  メモリブロックの割付け

- void **deallocate** (**pointer** p, **size_type**)
  メモリブロックの解放

- **size_type max_size** () const throw ()
  割付可能な最大要素数

- void **construct** (**pointer** p, const T &value)
  メモリブロックをT　型のインスタンスとして初期化

- void **destroy** (**pointer** p)
  メモリブロックをT　型のインスタンスとして削除

解説

template<typename T, class Pool> class toppers::allocator< T, Pool >

　このテンプレートクラスは、STLが使用できる環境において、コンテナ等の アロケータとして使用するための要件を満たしている。 μITRONのメモリプールを使用して、*allocator*　をカスタマイズする際に使用 することができる。

構造体 テンプレートtoppers::allocator< T, Pool >::rebind< T1 > の解説

異なる型のオブジェクトを割り付け可能にするためのテンプレートクラス

```
#include <allocator.hpp>
```

template<typename T, class Pool>template<typename T1> struct toppers::allocator< T, Pool >::rebind< T1 >

クラス テンプレート**toppers::cpuexception< Diagnostics, Stub >** の解説

CPU例外処理の機能をカプセル化したテンプレートクラス.

`#include <toppers/cpuexception.hpp>`

Public 型

- typedef t_dexc **definition**
  定義情報パケット型

- typedef **cpuexception_body**< Diagnostics, Stub > **body**
  ボディクラス

Static Public メンバ関数

- ER **define** (EXCNO excno, const **definition** *pk_dobj)
  CPU例外処理ハンドラの定義.

- ER **define** (EXCNO excno, ATR excatr, void(*exchdr)(VP))
  CPU例外処理ハンドラの定義（個別のフィールド指定）.

- template<class Body> ER **define** (EXCNO excno, ATR excatr=0)
  CPU例外処理ハンドラの定義（ボディ指定）.

- ER **undefine** (EXCNO excno)
  CPU例外処理ハンドラの定義取り消し.

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::cpuexception< Diagnostics, Stub >

参照:

cpuexception_body

cpuexception_body テンプレートクラスはCPU例外処理に対する操作をカプセル化している。

task 等と同様、ハンドラ内の処理は **cpuexception_body** テンプレートクラスの派生クラスとして 記述する必要がある。

クラス テンプレート**toppers::cpuexception_body< Diagnostics, Stub >** の解説

CPU例外処理ハンドラの本体動作を定義する基底クラス.

`#include <toppers/cpuexception.hpp>`

Public メンバ関数

- **cpuexception_body** (VP_INT exinf)
  コンストラクタ

- VP_INT **get_extended_info** () const
  拡張情報の参照

Static Public メンバ関数

- bool **sense_context** (VP_INT exinf)
  CPU例外発生時のコンテキストの参照.

- bool **sense_cpulock** (VP_INT exinf)
  CPU例外発生時のCPUロック状態の参照.

- bool **sense_dispatch** (VP_INT exinf)
  CPU例外発生時のディスパッチ禁止状態の参照.

- bool **sense_dispatch_pending** (VP_INT exinf)
  CPU例外発生時のディスパッチペンディング状態の参照.

- bool **sense_taskexception** (VP_INT exinf)
  CPU例外発生時のタスク例外禁止状態の参照.

Protected メンバ関数

- **~cpuexception_body** ()
  デストラクタ

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::cpuexception_body< Diagnostics, Stub >

参照:

cpuexception cpuexception_entry

CPU例外処理ハンドラを作成する場合、cpuexception_body テンプレートクラスから public 継承したクラスを定義し、その型を cpuexception_entry テンプレート関数のテンプレート 引数に渡す。

クラス テンプレートtoppers::cpulock< Diagnostics, Stub > の解説

CPUロック状態の制御をカプセル化したテンプレートクラス.

`#include <toppers/kernelstatus.hpp>`

Static Public メンバ関数

- bool **sense** ()
  CPUロック状態の参照.

- ER **lock** (**task_context_tag**=task_context)
  CPUロック状態への以降.

- ER **lock** (**non_task_context_tag**)
  CPUロック状態への以降（非タスクコンテキスト）.

- ER **lock** (**context_independent_tag**)
  CPUロック状態への以降（コンテキスト非依存）.

- ER **unlock** (**task_context_tag**=task_context)
  CPUロック状態の解除.

- ER **unlock** (**non_task_context_tag**)
  CPUロック状態の解除（非タスクコンテキスト）.

- ER **unlock** (**context_independent_tag**)
  CPUロック状態の解除（コンテキスト非依存）.

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::cpulock< Diagnostics, Stub >

cpulock テンプレートクラスは、CPUロック状態の制御をカプセル化している。 各メンバ関数は実行コンテキストをパラメータ化することで汎用性を持たせている。

クラス テンプレートtoppers::cyclichandler< Id, Diagnostics, Stub > の解説

周期ハンドラの機能をカプセル化したテンプレートクラス

`#include <toppers/cyclichandler.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

- typedef **cyclichandler_body**< Diagnostics, Stub > **body**
  ボディクラス

Public メンバ関数

コンストラクタ

- **cyclichandler** ()
  デフォルトコンストラクタ

- **cyclichandler** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **cyclichandler** (const **cyclichandler**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  周期ハンドラの生成

- ER **create** (ATR cycatr, VP_INT exinf, void(*cychdr)(VP_INT), RELTIM cyctim=1, RELTIM cycphs=0)
  周期ハンドラの生成（個別のフィールド指定）

- template<class Body> ER **create** (ATR cycatr=TA_STA, VP_INT exinf=0, RELTIM cyctim=1, RELTIM cycphs=0)
  周期ハンドラの生成（ボディ指定）

- ER **destroy** ()
  周期ハンドラの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  周期ハンドラの状態参照

起動と停止

- ER **start** () const
  周期ハンドラの起動

- ER **stop** () const
  周期ハンドラの停止

フレンド

- ID **get_unsafe_id** (const **cyclichandler**< Id, Diagnostics, Stub > &**cyclichandler**)
  周期ハンドラID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::cyclichandler< Id, Diagnostics, Stub >

参照:

cyclichandler_body

cyclichandler_body テンプレートクラスは周期ハンドラに対する操作をカプセル化している。

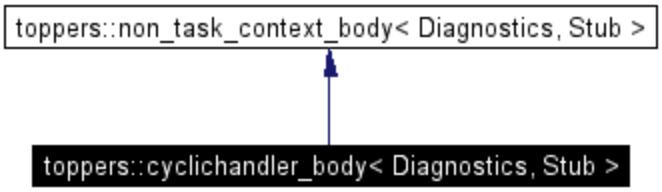
task 等と同様、ハンドラ内の処理は **cyclichandler_body** テンプレートクラスの派生クラスとして 記述する必要がある。

クラス テンプレートtoppers::cyclichandler_body< Diagnostics, Stub > の解説

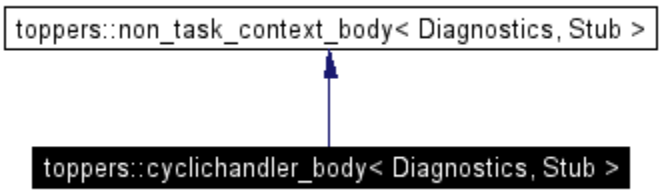
周期ハンドラの本体動作を定義する基底クラス

`#include <toppers/cyclichandler.hpp>`

toppers::cyclichandler_body< Diagnostics, Stub >に対する継承グラフ



toppers::cyclichandler_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **cyclichandler_body** (VP_INT exinf)
  コンストラクタ

- VP_INT **get_extended_info** () const
  拡張情報の取得

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **~cyclichandler_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::cyclichandler_body< Diagnostics, Stub >

参照:

cyclichandler cyclichandler_entry

周期ハンドラを作成する場合、cyclichandler_body テンプレートクラスから public 継承したクラスを定義し、その型を cyclichandler_entry テンプレート関数のテンプレート 引数に渡す。

クラス テンプレートtoppers::dataqueue< Id, Diagnostics, Stub > の解説

データキューの機能をカプセル化したテンプレートクラス

`#include <toppers/dataqueue.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **dataqueue** ()
  デフォルトコンストラクタ

- **dataqueue** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **dataqueue** (const **dataqueue**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  データキューの生成

- ER **create** (ATR dtqatr=TA_TFIFO, UINT dtqcnt=0, VP dtq=0)
  データキューの生成（個別のフィールド指定）

- ER **destroy** ()
  データキューの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  データキューの状態参照

データの送受信

- ER **send** (VP_INT data, **forever_tag**=forever, **task_context_tag**=task_context) const
  データキューへの送信

- ER **send** (VP_INT data, **polling_tag**, **task_context_tag**=task_context) const
  データキューへの送信（ポーリング）

- ER **send** (VP_INT data, TMO tmout, **task_context_tag**=task_context) const
  データキューへの送信（タイムアウト指定）

- ER **send** (VP_INT data, **forced_tag**, **task_context_tag**=task_context) const
  データキューへの強制送信

- ER **send** (VP_INT data, **polling_tag**, **non_task_context_tag**) const
  データキューへの送信（ポーリング/非タスクコンテキスト）

- ER **send** (VP_INT data, **forced_tag**, **non_task_context_tag**) const
  データキューへの強制送信（非タスクコンテキスト）

- ER **send** (VP_INT data, **polling_tag**, **context_independent_tag**) const
  データキューへの送信（ポーリング/コンテキスト非依存）

- ER **send** (VP_INT data, **forced_tag**, **context_independent_tag**) const
  データキューへの強制送信（コンテキスト非依存）

- ER **receive** (VP_INT *p_data, **forever_tag**=forever) const
  データキューからの受信

- ER **receive** (VP_INT *p_data, **polling_tag**) const
  データキューからの受信（ポーリング）

- ER **receive** (VP_INT *p_data, TMO tmout) const
  データキューからの受信（タイムアウト指定）

フレンド

- ID **get_unsafe_id** (const **dataqueue**< Id, Diagnostics, Stub > &**dataqueue**)
  データキューID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::dataqueue< Id, Diagnostics, Stub >

dataqueue テンプレートクラスは、データキューの機能をカプセル化し、 コンテキストや待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレートtoppers::dispatcher< Diagnostics, Stub > の解説

ディスパッチャの制御をカプセル化したテンプレートクラス

`#include <toppers/kernelstatus.hpp>`

Static Public メンバ関数

- bool **sense** ()
  ディスパッチ禁止状態の参照

- ER **disable** ()
  ディスパッチの禁止

- ER **enable** ()
  ディスパッチの許可

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::dispatcher< Diagnostics, Stub >

dispatcher テンプレートクラスは、ディスパッチ禁止状態の制御をカプセル化している。

構造体 toppers::dispatcher< Diagnostics, Stub >::pending の解説

ディスパッチ保留状態参照クラス

`#include <toppers/kernelstatus.hpp>`

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> struct toppers::dispatcher< Diagnostics, Stub >::pending

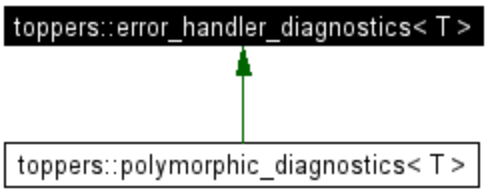
dispatcher テンプレートクラスは、ディスパッチ禁止状態の制御をカプセル化している。

クラス テンプレートtoppers::error_handler_diagnostics< T > の解説

エラーハンドラ関数を呼び出す診断クラス

`#include <toppers/diagnostics.hpp>`

toppers::error_handler_diagnostics< T >に対する継承グラフ



Public 型

- enum { **use_exception** = true }

Static Public メンバ関数

- ER_UINT **handle_error** (ER_UINT ercd)
  非静的メンバ関数内でのエラー処理

- ER_UINT **static_handle_error** (ER_UINT ercd)
  静的メンバ関数内でのエラー処理

- void **panic** ()
  致命的なエラーを検出した際のパニック処理

- handler_type **set_error_handler** (handler_type handler)
  エラーハンドラの設定

解説

template<class T = void> class toppers::error_handler_diagnostics< T >

error_handler_diagnostics テンプレートクラスは、静的メンバ関数*set_error_handler* によってエラー処理関数を登録し、エラー検出時に呼び出すためのものである。

エラー処理関数は共通のものを１つだけしか登録できないが、テンプレート引数*T* に適当な 型を指定することにより、異なるエラー処理関数を使い分けることも可能である。テンプレート引数*T* の型に特別な意味は無く、単にエラー処理関数を使い分けるためだけに 使用される。

```
// 以下のように error_handler_diagnostics を特化することが可能
class foo {};
typedef toppers::task<MAIN_TASK, toppers::error_handler_diagnostics<foo> > task;
```

列挙体の解説

template<class T = void> anonymous enum

列挙定数:

use_exception **error_handler_diagnostics** クラスが内部で例外を使用するかもしれないことを示す

クラス テンプレート**toppers::eventflag< Id, Diagnostics, Stub >** の解説

イベントフラグの機能をカプセル化したテンプレートクラス

`#include <toppers/eventflag.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **eventflag** ()
  デフォルトコンストラクタ

- **eventflag** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **eventflag** (const **eventflag**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  イベントフラグの生成

- ER **create** (ATR flgatr=TA_TFIFO|TA_WSGL, FLGPTN iflgptn=0)
  イベントフラグの生成（個別のフィールド指定）

- ER **destroy** ()
  イベントフラグの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  イベントフラグの状態参照

イベントフラグの操作

- ER **set** (FLGPTN setptn, **task_context_tag**=task_context) const
  イベントフラグのセット

- ER **set** (FLGPTN setptn, **non_task_context_tag**) const
  イベントフラグのセット（非タスクコンテキスト）

- ER **set** (FLGPTN setptn, **context_independent_tag**) const
  イベントフラグのセット（コンテキスト非依存）

- ER **clear** (FLGPTN clrptn) const
  イベントフラグのクリア

- ER **wait** (FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, **forever_tag**=forever) const
  イベントフラグ待ち

- ER **wait** (FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, **polling_tag**) const
  イベントフラグ待ち（ポーリング）

- ER **wait** (FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout) const
  イベントフラグ待ち（タイムアウト指定）

フレンド

- ID **get_unsafe_id** (const **eventflag**< Id, Diagnostics, Stub > &**eventflag**)
  イベントフラグID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::eventflag< Id, Diagnostics, Stub >

eventflag テンプレートクラスは、イベントフラグの機能をカプセル化し、 コンテキストや待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレート**toppers::fixed_new_pool< Size >** の解説

グローバルな new および delete 演算子による固定長メモリプール

`#include <toppers/allocator.hpp>`

Public メンバ関数

- ER **get** (VP *p_blk)
  固定長メモリブロックの獲得

- ER **get** (VP *p_blk, int sync)
  固定長メモリブロックの獲得（同期方法指定との互換用）

- ER **get** (UINT size, VP *p_blk)
  固定長メモリブロックの獲得（サイズ指定との互換用）

- ER **get** (UINT size, VP *p_blk, int sync)
  固定長メモリブロックの獲得（サイズ指定および同期方法指定との互換用）

- ER **release** (VP blk)
  固定長メモリブロックの解放

解説

template<std::size_t Size = 32> class toppers::fixed_new_pool< Size >

fixed_new_pool クテンプレートラスは、グローバルな new および delete 演算子を用いて、 固定長メモリプールの体裁を持たせたクラスである。 本来メモリプールを使用すべきところにこのクラスを指定することで、グローバルな new および delete 演算子を用いた動作に変更することができる。

クラス テンプレート**toppers::fixed_simple_pool< N, Size >** の解説

排他制御のない固定長メモリプール

`#include <toppers/allocator.hpp>`

Public メンバ関数

- **fixed_simple_pool** ()
  コンストラクタ

- ER **get** (VP *p_blk)
  固定長メモリブロックの獲得

- ER **get** (VP *p_blk, int sync)
  固定長メモリブロックの獲得（同期方法指定との互換用）

- ER **get** (UINT size, VP *p_blk)
  固定長メモリブロックの獲得（サイズ指定との互換用）

- ER **get** (UINT size, VP *p_blk, int sync)
  固定長メモリブロックの獲得（サイズ指定および同期方法指定との互換用）

- ER **release** (VP blk)
  固定長メモリブロックの解放

解説

template<std::size_t N = 32, std::size_t Size = 32> class toppers::fixed_simple_pool< N, Size >

fixed_simple_pool テンプレートクラスは、テンプレート引数*Size* で指定したサイズのメモリブロックを*N* 個割り付けられる単純な固定長メモリプールである。

fixed_simple_pool は、構造を単純化することで高速化を図っているため、排他制御は行っていない。

クラス テンプレートtoppers::fixed_sized_allocator< Size, Pool > の解説

固定サイズのアロケータを定義するためのヘルパークラス

`#include <toppers/allocator.hpp>`

Public メンバ関数

- template<typename Sync> void * **allocate** (Sync sync)
  メモリブロックの割り付け

- void * **allocate** ()
  メモリブロックの割り付け（永久待ち）

- void **deallocate** (void *p) throw ()
  メモリブロックの解放

- handler_type **set_new_handler** (handler_type handler)
  class_level_allocator のための set_new_handler 関数

解説

template<std::size_t Size, class Pool> class toppers::fixed_sized_allocator< Size, Pool >

fixed_sized_allocator テンプレートクラスは、固定長のメモリブロックを割り付けるための ヘルパークラスである。 このヘルパークラスは、主として***fixed_sized_new*** のテンプレート引数*Allocator* のために 使用される。

クラス テンプレートtoppers::fixed_sized_new< Host, Pool, Allocator > の解説

指定メモリプールを用いた new および delete 演算子

`#include <toppers/allocator.hpp>`

Public メンバ関数

- handler_type **set_new_handler** (handler_type handler)
  new演算子に失敗した際の処理関数の設定

Static Public メンバ関数

- void * **operator new** (std::size_t) throw (std::bad_alloc)
  new演算子

- void **operator delete** (void *p) throw ()
  delete演算子

- template<typename Sync> void * **operator new** (std::size_t, Sync sync) throw (std::bad_alloc)
  同期方法指定付きnew演算子

- template<typename Sync> void **operator delete** (void *p, Sync) throw ()
  同期方法指定付きnew演算子に対応するdelete演算子

- void * **operator new** (std::size_t size, const std::nothrow_t &nt) throw ()
  例外を投げないnew演算子

- void **operator delete** (void *p, const std::nothrow_t &) throw ()
  例外を投げないnew演算子に対応するdelete演算子

- template<typename Sync> void * **operator new** (std::size_t, const std::nothrow_t &, Sync sync) throw ()
  例外を投げない同期方法指定付きnew演算子

- template<typename Sync> void **operator delete** (void *p, const std::nothrow_t &, Sync) throw ()
  例外を投げない同期方法指定付きnew演算子に対応するdelete演算子

Protected メンバ関数

- **~fixed_sized_new** ()
  デストラクタ

解説

template<class Host, class Pool, template< std::size_t, class > class Allocator = fixed_sized_allocator> class toppers::fixed_sized_new< Host, Pool, Allocator >

fixed_sized_new は、テンプレート引数*Pool* で指定したメモリプールを用いた new および delete 演算子を定義している。

fixed_sized_new から public 継承することにより、そのクラスの new および delete 演算子をグローバルな new および delete 演算子と置き換えることができる。ただし、配列用の new および delete 演算子はデフォルトの ものが使用される。

テンプレート引数*Host* には、***variable_sized_new*** から派生させるクラスそのものを指定すること。 また、テンプレートテンプレート引数*Allocator* は、*Host* のオブジェクトサイズを指定して固定長メモリブロック を割り付けるためのヘルパーテンプレートであるが、ほとんどの場合はデフォルトのままでかまわない。

コンストラクタとデストラクタの解説

template<class Host, class Pool, template< std::size_t, class > class Allocator = fixed_sized_allocator> toppers::fixed_sized_new< Host, Pool, Allocator >::~fixed_sized_new () [inline, protected]

fixed_sized_newのポインタを用いて派生クラスがdeleteされることがないよう protected属性とする。

メンバ関数の解説

template<class Host, class Pool, template< std::size_t, class > class Allocator = fixed_sized_allocator> template<typename Sync> void* toppers::fixed_sized_new< Host, Pool, Allocator >::operator new (std::size_t, Sync *sync*) throw (std::bad_alloc) [inline, static]

引数:

sync 同期方法。forever、polling、およびタイムアウト時間を指定できる。

クラス テンプレートtoppers::fmempool< Id, Diagnostics, Stub > の解説

固定長メモリプールの機能をカプセル化したテンプレートクラス

`#include <toppers/fmempool.hpp>`

Public メンバ関数

コンストラクタ

- **fmempool** ()
  デフォルトコンストラクタ

- **fmempool** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **fmempool** (const **fmempool**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const creation *pk_cobj)
  固定長メモリプールの生成

- ER **create** (ATR mpfatr=TA_TFIFO, UINT blkcnt=32, UINT blksz=32, VP mpf=0)
  固定長メモリプールの生成（個別のフィールド指定）

- ER **destroy** ()
  固定長メモリプールの削除

状態参照

- ER **refer** (reference *pk_robj) const
  固定長メモリプールの状態参照

メモリブロックの獲得と解放

- ER **get** (VP *p_blk, **forever_tag**=forever) const
  固定長メモリブロックの獲得

- ER **get** (VP *p_blk, **polling_tag**) const
  固定長メモリブロックの獲得（ポーリング）

- ER **get** (VP *p_blk, TMO tmout) const
  固定長メモリブロックの獲得（タイムアウト指定）

- ER **get** (UINT, VP *p_blk, **forever_tag**=forever) const
  固定長メモリブロックの獲得

- ER **get** (UINT, VP *p_blk, **polling_tag**) const
  固定長メモリブロックの獲得（ポーリング）

- ER **get** (UINT, VP *p_blk, TMO tmout) const
  固定長メモリブロックの獲得（タイムアウト指定）

- ER **release** (VP blk) const
  固定長メモリブロックの解放

フレンド

- ID **get_unsafe_id** (const **fmempool**< Id, Diagnostics, Stub > &**fmempool**)
  固定長メモリプールID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::fmempool< Id, Diagnostics, Stub >

参照:

vmempool

fmempool テンプレートクラスは、固定長メモリプールをカプセル化するとともに、 他のメモリプールと交換可能なインタフェースを提供している。

クラス テンプレート**toppers::fncode_stub< Stub >** の解説

機能コードを用いたサービスコールスタブの定義

`#include <toppers/fncode_stub.hpp>`

Static Public メンバ関数

タスク管理機能

- ER **stub_cre_tsk** (ID tskid, const t_ctsk *pk_ctsk)
  タスクの生成

- ER **stub_del_tsk** (ID tskid)
  タスクの削除

- ER **stub_act_tsk** (ID tskid)
  タスクの起動

- ER_UINT **stub_can_act** (ID tskid)
  タスクの起動要求のキャンセル

- ER **stub_sta_tsk** (ID tskid, VP_INT stacd)
  タスクの起動（起動コード指定）

- void **stub_ext_tsk** ()
  タスクの終了

- void **stub_exd_tsk** ()
  タスクの終了と削除

- ER **stub_ter_tsk** (ID tskid)
  タスクの強制終了

- ER **stub_chg_pri** (ID tskid, PRI tskpri)
  タスク優先度の変更

- ER **stub_get_pri** (ID tskid, PRI *p_tskpri)
  タスク優先度の参照

- ER **stub_ref_tsk** (ID tskid, t_rtsk *pk_rtsk)
  タスクの状態参照

- ER **stub_ref_tst** (ID tskid, t_rtst *pk_rtst)
  タスクの状態参照（簡易版）

タスク付属同期機能

- ER **stub_slp_tsk** ()
  タスクの起床待ち

- ER **stub_tslp_tsk** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **stub_wup_tsk** (ID tskid)
  タスクの起床

- ER_UINT **stub_can_wup** (ID tskid)
  タスクの起床要求のキャンセル

- ER **stub_rel_wai** (ID tskid)
  待ち状態の強制解除

- ER **stub_sus_tsk** (ID tskid)
  強制待ち状態への以降

- ER **stub_rsm_tsk** (ID tskid)
  強制待ち状態からの再開

- ER **stub_frsm_tsk** (ID tskid)
  強制待ち状態からの強制再開

- ER **stub_dly_tsk** (RELTIM dlytim)
  タスクの遅延

タスク例外処理機能

- ER **stub_def_tex** (ID tskid, const t_dtex *pk_dtex)
  タスク例外処理ルーチンの定義

- ER **stub_ras_tex** (ID tskid, TEXPTN rasptn)

タスク例外処理の要求

- ER **stub_dis_tex** ()
  タスク例外処理の禁止

- ER **stub_ena_tex** ()
  タスク例外処理の許可

- bool **stub_sns_tex** ()
  タスク例外処理禁止状態の参照

- ER **stub_ref_tex** (ID tskid, t_rtex *pk_rtex)
  タスク例外処理の状態参照

セマフォ

- ER **stub_cre_sem** (ID semid, const t_csem *pk_csem)
  セマフォの生成

- ER **stub_del_sem** (ID semid)
  セマフォの削除

- ER **stub_sig_sem** (ID semid)
  セマフォ資源の返却

- ER **stub_wai_sem** (ID semid)
  セマフォ資源の獲得

- ER **stub_pol_sem** (ID semid)
  セマフォ資源の獲得（ポーリング）

- ER **stub_twai_sem** (ID semid, TMO tmout)
  セマフォ資源の獲得（タイムアウト指定）

- ER **stub_ref_sem** (ID semid, t_rsem *pk_rsem)
  セマフォの状態参照

イベントフラグ

- ER **stub_cre_flg** (ID flgid, const t_cflg *pk_cflg)
  イベントフラグの生成

- ER **stub_del_flg** (ID flgid)
  イベントフラグの削除

- ER **stub_set_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット

- ER **stub_clr_flg** (ID flgid, FLGPTN clrptn)
  イベントフラグのクリア

- ER **stub_wai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち

- ER **stub_pol_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち（ポーリング）

- ER **stub_twai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout)
  イベントフラグ待ち（タイムアウト指定）

- ER **stub_ref_flg** (ID flgid, t_rflg *pk_rflg)
  イベントフラグの状態参照

データキュー

- ER **stub_cre_dtq** (ID dtqid, const t_cdtq *pk_cdtq)
  データキューの生成

- ER **stub_del_dtq** (ID dtqid)
  データキューの削除

- ER **stub_snd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信

- ER **stub_psnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（ポーリング）

- ER **stub_tsnd_dtq** (ID dtqid, VP_INT data, TMO tmout)
  データキューへの送信（タイムアウト指定）

- ER **stub_fsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの強制送信

- ER **stub_rcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信

- ER **stub_prcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信（ポーリング）

- ER **stub_trcv_dtq** (ID dtqid, VP_INT *p_data, TMO tmout)
  データキューからの受信（タイムアウト指定）

- ER **stub_ref_dtq** (ID dtqid, t_rdtq *pk_rdtq)
  データキューの状態参照

メールボックス

- ER **stub_cre_mbx** (ID mbxid, const t_cmbx *pk_cmbx)
  メールボックスの生成

- ER **stub_del_mbx** (ID mbxid)
  メールボックスの削除

- ER **stub_snd_mbx** (ID mbxid, T_MSG *pk_msg)
  メールボックスへの送信

- ER **stub_rcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信

- ER **stub_prcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信（ポーリング）

- ER **stub_trcv_mbx** (ID mbxid, T_MSG **ppk_msg, TMO tmout)
  メールボックスからの受信（タイムアウト指定）

- ER **stub_ref_mbx** (ID mbxid, t_rmbx *pk_rmbx)
  メールボックスの状態参照

固定長メモリプール

- ER **stub_cre_mpf** (ID mpfid, const t_cmpf *pk_cmpf)
  固定長メモリプールの生成

- ER **stub_del_mpf** (ID mpfid)
  固定長メモリプールの削除

- ER **stub_rel_mpf** (ID mpfid, VP blk)
  固定長メモリブロックの解放

- ER **stub_get_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得

- ER **stub_pget_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpf** (ID mpfid, VP *p_blk, TMO tmout)
  固定長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpf** (ID mpfid, t_rmpf *pk_rmpf)
  固定長メモリプールの状態参照

時間管理機能

- ER **stub_set_tim** (const SYSTIM *p_systim)
  システム時刻の設定

- ER **stub_get_tim** (SYSTIM *p_systim)
  システム時刻の参照

周期ハンドラ

- ER **stub_cre_cyc** (ID cycid, const t_ccyc *pk_ccyc)
  周期ハンドラの生成

- ER **stub_del_cyc** (ID cycid)
  周期ハンドラの削除

- ER **stub_sta_cyc** (ID cycid)
  周期ハンドラの起動

- ER **stub_stp_cyc** (ID cycid)
  周期ハンドラの停止

- ER **stub_ref_cyc** (ID cycid, t_rcyc *pk_rcyc)

周期ハンドラの状態参照

システム状態管理機能

- ER **stub_rot_rdq** (PRI tskpri)
  タスク優先順位の回転

- ER **stub_get_tid** (ID *p_tskid)
  実行状態のタスクIDの参照

- ER **stub_loc_cpu** ()
  CPUロック状態への移行.

- ER **stub_unl_cpu** ()
  CPUロック状態の解除.

- ER **stub_dis_dsp** ()
  ディスパッチの禁止

- ER **stub_ena_dsp** ()
  ディスパッチの許可

- bool **stub_sns_ctx** ()
  コンテキストの参照

- bool **stub_sns_loc** ()
  CPUロック状態の参照.

- bool **stub_sns_dsp** ()
  ディスパッチ禁止状態の参照

- bool **stub_sns_dpn** ()
  ディスパッチ保留状態の参照

- ER **stub_ref_sys** (t_rsys *pk_rsys)
  システム状態の参照

割込み管理機能

- ER **stub_def_inh** (INHNO inhno, const t_dinh *pk_dinh)
  割込みハンドラの定義

- ER **stub_cre_isr** (ID isrid, const t_cisr *pk_cisr)
  割込みサービスルーチンの生成

- ER **stub_del_isr** (ID isrid)
  割込みサービスルーチンの削除

- ER **stub_ref_isr** (ID isrid, t_risr *pk_risr)
  割込みサービスルーチンの状態参照

- ER **stub_dis_int** (INTNO intno)
  割込みの禁止

- ER **stub_ena_int** (INTNO intno)
  割込みの許可

- ER **stub_chg_ims** (IMS ims)
  割込みマスクの変更

- ER **stub_get_ims** (IMS *P_ims)
  割込みマスクの参照

システム構成管理機能

- ER **stub_def_exc** (EXCNO excno, const t_dexc *pk_dexc)
  CPU例外ハンドラの定義.

- ER **stub_ref_cfg** (t_rcfg *pk_rcfg)
  コンフィギュレーション情報の参照

- ER **stub_ref_ver** (t_rver *pk_rver)
  バージョン情報の参照

非タスクコンテキスト用のサービスコール

- ER **stub_iact_tsk** (ID tskid)
  タスクの起動（非タスクコンテキスト）

- ER **stub_iwup_tsk** (ID tskid)
  タスクの起床（非タスクコンテキスト）

- ER **stub_irel_wai** (ID tskid)
  待ち状態の強制解除（非タスクコンテキスト）

- ER **stub_iras_tex** (ID tskid, TEXPTN rasptn)
  タスク例外処理の要求（非タスクコンテキスト）

- ER **stub_isig_sem** (ID semid)
  セマフォ資源の返却（非タスクコンテキスト）

- ER **stub_iset_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット（非タスクコンテキスト）

- ER **stub_ipsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（非タスクコンテキスト）

- ER **stub_ifsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの強制送信（非タスクコンテキスト）

- ER **stub_irot_rdq** (PRI tskpri)
  タスク優先順位の回転（非タスクコンテキスト）

- ER **stub_iget_tid** (ID *p_tskid)
  実行状態のタスクIDの参照（非タスクコンテキスト）

- ER **stub_iloc_cpu** ()
  CPUロック状態への移行（非タスクコンテキスト）.

- ER **stub_iunl_cpu** ()
  CPUロック状態の解除（非タスクコンテキスト）.

- ER **stub_isig_tim** ()
  タイムティックの供給

ミューテックス

- ER **stub_cre_mtx** (ID mtxid, const t_cmtx *pk_cmtx)
  ミューテックスの生成

- ER **stub_del_mtx** (ID mtxid)
  ミューテックスの削除

- ER **stub_unl_mtx** (ID mtxid)
  ミューテックスのロック解除

- ER **stub_loc_mtx** (ID mtxid)
  ミューテックスのロック

- ER **stub_ploc_mtx** (ID mtxid)
  ミューテックスのロック（ポーリング）

- ER **stub_tloc_mtx** (ID mtxid, TMO tmout)
  ミューテックスのロック（タイムアウト指定）

- ER **stub_ref_mtx** (ID mtxid, t_rmtx *pk_rmtx)
  ミューテックスの状態参照

メッセージバッファ

- ER **stub_cre_mbf** (ID mbfid, const t_cmbf *pk_cmbf)
  メッセージバッファの生成

- ER **stub_del_mbf** (ID mbfid)
  メッセージバッファの削除

- ER **stub_snd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信

- ER **stub_psnd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信（ポーリング）

- ER **stub_tsnd_mbf** (ID mbfid, const void *msg, UINT msgsz, TMO tmout)
  メッセージバッファへの送信（タイムアウト指定）

- ER_UINT **stub_rcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信

- ER_UINT **stub_prcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信（ポーリング）

- ER_UINT **stub_trcv_mbf** (ID mbfid, void *msg, TMO tmout)
  メッセージバッファからの受信（タイムアウト指定）

- ER **stub_ref_mbf** (ID mbfid, t_rmbf *pk_rmbf)
  メッセージバッファの状態参照

ランデブ

- ER **stub_cre_por** (ID porid, const t_cpor *pk_cpor)
  ランデブポートの生成

- ER **stub_del_por** (ID porid)
  ランデブポートの削除

- ER_UINT **stub_cal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz)
  ランデブポートの呼出し

- ER_UINT **stub_tcal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz, TMO tmout)
  ランデブポートの呼出し（タイムアウト指定）

- ER_UINT **stub_acp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付

- ER_UINT **stub_pacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付（ポーリング）

- ER_UINT **stub_tacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg, TMO tmout)
  ランデブポートの受付（タイムアウト指定）

- ER **stub_fwd_por** (ID porid, RDVPTN acpptn, RDVNO rdvno, const void *msg, UINT cmsgsz)
  ランデブの回送

- ER **stub_rpl_rdv** (RDVNO rdvno, const void *msg, UINT rmsgsz)
  ランデブの返答

- ER **stub_ref_por** (ID porid, t_rpor *pk_rpor)
  ランデブポートの状態参照

- ER **stub_ref_rdv** (RDVNO rdvid, t_rrdv *pk_rrdv)
  ランデブの状態参照

可変長メモリプール

- ER **stub_cre_mpl** (ID mplid, const t_cmpl *pk_cmpl)

可変長メモリプールの生成

- ER **stub_del_mpl** (ID mplid)
  可変長メモリプールの削除

- ER **stub_rel_mpl** (ID mplid, VP blk)
  可変長メモリブロックの解放

- ER **stub_get_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得

- ER **stub_pget_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpl** (ID mplid, UINT blksz, VP *p_blk, TMO tmout)
  可変長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpl** (ID mplid, t_rmpl *pk_rmpl)
  可変長メモリプールの状態参照

アラームハンドラ

- ER **stub_cre_alm** (ID almid, const t_calm *pk_calm)
  アラームハンドラの生成

- ER **stub_del_alm** (ID almid)
  アラームハンドラの削除

- ER **stub_sta_alm** (ID almid, RELTIM almtim)
  アラームハンドラの起動

- ER **stub_stp_alm** (ID almid)
  アラームハンドラの停止

- ER **stub_ref_alm** (ID almid, t_ralm *pk_ralm)
  アラームハンドラの状態参照

オーバーランハンドラ

- ER **stub_def_ovr** (const t_dovr *pk_dovr)
  オーバーランハンドラの定義

- ER **stub_sta_ovr** (ID tskid, OVRTIM ovrtim)
  オーバーランハンドラの起動

- ER **stub_stp_ovr** (ID tskid)
  オーバーランハンドラの停止

- ER **stub_ref_ovr** (ID tskid, t_rovr *pk_rovr)
  オーバーランハンドラの状態参照

ID番号自動生成

- ER_ID **stub_acre_tsk** (const t_ctsk *pk_ctsk)
  タスクの生成（ID番号自動生成）

- ER_ID **stub_acre_sem** (const t_csem *pk_csem)
  セマフォの生成（ID番号自動生成）

- ER_ID **stub_acre_flg** (const t_cflg *pk_cflg)
  イベントフラグの生成（ID番号自動生成）

- ER_ID **stub_acre_dtq** (const t_cdtq *pk_cdtq)
  データキューの生成（ID番号自動生成）

- ER_ID **stub_acre_mbx** (const t_cmbx *pk_cmbx)
  メールボックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mtx** (const t_cmtx *pk_cmtx)
  ミューテックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mbf** (const t_cmbf *pk_cmbf)
  メッセージバッファの生成（ID番号自動生成）

- ER_ID **stub_acre_por** (const t_cpor *pk_cpor)
  ランデブポートの生成（ID番号自動生成）

- ER_ID **stub_acre_mpf** (const t_cmpf *pk_cmpf)
  固定長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_mpl** (const t_cmpl *pk_cmpl)
  可変長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_cyc** (const t_ccyc *pk_ccyc)
  周期ハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_alm** (const t_calm *pk_calm)
  アラームハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_isr** (const t_cisr *pk_cisr)
  割込みサービスルーチンの生成（ID番号自動生成）

TOPPERSカーネルの独自仕様

これらはTOPPERSカーネル独自のサービスコールだが必須の要素であり、他のカーネルを使用する場合でも対応するサービスコールを実装する 必要がある。

- bool **stub_vxsns_ctx** (VP_INT exinf)
  CPU例外発生時のコンテキストの参照.

- bool **stub_vxsns_loc** (VP_INT exinf)
  CPU例外発生時のCPUロック状態の参照.

- bool **stub_vxsns_dsp** (VP_INT exinf)
  CPU例外発生時のディスパッチ禁止状態の参照.

- bool **stub_vxsns_dpn** (VP_INT exinf)
  CPU例外発生時のディスパッチ保留状態の参照.

- bool **stub_vxsns_tex** (VP_INT exinf)
  CPU例外発生時のタスク例外処理禁止状態の参照.

- bool **stub_vsns_ini** ()
  カーネル動作状態の参照

TOPPERSカーネルの独自拡張

- ER **stub_vxget_tim** (SYSUTIM *p_sysutim)
  性能評価用システム時刻の参照

サービスコール管理機能

- ER **stub_def_svc** (FN fncd, const t_dsvc *pk_dsvc)
  拡張サービスコールの定義

解説

template<class Stub = raw_stub<>> class toppers::fncode_stub< Stub >

注意:

cal_svc が標準サービスコールの呼び出しをサポートしない場合は正しく動作しない。

fncode_stub は*cal_svc* を介してサービスコールを呼び出す。 このサービスコールスタブを使用することで、機能コードを用いた呼び出しを通常の呼び出しと 全く同じ形式で行うことができる。

テンプレート引数の*Stub* は、*stub_cal_svc* メンバ関数が定義されていれば問題ないため、ユーザが *stub_cal_svc* だけを定義した独自のスタブを用意することで、***fncode_stub*** を使用することができる。

構造体 テンプレートtoppers::has_obj< ObjType > の解説

カーネルが指定したカーネルオブジェクトをサポートしているかどうかを判別する

`#include <toppers/kernel.hpp>`

Public 型

- enum { **value** = true }

解説

template<object_type ObjType> struct toppers::has_obj< ObjType >

注意:

カーネルオブジェクトがサポートされているかどうかは、主なサービスコールがあるかどうかで判別している。

テンプレート引数*ObjType* で指定したカーネルオブジェクトをカーネルがサポートしているかどうかをメンバ定数*value* として定義する。

ObjType には、 object_type 型の列挙定数である`*toppers::obj_カ* [ネルオブジェクト名 ' を指定すること。

列挙体の解説

template<object_type ObjType> anonymous enum

列挙定数:

value *ObjType* で指定したカーネルオブジェクトがサポートされていれば*true*

構造体 テンプレートtoppers::has_svc< Fncd > の解説

カーネルが指定したサービスコールをサポートしているかどうかを判別する

`#include <toppers/kernel.hpp>`

Public 型

- enum { **value** = true }

解説

template<function_code_type Fncd> struct toppers::has_svc< Fncd >

注意:

サービスコールがサポートされているかどうかは､プロトタイプかマクロ定義があるかどうかで判別しており、 実際の機能面には配慮していない。

テンプレート引数*Fncd* で指定したサービスコールをカーネルがサポートしているかどうかをメンバ定数*value* として定義する。

Fncd には、 function_code_type 型の列挙定数である`*toppers::tfn_*サ [ビスコール名 ' を指定すること。

列挙体の解説

template<function_code_type Fncd> anonymous enum

列挙定数:

value *Fncd* で指定したサービスコールがサポートされていれば*true*

クラス テンプレートtoppers::interrupt< Diagnostics, Stub > の解説

割り込みを管理・制御するテンプレートクラス

```
#include <toppers/interrupt.hpp>
```

Static Public メンバ関数

割り込みの禁止・許可

- ER **disable** (INTNO intno)
  割込みの禁止

- ER **enable** (INTNO intno)
  割込みの許可

割り込みマスク処理

- ER **change_mask** (IMS ims)
  割込みマスクの変更

- ER **get_mask** (IMS *p_ims)
  割込みマスクの参照

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::interrupt< Diagnostics, Stub >

参照:

interrupthandler interrupthandler_body interrupthandler_entry isr

interrupt テンプレートクラスは、割り込みの禁止・許可とマスク処理を カプセル化したものである。 このクラスを用いて操作した結果生じた副作用は、割り込みハンドラ および割り込みサービスルーチンの双方に影響を与える。

クラス テンプレート**toppers::interrupthandler< Diagnostics, Stub >** の解説

割込みハンドラを管理・制御するテンプレートクラス

`#include <toppers/interrupt.hpp>`

Public 型

- typedef t_dinh **definition**
  割込みハンドラ定義情報パケット型

Static Public メンバ関数

- ER **define** (INHNO inhno, const **definition** *pk_dobj)
  割込みハンドラの定義

- ER **define** (INHNO inhno, ATR inhatr, void(*inthdr)())
  割込みハンドラの定義（個別のフィールド指定）

- template<class Body> ER **define** (INHNO inhno, ATR inhatr=0)
  割込みハンドラの定義（ボディ指定）

- ER **undefine** (INHNO inhno)
  割込みハンドラの定義取り消し

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::interrupthandler< Diagnostics, Stub >

参照:

interrupt interrupthandler_body interrupthandler_entry isr

interrupthandler_body テンプレートクラスは割込みハンドラに対する操作をカプセル化している。

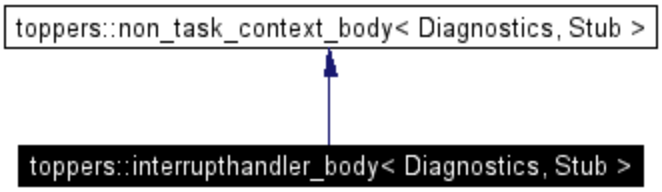
task 等と同様、ハンドラ内の処理は **interrupthandler_body** テンプレートクラスの派生クラスとして 記述する必要がある。

クラス テンプレートtoppers::interrupthandler_body< Diagnostics, Stub > の解説

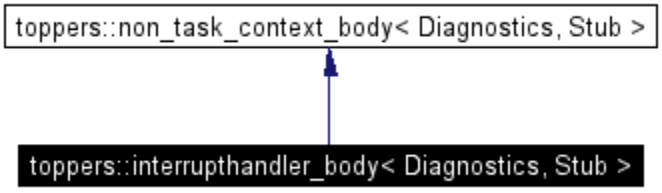
割込みハンドラの本体動作を定義する基底クラス

`#include <toppers/interrupt.hpp>`

toppers::interrupthandler_body< Diagnostics, Stub >に対する継承グラフ



toppers::interrupthandler_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::interrupthandler_body< Diagnostics, Stub >

参照:

interrupt interrupthandler_entry

割込みハンドラを作成する場合、interrupthandler_body テンプレートクラスから public 継承したクラスを定義し、その型を interrupthandler_entry テンプレート関数のテンプレート引数に渡す。

クラス テンプレート**toppers::isr< Id, Diagnostics, Stub >** の解説

割込みサービスルーチンの機能をカプセル化したテンプレートクラス

`#include <toppers/interrupt.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

- typedef **isr_body**< Diagnostics, Stub > **body**
  ボディクラス

Public メンバ関数

コンストラクタ

- **isr** ()
  デフォルトコンストラクタ

- **isr** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **isr** (const **isr**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  割込みサービスルーチンの生成

- ER **create** (ATR isratr, VP_INT exinf, INTNO intno, void(***isr**)(VP_INT))
  割込みサービスルーチンの生成（個別のフィールド指定）

- template<class Body> ER **create** (ATR isratr, VP_INT exinf, INTNO intno)
  割込みサービスルーチンの生成（ボディ指定）

- ER **destroy** ()
  割込みサービスルーチンの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  割込みサービスルーチンの状態参照

フレンド

- ID **get_unsafe_id** (const **isr**< Id, Diagnostics, Stub > &**isr**)
  割込みサービスルーチンID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::isr< Id, Diagnostics, Stub >

参照:

isr_body
interrupt interrupthandler

isr_body テンプレートクラスは割込みサービスルーチンに対する操作をカプセル化している。

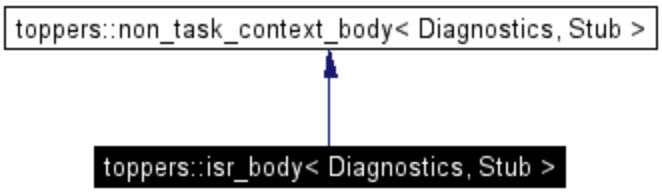
task 等と同様、ハンドラ内の処理は **isr_body** テンプレートクラスの派生クラスとして記述する必要がある。

クラス テンプレートtoppers::isr_body< Diagnostics, Stub > の解説

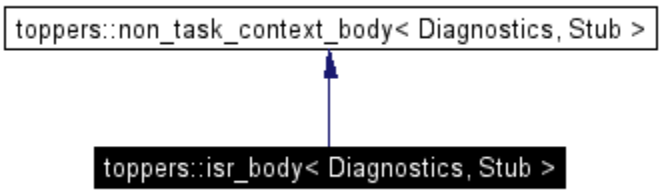
割込みサービスルーチンの本体動作を定義する基底クラス

`#include <toppers/interrupt.hpp>`

toppers::isr_body< Diagnostics, Stub >に対する継承グラフ



toppers::isr_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **isr_body** (VP_INT exinf)
  コンストラクタ

- VP_INT **get_extended_info** () const
  拡張情報の取得

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::isr_body< Diagnostics, Stub >

参照:

isr isr_entry

割込みサービスルーチンを作成する場合、isr_body テンプレートクラスから public 継承したクラスを定義し、その型を isr_entry テンプレート関数のテンプレート引数に渡す。

クラス テンプレートtoppers::kernelstatus< Diagnostics, Stub > の解説

カーネル状態に関する操作をカプセル化したテンプレートクラス

```
#include <toppers/kernelstatus.hpp>
```

Public 型

- typedef t_rsys **system_reference**
  システム情報参照パケット型

- typedef t_rver **version_reference**
  バージョン情報参照パケット型

- typedef t_rcfg **configuration_reference**
  コンフィギュレーション情報参照パケット型

- typedef **system_reference reference**
  システム情報参照パケット型

Static Public メンバ関数

- ER **refer** (**system_reference** *pk_robj)
  システム情報の参照

- ER **refer** (**version_reference** *pk_robj)
  カーネルのバージョン情報の参照

- ER **refer** (**configuration_reference** *pk_robj)
  コンフィギュレーション情報の参照

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::kernelstatus< Diagnostics, Stub >

kernelstatus テンプレートクラスは主としてカーネルの状態参照をカプセル化している。カーネルの状態にはシステム情報やバージョン情報など、いくつかあるが、それらを多重定義された 同名のメンバ関数で参照できるようになっている。

構造体 toppers::kernelstatus< Diagnostics, Stub >::context の解説

実行中コンテキスト参照クラス

`#include <toppers/kernelstatus.hpp>`

Static Public メンバ関数

- bool **sense** ()
  コンテキストの参照

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> struct toppers::kernelstatus< Diagnostics, Stub >::context

クラス テンプレートtoppers::lock< LockObj, SenseKernel > の解説

クリティカルセクションを生成するためのテンプレートクラス

`#include <toppers/sync.hpp>`

Public メンバ関数

- **lock** ()
  デフォルトコンストラクタ

- template<typename T> **lock** (T arg)
  パラメータ指定のコンストラクタ

- **~lock** () throw ()
  デストラクタ

解説

template<class LockObj, bool SenseKernel = false> class toppers::lock< LockObj, SenseKernel >

参照:

recursive_lock

例外安全なクリティカルセクションを、指定したロック機構を用いて生成する。 誤使用を避けるため、このクラスではポーリングやタイムアウト指定はサポートしない。

テンプレート引数*LockObj* には、*semaphore* や*mutex* 、または*cpulock* 等を指定することができる。*LockObj* に*void* を指定した場合には、 実際にはクリティカルセクションが生成されない。

テンプレート引数*Sensekernel* は、クリティカルセクションを生成するにあたり、 予めカーネルの動作状態を*vsns_ini* によって参照するかどうかを指定することができる。*SenseKernel* が*true* であればカーネル動作状態が参照される。

注意:

デフォルトでは*SenseKernel*　は*false*　になっているため、 カーネルが動作している必要がある。

クラス テンプレート**toppers::mailbox< Id, Diagnostics, Stub >** の解説

メールボックスの機能をカプセル化したテンプレートクラス

`#include <toppers/mailbox.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **mailbox** ()
  デフォルトコンストラクタ

- **mailbox** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **mailbox** (const **mailbox**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  メールボックスの生成

- ER **create** (ATR mbxatr=TA_TFIFO|TA_MFIFO, PRI maxmpri=TMAX_MPRI, VP mprihd=0)
  メールボックスの生成（個別のフィールド指定）

- ER **destroy** ()
  メールボックスの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  メールボックスの状態参照

メッセージの送受信

- ER **send** (t_msg *pk_msg) const
  メールボックスへの送信

- ER **receive** (t_msg **ppk_msg, **forever_tag**=forever) const
  メールボックスからの受信

- ER **receive** (t_msg **ppk_msg, **polling_tag**) const
  メールボックスからの受信（ポーリング）

- ER **receive** (t_msg **ppk_msg, TMO tmout) const
  メールボックスからの受信（タイムアウト指定）

フレンド

- ID **get_unsafe_id** (const **mailbox**< Id, Diagnostics, Stub > &**mailbox**)
  メールボックスID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::mailbox< Id, Diagnostics, Stub >

mailbox テンプレートクラスは、メールボックスの機能をカプセル化し、 待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレートtoppers::messagebuffer< Id, Diagnostics, Stub > の解説

メッセージバッファの機能をカプセル化したテンプレートクラス

`#include <toppers/messagebuffer.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **messagebuffer** ()
  デフォルトコンストラクタ

- **messagebuffer** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **messagebuffer** (const **messagebuffer**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  メッセージバッファの生成

- ER **create** (ATR mbfatr=TA_TFIFO, UINT maxmsz=256, SIZE mbfsz=0, VP mbf=0)
  メッセージバッファの生成（個別のフィールド指定）

- ER **destroy** ()
  メッセージバッファの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  メッセージバッファの状態参照

メッセージの送受信

- ER **send** (const void *msg, UINT msgsz, **forever_tag**=forever) const
  メッセージバッファへの送信

- ER **send** (const void *msg, UINT msgsz, **polling_tag**) const
  メッセージバッファへの送信（ポーリング）

- ER **send** (const void *msg, UINT msgsz, TMO tmout) const
  メッセージバッファへの送信（タイムアウト指定）

- ER_UINT **receive** (void *msg, **forever_tag**=forever) const
  メッセージバッファからの受信

- ER_UINT **receive** (void *msg, **polling_tag**) const
  メッセージバッファからの受信（ポーリング）

- ER_UINT **receive** (void *msg, TMO tmout) const
  メッセージバッファからの受信（タイムアウト指定）

フレンド

- ID **get_unsafe_id** (const **messagebuffer**< Id, Diagnostics, Stub > &**messagebuffer**)
  メッセージバッフヴヤ号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::messagebuffer< Id, Diagnostics, Stub >

messagebuffer テンプレートクラスは、メッセージバッファの機能をカプセル化し、 待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレートtoppers::mutex< Id, Diagnostics, Stub > の解説

ミューテックスの機能をカプセル化したテンプレートクラス

`#include <toppers/mutex.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照パケット型

Public メンバ関数

コンストラクタ

- **mutex** ()
  デフォルトコンストラクタ

- **mutex** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **mutex** (const **mutex**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  ミューテックスの生成

- ER **create** (ATR mtxatr=TA_TFIFO, PRI ceilpri=1)
  ミューテックスの生成（個別のフィールド指定）

- ER **destroy** ()

ミューテックスの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  ミューテックスの状態参照

ロック制御

- ER **lock** (**forever_tag**=forever) const
  ミューテックスのロック

- ER **lock** (**polling_tag**) const
  ミューテックスのロック（ポーリング）

- ER **lock** (TMO tmout) const
  ミューテックスのロック（タイムアウト指定）

- ER **unlock** () const
  ミューテックスのロック解除

フレンド

- ID **get_unsafe_id** (const **mutex**< Id, Diagnostics, Stub > &**mutex**)
  ミューテックスID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::mutex< Id, Diagnostics, Stub >

参照:

lock recursive_lock

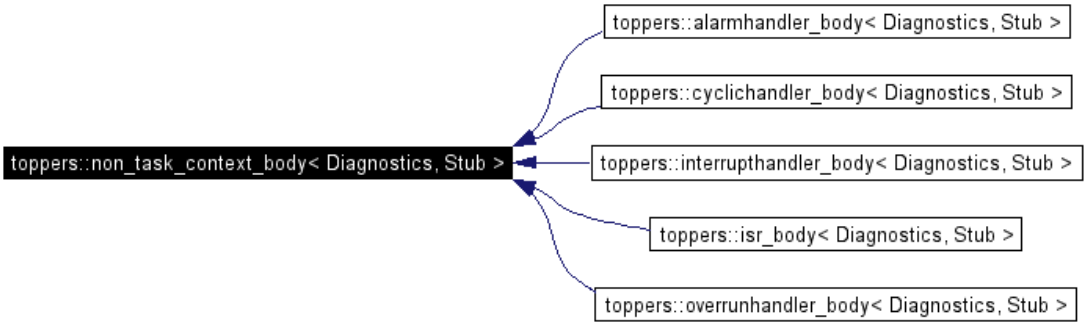
mutex テンプレートクラスは、ミューテックスの機能をカプセル化し、 コンテキストや待機方法をパラメータ化するためのインタフェースを提供する。 このクラスで提供されるインタフェースは、他の同期オブジェクトクラスと 交換可能になっている。

クラス テンプレートtoppers::non_task_context_body< Diagnostics, Stub > の解説

非タスクコンテキストのハンドラ本体定義用基底クラス

`#include <toppers/context.hpp>`

toppers::non_task_context_body< Diagnostics, Stub >に対する継承グラフ



Public メンバ関数

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**

サービスコールスタブ

Protected メンバ関数

- **non_task_context_body** ()
  コンストラクタ

- **~non_task_context_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics, class Stub> class toppers::non_task_context_body< Diagnostics, Stub >

non_task_context_body テンプレートクラスは、非タスクコンテキスト用ボディクラスの基底クラスとして使用される。

non_task_context_body テンプレートクラスから派生したクラスでは、*context_independent* 定数は *non_task_context* 定数と同じであり、シンタックスを汎用化するために使用することができる。

クラス **toppers::null_diagnostics** の解説

特に何も行わない診断クラス

`#include <toppers/diagnostics.hpp>`

Public 型

- enum { **use_exception** = false }

Static Public メンバ関数

- ER_UINT **handle_error** (ER_UINT ercd)
  非静的メンバ関数内でのエラー処理

- ER_UINT **static_handle_error** (ER_UINT ercd)
  静的メンバ関数内でのエラー処理

- void **panic** ()
  致命的なエラーを検出した際のパニック処理

解説

null_diagnostics クラスは具体的な診断を何も行わない。 このクラスは本ライブラリの大多数のテンプレートクラスが必要とする *Diagnostics* テンプレート引数のデフォルトポリシーとして使用される。

null_diagnostics のメンバ関数のうち*panic* だけは、例外的に作用し、 プログラムを異常終了させる。

列挙体の解説

anonymous enum

列挙定数:

use_exception **null_diagnostics** クラスが内部で例外を使用しないことを示す

クラス **toppers::null_tracer** の解説

何も出力しないデータトレースクラス

`#include <toppers/trace.hpp>`

Public メンバ関数

- template<typename T> **null_tracer** & **operator<<** (const T &obj)
  ダミー出力演算子

解説

null_tracer はデータトレースを抑止するためのクラスとして使用する。 他のデータトレースクラスを使用しているコードに対して、データとレースクラスの 型を***null_tracer***に変更するだけでデータトレースを抑止することができる。

クラス テンプレートtoppers::object_controller< Id, Object, Stub > の解説

カーネルオブジェクトの生成・削除・状態参照を制御するクラス

`#include <toppers/object.hpp>`

Static Public メンバ関数

- ER **create** (**object_id**< Id > &objid, const creation *pk_cobj)
  カーネルオブジェクトの生成

- ER **destroy** (**object_id**< Id > &objid)
  カーネルオブジェクトの削除

- ER **refer** (const **object_id**< Id > &objid, reference *pk_robj)
  カーネルオブジェクトの状態参照

解説

template<ID Id, object_type Object, class Stub> class toppers::object_controller< Id, Object, Stub >

object_controller テンプレートクラスはカーネルオブジェクトの生成、削除、 および状態参照をカプセル化している。

テンプレート引数*Id* が 0 以外の場合にはカーネルオブジェクトの生成には *cre_* 系サービスコールが使用される。また、*Id* が 0 の場合には*acre_* 系サービスコールが使用され、生成されたカーネルオブジェクトのID番号を自動的に 関連付けるようになっている。

クラス テンプレートtoppers::object_id< Id > の解説

カーネルオブジェクトのID番号を管理するクラス

`#include <toppers/object.hpp>`

Public メンバ関数

- **object_id** ()
  コンストラクタ

- **object_id** (ID id)
  ID番号指定のコンストラクタ.

- void **attach** (ID)
  カーネルオブジェクトID番号を関連付ける

- ID **detach** ()
  カーネルオブジェクトID番号を切り離す

- ID **id** () const
  関連付けられているカーネルオブジェクトID番号の参照

解説

template<ID Id> class toppers::object_id< Id >

object_id テンプレートクラスはカーネルオブジェクトのID番号を扱うためのクラスである。 テンプレート引数*Id* は、通常、コンフィギュレータが生成する kernel_id.h で定義される ID番号を指定するが、 0 を指定することでID番号をメンバ変数として保持することも可能である。 その場合には、実行時に関連付けられているID番号を設定・変更することができる。

注意:

ID番号が負の場合には対応していない。

クラス **toppers::outline_stub** の解説

外部関数を用いたサービスコールスタブの定義

`#include <toppers/outline_stub.hpp>`

Static Public メンバ関数

タスク管理機能

- ER **stub_cre_tsk** (ID tskid, const t_ctsk *pk_ctsk)
  タスクの生成

- ER **stub_del_tsk** (ID tskid)
  タスクの削除

- ER **stub_act_tsk** (ID tskid)
  タスクの起動

- ER_UINT **stub_can_act** (ID tskid)
  タスクの起動要求のキャンセル

- ER **stub_sta_tsk** (ID tskid, VP_INT stacd)
  タスクの起動（起動コード指定）

- void **stub_ext_tsk** ()
  タスクの終了

- void **stub_exd_tsk** ()
  タスクの終了と削除

- ER **stub_ter_tsk** (ID tskid)
  タスクの強制終了

- ER **stub_chg_pri** (ID tskid, PRI tskpri)
  タスク優先度の変更

- ER **stub_get_pri** (ID tskid, PRI *p_tskpri)
  タスク優先度の参照

- ER **stub_ref_tsk** (ID tskid, t_rtsk *pk_rtsk)
  タスクの状態参照

- ER **stub_ref_tst** (ID tskid, t_rtst *pk_rtst)
  タスクの状態参照（簡易版）

タスク付属同期機能

- ER **stub_slp_tsk** ()
  タスクの起床待ち

- ER **stub_tslp_tsk** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **stub_wup_tsk** (ID tskid)
  タスクの起床

- ER_UINT **stub_can_wup** (ID tskid)
  タスクの起床要求のキャンセル

- ER **stub_rel_wai** (ID tskid)
  待ち状態の強制解除

- ER **stub_sus_tsk** (ID tskid)
  強制待ち状態への以降

- ER **stub_rsm_tsk** (ID tskid)
  強制待ち状態からの再開

- ER **stub_frsm_tsk** (ID tskid)
  強制待ち状態からの強制再開

- ER **stub_dly_tsk** (RELTIM dlytim)
  タスクの遅延

タスク例外処理機能

- ER **stub_def_tex** (ID tskid, const t_dtex *pk_dtex)
  タスク例外処理ルーチンの定義

- ER **stub_ras_tex** (ID tskid, TEXPTN rasptn)
  タスク例外処理の要求

- ER **stub_dis_tex** ()
  タスク例外処理の禁止

- ER **stub_ena_tex** ()
  タスク例外処理の許可

- bool **stub_sns_tex** ()
  タスク例外処理禁止状態の参照

- ER **stub_ref_tex** (ID tskid, t_rtex *pk_rtex)
  タスク例外処理の状態参照

セマフォ

- ER **stub_cre_sem** (ID semid, const t_csem *pk_csem)
  セマフォの生成

- ER **stub_del_sem** (ID semid)
  セマフォの削除

- ER **stub_sig_sem** (ID semid)
  セマフォ資源の返却

- ER **stub_wai_sem** (ID semid)
  セマフォ資源の獲得

- ER **stub_pol_sem** (ID semid)
  セマフォ資源の獲得（ポーリング）

- ER **stub_twai_sem** (ID semid, TMO tmout)
  セマフォ資源の獲得（タイムアウト指定）

- ER **stub_ref_sem** (ID semid, t_rsem *pk_rsem)
  セマフォの状態参照

イベントフラグ

- ER **stub_cre_flg** (ID flgid, const t_cflg *pk_cflg)

イベントフラグの生成

- ER **stub_del_flg** (ID flgid)
  イベントフラグの削除

- ER **stub_set_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット

- ER **stub_clr_flg** (ID flgid, FLGPTN clrptn)
  イベントフラグのクリア

- ER **stub_wai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち

- ER **stub_pol_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち（ポーリング）

- ER **stub_twai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout)
  イベントフラグ待ち（タイムアウト指定）

- ER **stub_ref_flg** (ID flgid, t_rflg *pk_rflg)
  イベントフラグの状態参照

データキュー

- ER **stub_cre_dtq** (ID dtqid, const t_cdtq *pk_cdtq)
  データキューの生成

- ER **stub_del_dtq** (ID dtqid)
  データキューの削除

- ER **stub_snd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信

- ER **stub_psnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（ポーリング）

- ER **stub_tsnd_dtq** (ID dtqid, VP_INT data, TMO tmout)
  データキューへの送信（タイムアウト指定）

- ER **stub_fsnd_dtq** (ID dtqid, VP_INT data)

データキューへの強制送信

- ER **stub_rcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信

- ER **stub_prcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信（ポーリング）

- ER **stub_trcv_dtq** (ID dtqid, VP_INT *p_data, TMO tmout)
  データキューからの受信（タイムアウト指定）

- ER **stub_ref_dtq** (ID dtqid, t_rdtq *pk_rdtq)
  データキューの状態参照

メールボックス

- ER **stub_cre_mbx** (ID mbxid, const t_cmbx *pk_cmbx)
  メールボックスの生成

- ER **stub_del_mbx** (ID mbxid)
  メールボックスの削除

- ER **stub_snd_mbx** (ID mbxid, T_MSG *pk_msg)
  メールボックスへの送信

- ER **stub_rcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信

- ER **stub_prcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信（ポーリング）

- ER **stub_trcv_mbx** (ID mbxid, T_MSG **ppk_msg, TMO tmout)
  メールボックスからの受信（タイムアウト指定）

- ER **stub_ref_mbx** (ID mbxid, t_rmbx *pk_rmbx)
  メールボックスの状態参照

固定長メモリプール

- ER **stub_cre_mpf** (ID mpfid, const t_cmpf *pk_cmpf)
  固定長メモリプールの生成

- ER **stub_del_mpf** (ID mpfid)
  固定長メモリプールの削除

- ER **stub_rel_mpf** (ID mpfid, VP blk)
  固定長メモリブロックの解放

- ER **stub_get_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得

- ER **stub_pget_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpf** (ID mpfid, VP *p_blk, TMO tmout)
  固定長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpf** (ID mpfid, t_rmpf *pk_rmpf)
  固定長メモリプールの状態参照

時間管理機能

- ER **stub_set_tim** (const SYSTIM *p_systim)
  システム時刻の設定

- ER **stub_get_tim** (SYSTIM *p_systim)
  システム時刻の参照

周期ハンドラ

- ER **stub_cre_cyc** (ID cycid, const t_ccyc *pk_ccyc)
  周期ハンドラの生成

- ER **stub_del_cyc** (ID cycid)
  周期ハンドラの削除

- ER **stub_sta_cyc** (ID cycid)
  周期ハンドラの起動

- ER **stub_stp_cyc** (ID cycid)
  周期ハンドラの停止

- ER **stub_ref_cyc** (ID cycid, t_rcyc *pk_rcyc)
  周期ハンドラの状態参照

システム状態管理機能

- ER **stub_rot_rdq** (PRI tskpri)
  タスク優先順位の回転

- ER **stub_get_tid** (ID *p_tskid)
  実行状態のタスクIDの参照

- ER **stub_loc_cpu** ()
  CPUロック状態への移行.

- ER **stub_unl_cpu** ()
  CPUロック状態の解除.

- ER **stub_dis_dsp** ()
  ディスパッチの禁止

- ER **stub_ena_dsp** ()
  ディスパッチの許可

- bool **stub_sns_ctx** ()
  コンテキストの参照

- bool **stub_sns_loc** ()
  CPUロック状態の参照.

- bool **stub_sns_dsp** ()
  ディスパッチ禁止状態の参照

- bool **stub_sns_dpn** ()
  ディスパッチ保留状態の参照

- ER **stub_ref_sys** (t_rsys *pk_rsys)
  システム状態の参照

割込み管理機能

- ER **stub_def_inh** (INHNO inhno, const t_dinh *pk_dinh)

割込みハンドラの定義

- ER **stub_cre_isr** (ID isrid, const t_cisr *pk_cisr)
  割込みサービスルーチンの生成

- ER **stub_del_isr** (ID isrid)
  割込みサービスルーチンの削除

- ER **stub_ref_isr** (ID isrid, t_risr *pk_risr)
  割込みサービスルーチンの状態参照

- ER **stub_dis_int** (INTNO intno)
  割込みの禁止

- ER **stub_ena_int** (INTNO intno)
  割込みの許可

- ER **stub_chg_ims** (IMS ims)
  割込みマスクの変更

- ER **stub_get_ims** (IMS *P_ims)
  割込みマスクの参照

システム構成管理機能

- ER **stub_def_exc** (EXCNO excno, const t_dexc *pk_dexc)
  CPU例外ハンドラの定義.

- ER **stub_ref_cfg** (t_rcfg *pk_rcfg)
  コンフィギュレーション情報の参照

- ER **stub_ref_ver** (t_rver *pk_rver)
  バージョン情報の参照

非タスクコンテキスト用のサービスコール

- ER **stub_iact_tsk** (ID tskid)
  タスクの起動（非タスクコンテキスト）

- ER **stub_iwup_tsk** (ID tskid)
  タスクの起床（非タスクコンテキスト）

- ER **stub_irel_wai** (ID tskid)
  待ち状態の強制解除（非タスクコンテキスト）

- ER **stub_iras_tex** (ID tskid, TEXPTN rasptn)
  タスク例外処理の要求（非タスクコンテキスト）

- ER **stub_isig_sem** (ID semid)
  セマフォ資源の返却（非タスクコンテキスト）

- ER **stub_iset_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット（非タスクコンテキスト）

- ER **stub_ipsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（非タスクコンテキスト）

- ER **stub_ifsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの強制送信（非タスクコンテキスト）

- ER **stub_irot_rdq** (PRI tskpri)
  タスク優先順位の回転（非タスクコンテキスト）

- ER **stub_iget_tid** (ID *p_tskid)
  実行状態のタスクIDの参照（非タスクコンテキスト）

- ER **stub_iloc_cpu** ()
  CPUロック状態への移行（非タスクコンテキスト）.

- ER **stub_iunl_cpu** ()
  CPUロック状態の解除（非タスクコンテキスト）.

- ER **stub_isig_tim** ()
  タイムティックの供給

ミューテックス

- ER **stub_cre_mtx** (ID mtxid, const t_cmtx *pk_cmtx)
  ミューテックスの生成

- ER **stub_del_mtx** (ID mtxid)
  ミューテックスの削除

- ER **stub_unl_mtx** (ID mtxid)
  ミューテックスのロック解除

- ER **stub_loc_mtx** (ID mtxid)
  ミューテックスのロック

- ER **stub_ploc_mtx** (ID mtxid)
  ミューテックスのロック（ポーリング）

- ER **stub_tloc_mtx** (ID mtxid, TMO tmout)
  ミューテックスのロック（タイムアウト指定）

- ER **stub_ref_mtx** (ID mtxid, t_rmtx *pk_rmtx)
  ミューテックスの状態参照

メッセージバッファ

- ER **stub_cre_mbf** (ID mbfid, const t_cmbf *pk_cmbf)
  メッセージバッファの生成

- ER **stub_del_mbf** (ID mbfid)
  メッセージバッファの削除

- ER **stub_snd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信

- ER **stub_psnd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信（ポーリング）

- ER **stub_tsnd_mbf** (ID mbfid, const void *msg, UINT msgsz, TMO tmout)
  メッセージバッファへの送信（タイムアウト指定）

- ER_UINT **stub_rcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信

- ER_UINT **stub_prcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信（ポーリング）

- ER_UINT **stub_trcv_mbf** (ID mbfid, void *msg, TMO tmout)
  メッセージバッファからの受信（タイムアウト指定）

- ER **stub_ref_mbf** (ID mbfid, t_rmbf *pk_rmbf)
  メッセージバッファの状態参照

ランデブ

- ER **stub_cre_por** (ID porid, const t_cpor *pk_cpor)
  ランデブポートの生成

- ER **stub_del_por** (ID porid)
  ランデブポートの削除

- ER_UINT **stub_cal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz)
  ランデブポートの呼出し

- ER_UINT **stub_tcal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz, TMO tmout)
  ランデブポートの呼出し（タイムアウト指定）

- ER_UINT **stub_acp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付

- ER_UINT **stub_pacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付（ポーリング）

- ER_UINT **stub_tacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg, TMO tmout)
  ランデブポートの受付（タイムアウト指定）

- ER **stub_fwd_por** (ID porid, RDVPTN acpptn, RDVNO rdvno, const void *msg, UINT cmsgsz)
  ランデブの回送

- ER **stub_rpl_rdv** (RDVNO rdvno, const void *msg, UINT rmsgsz)
  ランデブの返答

- ER **stub_ref_por** (ID porid, t_rpor *pk_rpor)
  ランデブポートの状態参照

- ER **stub_ref_rdv** (RDVNO rdvid, t_rrdv *pk_rrdv)
  ランデブの状態参照

可変長メモリプール

- ER **stub_cre_mpl** (ID mplid, const t_cmpl *pk_cmpl)
  可変長メモリプールの生成

- ER **stub_del_mpl** (ID mplid)
  可変長メモリプールの削除

- ER **stub_rel_mpl** (ID mplid, VP blk)
  可変長メモリブロックの解放

- ER **stub_get_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得

- ER **stub_pget_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpl** (ID mplid, UINT blksz, VP *p_blk, TMO tmout)
  可変長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpl** (ID mplid, t_rmpl *pk_rmpl)
  可変長メモリプールの状態参照

アラームハンドラ

- ER **stub_cre_alm** (ID almid, const t_calm *pk_calm)
  アラームハンドラの生成

- ER **stub_del_alm** (ID almid)
  アラームハンドラの削除

- ER **stub_sta_alm** (ID almid, RELTIM almtim)
  アラームハンドラの起動

- ER **stub_stp_alm** (ID almid)
  アラームハンドラの停止

- ER **stub_ref_alm** (ID almid, t_ralm *pk_ralm)
  アラームハンドラの状態参照

オーバーランハンドラ

- ER **stub_def_ovr** (const t_dovr *pk_dovr)
  オーバーランハンドラの定義

- ER **stub_sta_ovr** (ID tskid, OVRTIM ovrtim)
  オーバーランハンドラの起動

- ER **stub_stp_ovr** (ID tskid)
  オーバーランハンドラの停止

- ER **stub_ref_ovr** (ID tskid, t_rovr *pk_rovr)
  オーバーランハンドラの状態参照

ID番号自動生成

- ER_ID **stub_acre_tsk** (const t_ctsk *pk_ctsk)
  タスクの生成（ID番号自動生成）

- ER_ID **stub_acre_sem** (const t_csem *pk_csem)
  セマフォの生成（ID番号自動生成）

- ER_ID **stub_acre_flg** (const t_cflg *pk_cflg)
  イベントフラグの生成（ID番号自動生成）

- ER_ID **stub_acre_dtq** (const t_cdtq *pk_cdtq)
  データキューの生成（ID番号自動生成）

- ER_ID **stub_acre_mbx** (const t_cmbx *pk_cmbx)
  メールボックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mtx** (const t_cmtx *pk_cmtx)
  ミューテックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mbf** (const t_cmbf *pk_cmbf)
  メッセージバッファの生成（ID番号自動生成）

- ER_ID **stub_acre_por** (const t_cpor *pk_cpor)
  ランデブポートの生成（ID番号自動生成）

- ER_ID **stub_acre_mpf** (const t_cmpf *pk_cmpf)
  固定長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_mpl** (const t_cmpl *pk_cmpl)
  可変長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_cyc** (const t_ccyc *pk_ccyc)
  周期ハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_alm** (const t_calm *pk_calm)
  アラームハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_isr** (const t_cisr *pk_cisr)
  割込みサービスルーチンの生成（ID番号自動生成）

TOPPERSカーネルの独自仕様

これらはTOPPERSカーネル独自のサービスコールだが必須の要素であり、他のカーネルを使用する場合でも対応するサービスコールを実装する 必要がある。

- bool **stub_vxsns_ctx** (VP_INT exinf)
  CPU例外発生時のコンテキストの参照.

- bool **stub_vxsns_loc** (VP_INT exinf)
  CPU例外発生時のCPUロック状態の参照.

- bool **stub_vxsns_dsp** (VP_INT exinf)
  CPU例外発生時のディスパッチ禁止状態の参照.

- bool **stub_vxsns_dpn** (VP_INT exinf)
  CPU例外発生時のディスパッチ保留状態の参照.

- bool **stub_vxsns_tex** (VP_INT exinf)
  CPU例外発生時のタスク例外処理禁止状態の参照.

- bool **stub_vsns_ini** ()
  カーネル動作状態の参照

TOPPERSカーネルの独自拡張

- ER **stub_vxget_tim** (SYSUTIM *p_sysutim)
  性能評価用システム時刻の参照

サービスコール管理機能

- ER **stub_def_svc** (FN fncd, const t_dsvc *pk_dsvc)
  拡張サービスコールの定義

- ER_UINT **stub_cal_svc** (FN fncd)
  拡張サービスコールの呼出し（パラメータなし）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1)
  拡張サービスコールの呼出し（パラメータ x 1）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2)
  拡張サービスコールの呼出し（パラメータ x 2）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3)
  拡張サービスコールの呼出し（パラメータ x 3）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4)
  拡張サービスコールの呼出し（パラメータ x 4）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4, VP_INT par5)
  拡張サービスコールの呼出し（パラメータ x 5）

解説

注意:

このサービスコールスタブを使用する場合には*libcxxwrap.a*　をリンクする必要がある。

覚え書き:

各メンバ関数へのポインタをテンプレート引数にできるように、形式的にはインライン関数にしている。

μITRONカーネルの多くは、C言語の使用を前提としているため、サービスコールのプロトタイプに例外指定が付いていない。 そのため、内部でサービスコールを呼び出す関数の多くは、サービスコールが何らかの例外を投げるものと仮定して コンパイルされるため、不必要なコードが展開される場合が少なくない。

outline_stub はそうしたカーネルでも、できる限り空間効率を向上させるためのサービスコールスタブである。また、そのようなカーネルでは、***outline_stub*** を使用することで、空間効率だけでなく、実行効率が向上することもあり得る。 実際に効率にどのような影響が出るかは、コンパイラの実装に強く依存するため、その都度評価する必要がある。

/JSPカーネル 1.4以降または TOPPERS/FI4カーネル 1.0以降、およびそれらをベースとしたカーネルを使用する限り、 ***outline_stub*** を使用する必要性はない。

クラス テンプレート**toppers::overrunhandler< Diagnostics, Stub >** の解説

オーバーランハンドラの機能をカプセル化したテンプレートクラス

`#include <toppers/overrunhandler.hpp>`

Public 型

- typedef t_dovr **definition**
  定義情報パケット型

- typedef t_rovr **reference**
  状態参照情報パケット型

- typedef **overrunhandler_body**< Diagnostics, Stub > **body**
  ボディクラス

Static Public メンバ関数

ハンドラの定義

- ER **define** (const **definition** *pk_cobj)
  オーバーランハンドラの定義

- ER **define** (ATR ovratr, void(*ovrhdr)(ID, VP_INT))
  オーバーランハンドラの定義（個別のフィールド指定）

- template<class Body> ER **define** (ATR ovratr=0)
  オーバーランハンドラの定義（ボディ指定）

- ER **undefine** ()
  オーバーランハンドラの定義取り消し

状態参照

- template<ID Id, class Diagnostics2, class Stub2> ER **refer** (const **task**< Id, Diagnostics2, Stub2 > &host, **reference** *pk_robj)
  オーバーランハンドラの状態参照

- ER **refer** (**reference** *pk_robj)
  自タスクに対するオーバーランハンドラの状態参照

起動と停止

- template<ID Id, class Diagnostics2, class Stub2> ER **start** (const **task**< Id, Diagnostics2, Stub2 > &host, OVRTIM ovrtim)
  オーバーランハンドラの起動

- template<ID Id, class Diagnostics2, class Stub2> ER **stop** (const **task**< Id, Diagnostics2, Stub2 > &host)
  オーバーランハンドラの停止

- ER **start** (OVRTIM ovrtim)
  自タスクに対するオーバーランハンドラの起動

- ER **stop** ()
  自タスクに対するオーバーランハンドラの停止

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::overrunhandler< Diagnostics, Stub >

テンプレートクラスはアラームハンドラに対する操作をカプセル化している。

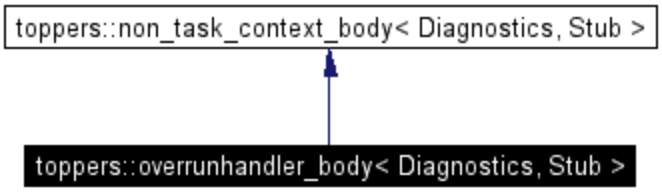
task 等と同様、ハンドラ内の処理は **overrunhandler_body** テンプレートクラスの派生クラスとして 記述する必要がある。

クラス テンプレートtoppers::overrunhandler_body< Diagnostics, Stub > の解説

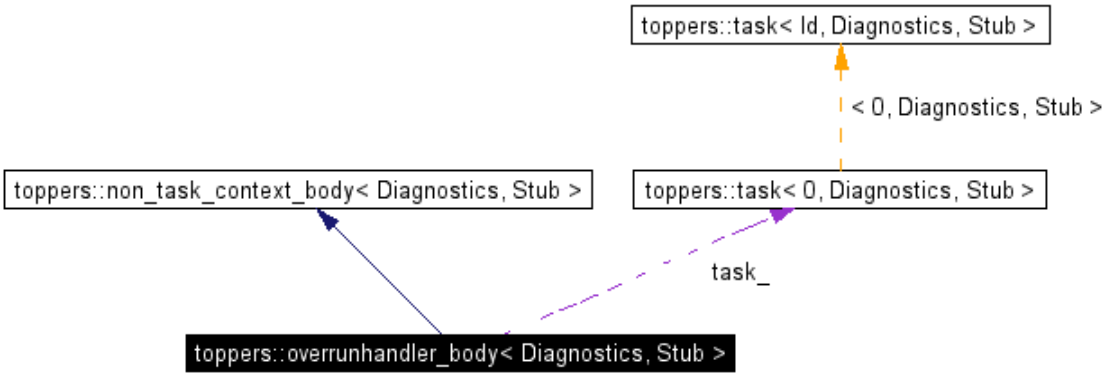
オーバーランハンドラの本体動作を定義する基底クラス

`#include <toppers/overrunhandler.hpp>`

toppers::overrunhandler_body< Diagnostics, Stub >に対する継承グラフ



toppers::overrunhandler_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **overrunhandler_body** (ID tskid, VP_INT exinf)
  コンストラクタ

- const **task**< 0, Diagnostics, Stub > & **get_task** () const
  タスク管理オブジェクトの参照

- VP_INT **get_extended_info** () const

拡張情報の取得

- void **run** ()
  ハンドラ本体動作の実行

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **non_task_context_tag context_independent** = toppers::non_task_context
  現在のコンテキスト（すなわち非タスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **~overrunhandler_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::overrunhandler_body< Diagnostics, Stub >

参照:

overrunhandler overrunhandler_entry

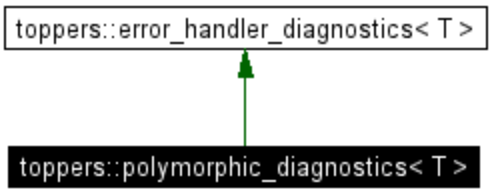
オーバーランハンドラを作成する場合、overrunhandler_body テンプレートクラスから public 継承したクラスを定義し、その型を overrunhandler_entry テンプレート関数のテンプレート 引数に渡す。

クラス テンプレートtoppers::polymorphic_diagnostics< T > の解説

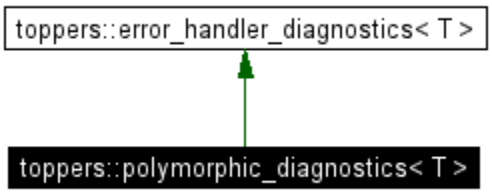
各処理を仮想関数として提供する診断クラス

`#include <toppers/diagnostics.hpp>`

toppers::polymorphic_diagnostics< T >に対する継承グラフ



toppers::polymorphic_diagnostics< T >のコラボレーション図



Public 型

- enum { **use_exception** = true }

Public メンバ関数

- virtual **~polymorphic_diagnostics** () throw ()
  仮想デストラクタ

- virtual ER_UINT **handle_error** (ER_UINT ercd) const
  非静的メンバ関数内でのエラー処理

- virtual void **panic** () const
  致命的なエラーを検出した際のパニック処理

Protected 型

- enum { **use_exception** = true }

Static Protected メンバ関数

- ER_UINT **handle_error** (ER_UINT ercd)
  非静的メンバ関数内でのエラー処理

- ER_UINT **static_handle_error** (ER_UINT ercd)
  静的メンバ関数内でのエラー処理

- void **panic** ()
  致命的なエラーを検出した際のパニック処理

- handler_type **set_error_handler** (handler_type handler)
  エラーハンドラの設定

解説

template<class T = void> class toppers::polymorphic_diagnostics< T >

polymorphic_diagnostics テンプレートクラスは***error_handler_diagnostics*** テンプレートクラスから protected 継承している。 デフォルトの動作は***error_handler_diagnostics*** と変わらないが、*handler_error* および*panic* メンバ関数が仮想関数として実装されている。また、デストラクタも 仮想関数になっているため、本ライブラリが提供するクラスから派生クラスを作る場合に 有効である。

列挙体の解説

template<class T = void> anonymous enum `[inherited]`

列挙定数:

use_exception **error_handler_diagnostics** クラスが内部で例外を使用するかもしれないことを示す

template<class T = void> anonymous enum

列挙定数:

use_exception **error_handler_diagnostics** クラスが内部で例外を使用するかもしれないことを示す

クラス テンプレート**toppers::raw_stub< Exist >** の解説

最も単純なサービスコールスタブ

`#include <toppers/raw_stub.hpp>`

Static Public メンバ関数

タスク管理機能

- ER **stub_cre_tsk** (ID tskid, const t_ctsk *pk_ctsk)
  タスクの生成
- ER **stub_del_tsk** (ID tskid)
  タスクの削除
- ER **stub_act_tsk** (ID tskid)
  タスクの起動
- ER_UINT **stub_can_act** (ID tskid)
  タスクの起動要求のキャンセル
- ER **stub_sta_tsk** (ID tskid, VP_INT stacd)
  タスクの起動（起動コード指定）
- void **stub_ext_tsk** ()
  タスクの終了
- void **stub_exd_tsk** ()
  タスクの終了と削除
- ER **stub_ter_tsk** (ID tskid)
  タスクの強制終了
- ER **stub_chg_pri** (ID tskid, PRI tskpri)
  タスク優先度の変更
- ER **stub_get_pri** (ID tskid, PRI *p_tskpri)
  タスク優先度の参照

- ER **stub_ref_tsk** (ID tskid, t_rtsk *pk_rtsk)
  タスクの状態参照

- ER **stub_ref_tst** (ID tskid, t_rtst *pk_rtst)
  タスクの状態参照（簡易版）

タスク付属同期機能

- ER **stub_slp_tsk** ()
  タスクの起床待ち

- ER **stub_tslp_tsk** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **stub_wup_tsk** (ID tskid)
  タスクの起床

- ER_UINT **stub_can_wup** (ID tskid)
  タスクの起床要求のキャンセル

- ER **stub_rel_wai** (ID tskid)
  待ち状態の強制解除

- ER **stub_sus_tsk** (ID tskid)
  強制待ち状態への以降

- ER **stub_rsm_tsk** (ID tskid)
  強制待ち状態からの再開

- ER **stub_frsm_tsk** (ID tskid)
  強制待ち状態からの強制再開

- ER **stub_dly_tsk** (RELTIM dlytim)
  タスクの遅延

タスク例外処理機能

- ER **stub_def_tex** (ID tskid, const t_dtex *pk_dtex)
  タスク例外処理ルーチンの定義

- ER **stub_ras_tex** (ID tskid, TEXPTN rasptn)

タスク例外処理の要求

- ER **stub_dis_tex** ()
  タスク例外処理の禁止

- ER **stub_ena_tex** ()
  タスク例外処理の許可

- bool **stub_sns_tex** ()
  タスク例外処理禁止状態の参照

- ER **stub_ref_tex** (ID tskid, t_rtex *pk_rtex)
  タスク例外処理の状態参照

セマフォ

- ER **stub_cre_sem** (ID semid, const t_csem *pk_csem)
  セマフォの生成

- ER **stub_del_sem** (ID semid)
  セマフォの削除

- ER **stub_sig_sem** (ID semid)
  セマフォ資源の返却

- ER **stub_wai_sem** (ID semid)
  セマフォ資源の獲得

- ER **stub_pol_sem** (ID semid)
  セマフォ資源の獲得（ポーリング）

- ER **stub_twai_sem** (ID semid, TMO tmout)
  セマフォ資源の獲得（タイムアウト指定）

- ER **stub_ref_sem** (ID semid, t_rsem *pk_rsem)
  セマフォの状態参照

イベントフラグ

- ER **stub_cre_flg** (ID flgid, const t_cflg *pk_cflg)
  イベントフラグの生成

- ER **stub_del_flg** (ID flgid)
  イベントフラグの削除

- ER **stub_set_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット

- ER **stub_clr_flg** (ID flgid, FLGPTN clrptn)
  イベントフラグのクリア

- ER **stub_wai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち

- ER **stub_pol_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn)
  イベントフラグ待ち（ポーリング）

- ER **stub_twai_flg** (ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout)
  イベントフラグ待ち（タイムアウト指定）

- ER **stub_ref_flg** (ID flgid, t_rflg *pk_rflg)
  イベントフラグの状態参照

データキュー

- ER **stub_cre_dtq** (ID dtqid, const t_cdtq *pk_cdtq)
  データキューの生成

- ER **stub_del_dtq** (ID dtqid)
  データキューの削除

- ER **stub_snd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信

- ER **stub_psnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（ポーリング）

- ER **stub_tsnd_dtq** (ID dtqid, VP_INT data, TMO tmout)
  データキューへの送信（タイムアウト指定）

- ER **stub_fsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの強制送信

- ER **stub_rcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信

- ER **stub_prcv_dtq** (ID dtqid, VP_INT *p_data)
  データキューからの受信（ポーリング）

- ER **stub_trcv_dtq** (ID dtqid, VP_INT *p_data, TMO tmout)
  データキューからの受信（タイムアウト指定）

- ER **stub_ref_dtq** (ID dtqid, t_rdtq *pk_rdtq)
  データキューの状態参照

メールボックス

- ER **stub_cre_mbx** (ID mbxid, const t_cmbx *pk_cmbx)
  メールボックスの生成

- ER **stub_del_mbx** (ID mbxid)
  メールボックスの削除

- ER **stub_snd_mbx** (ID mbxid, T_MSG *pk_msg)
  メールボックスへの送信

- ER **stub_rcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信

- ER **stub_prcv_mbx** (ID mbxid, T_MSG **ppk_msg)
  メールボックスからの受信（ポーリング）

- ER **stub_trcv_mbx** (ID mbxid, T_MSG **ppk_msg, TMO tmout)
  メールボックスからの受信（タイムアウト指定）

- ER **stub_ref_mbx** (ID mbxid, t_rmbx *pk_rmbx)
  メールボックスの状態参照

固定長メモリプール

- ER **stub_cre_mpf** (ID mpfid, const t_cmpf *pk_cmpf)
  固定長メモリプールの生成

- ER **stub_del_mpf** (ID mpfid)
  固定長メモリプールの削除

- ER **stub_rel_mpf** (ID mpfid, VP blk)
  固定長メモリブロックの解放

- ER **stub_get_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得

- ER **stub_pget_mpf** (ID mpfid, VP *p_blk)
  固定長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpf** (ID mpfid, VP *p_blk, TMO tmout)
  固定長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpf** (ID mpfid, t_rmpf *pk_rmpf)
  固定長メモリプールの状態参照

時間管理機能

- ER **stub_set_tim** (const SYSTIM *p_systim)
  システム時刻の設定

- ER **stub_get_tim** (SYSTIM *p_systim)
  システム時刻の参照

周期ハンドラ

- ER **stub_cre_cyc** (ID cycid, const t_ccyc *pk_ccyc)
  周期ハンドラの生成

- ER **stub_del_cyc** (ID cycid)
  周期ハンドラの削除

- ER **stub_sta_cyc** (ID cycid)
  周期ハンドラの起動

- ER **stub_stp_cyc** (ID cycid)
  周期ハンドラの停止

- ER **stub_ref_cyc** (ID cycid, t_rcyc *pk_rcyc)

周期ハンドラの状態参照

システム状態管理機能

- ER **stub_rot_rdq** (PRI tskpri)
  タスク優先順位の回転

- ER **stub_get_tid** (ID *p_tskid)
  実行状態のタスクIDの参照

- ER **stub_loc_cpu** ()
  CPUロック状態への移行.

- ER **stub_unl_cpu** ()
  CPUロック状態の解除.

- ER **stub_dis_dsp** ()
  ディスパッチの禁止

- ER **stub_ena_dsp** ()
  ディスパッチの許可

- bool **stub_sns_ctx** ()
  コンテキストの参照

- bool **stub_sns_loc** ()
  CPUロック状態の参照.

- bool **stub_sns_dsp** ()
  ディスパッチ禁止状態の参照

- bool **stub_sns_dpn** ()
  ディスパッチ保留状態の参照

- ER **stub_ref_sys** (t_rsys *pk_rsys)
  システム状態の参照

割込み管理機能

- ER **stub_def_inh** (INHNO inhno, const t_dinh *pk_dinh)
  割込みハンドラの定義

- ER **stub_cre_isr** (ID isrid, const t_cisr *pk_cisr)
  割込みサービスルーチンの生成

- ER **stub_del_isr** (ID isrid)
  割込みサービスルーチンの削除

- ER **stub_ref_isr** (ID isrid, t_risr *pk_risr)
  割込みサービスルーチンの状態参照

- ER **stub_dis_int** (INTNO intno)
  割込みの禁止

- ER **stub_ena_int** (INTNO intno)
  割込みの許可

- ER **stub_chg_ims** (IMS ims)
  割込みマスクの変更

- ER **stub_get_ims** (IMS *P_ims)
  割込みマスクの参照

システム構成管理機能

- ER **stub_def_exc** (EXCNO excno, const t_dexc *pk_dexc)
  CPU例外ハンドラの定義.

- ER **stub_ref_cfg** (t_rcfg *pk_rcfg)
  コンフィギュレーション情報の参照

- ER **stub_ref_ver** (t_rver *pk_rver)
  バージョン情報の参照

非タスクコンテキスト用のサービスコール

- ER **stub_iact_tsk** (ID tskid)
  タスクの起動（非タスクコンテキスト）

- ER **stub_iwup_tsk** (ID tskid)
  タスクの起床（非タスクコンテキスト）

- ER **stub_irel_wai** (ID tskid)
  待ち状態の強制解除（非タスクコンテキスト）

- ER **stub_iras_tex** (ID tskid, TEXPTN rasptn)
  タスク例外処理の要求（非タスクコンテキスト）

- ER **stub_isig_sem** (ID semid)
  セマフォ資源の返却（非タスクコンテキスト）

- ER **stub_iset_flg** (ID flgid, FLGPTN setptn)
  イベントフラグのセット（非タスクコンテキスト）

- ER **stub_ipsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの送信（非タスクコンテキスト）

- ER **stub_ifsnd_dtq** (ID dtqid, VP_INT data)
  データキューへの強制送信（非タスクコンテキスト）

- ER **stub_irot_rdq** (PRI tskpri)
  タスク優先順位の回転（非タスクコンテキスト）

- ER **stub_iget_tid** (ID *p_tskid)
  実行状態のタスクIDの参照（非タスクコンテキスト）

- ER **stub_iloc_cpu** ()
  CPUロック状態への移行（非タスクコンテキスト）.

- ER **stub_iunl_cpu** ()
  CPUロック状態の解除（非タスクコンテキスト）.

- ER **stub_isig_tim** ()
  タイムティックの供給

ミューテックス

- ER **stub_cre_mtx** (ID mtxid, const t_cmtx *pk_cmtx)
  ミューテックスの生成

- ER **stub_del_mtx** (ID mtxid)
  ミューテックスの削除

- ER **stub_unl_mtx** (ID mtxid)
  ミューテックスのロック解除

- ER **stub_loc_mtx** (ID mtxid)
  ミューテックスのロック

- ER **stub_ploc_mtx** (ID mtxid)
  ミューテックスのロック（ポーリング）

- ER **stub_tloc_mtx** (ID mtxid, TMO tmout)
  ミューテックスのロック（タイムアウト指定）

- ER **stub_ref_mtx** (ID mtxid, t_rmtx *pk_rmtx)
  ミューテックスの状態参照

メッセージバッファ

- ER **stub_cre_mbf** (ID mbfid, const t_cmbf *pk_cmbf)
  メッセージバッファの生成

- ER **stub_del_mbf** (ID mbfid)
  メッセージバッファの削除

- ER **stub_snd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信

- ER **stub_psnd_mbf** (ID mbfid, const void *msg, UINT msgsz)
  メッセージバッファへの送信（ポーリング）

- ER **stub_tsnd_mbf** (ID mbfid, const void *msg, UINT msgsz, TMO tmout)
  メッセージバッファへの送信（タイムアウト指定）

- ER_UINT **stub_rcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信

- ER_UINT **stub_prcv_mbf** (ID mbfid, void *msg)
  メッセージバッファからの受信（ポーリング）

- ER_UINT **stub_trcv_mbf** (ID mbfid, void *msg, TMO tmout)
  メッセージバッファからの受信（タイムアウト指定）

- ER **stub_ref_mbf** (ID mbfid, t_rmbf *pk_rmbf)
  メッセージバッファの状態参照

ランデブ

- ER **stub_cre_por** (ID porid, const t_cpor *pk_cpor)
  ランデブポートの生成

- ER **stub_del_por** (ID porid)
  ランデブポートの削除

- ER_UINT **stub_cal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz)
  ランデブポートの呼出し

- ER_UINT **stub_tcal_por** (ID porid, RDVPTN calptn, void *msg, UINT cmsgsz, TMO tmout)
  ランデブポートの呼出し（タイムアウト指定）

- ER_UINT **stub_acp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付

- ER_UINT **stub_pacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg)
  ランデブポートの受付（ポーリング）

- ER_UINT **stub_tacp_por** (ID porid, RDVPTN acpptn, RDVNO *p_rdvno, void *msg, TMO tmout)
  ランデブポートの受付（タイムアウト指定）

- ER **stub_fwd_por** (ID porid, RDVPTN acpptn, RDVNO rdvno, const void *msg, UINT cmsgsz)
  ランデブの回送

- ER **stub_rpl_rdv** (RDVNO rdvno, const void *msg, UINT rmsgsz)
  ランデブの返答

- ER **stub_ref_por** (ID porid, t_rpor *pk_rpor)
  ランデブポートの状態参照

- ER **stub_ref_rdv** (RDVNO rdvid, t_rrdv *pk_rrdv)
  ランデブの状態参照

可変長メモリプール

- ER **stub_cre_mpl** (ID mplid, const t_cmpl *pk_cmpl)

可変長メモリプールの生成

- ER **stub_del_mpl** (ID mplid)
  可変長メモリプールの削除

- ER **stub_rel_mpl** (ID mplid, VP blk)
  可変長メモリブロックの解放

- ER **stub_get_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得

- ER **stub_pget_mpl** (ID mplid, UINT blksz, VP *p_blk)
  可変長メモリブロックの獲得（ポーリング）

- ER **stub_tget_mpl** (ID mplid, UINT blksz, VP *p_blk, TMO tmout)
  可変長メモリブロックの獲得（タイムアウト指定）

- ER **stub_ref_mpl** (ID mplid, t_rmpl *pk_rmpl)
  可変長メモリプールの状態参照

アラームハンドラ

- ER **stub_cre_alm** (ID almid, const t_calm *pk_calm)
  アラームハンドラの生成

- ER **stub_del_alm** (ID almid)
  アラームハンドラの削除

- ER **stub_sta_alm** (ID almid, RELTIM almtim)
  アラームハンドラの起動

- ER **stub_stp_alm** (ID almid)
  アラームハンドラの停止

- ER **stub_ref_alm** (ID almid, t_ralm *pk_ralm)
  アラームハンドラの状態参照

オーバーランハンドラ

- ER **stub_def_ovr** (const t_dovr *pk_dovr)
  オーバーランハンドラの定義

- ER **stub_sta_ovr** (ID tskid, OVRTIM ovrtim)
  オーバーランハンドラの起動

- ER **stub_stp_ovr** (ID tskid)
  オーバーランハンドラの停止

- ER **stub_ref_ovr** (ID tskid, t_rovr *pk_rovr)
  オーバーランハンドラの状態参照

ID番号自動生成

- ER_ID **stub_acre_tsk** (const t_ctsk *pk_ctsk)
  タスクの生成（ID番号自動生成）

- ER_ID **stub_acre_sem** (const t_csem *pk_csem)
  セマフォの生成（ID番号自動生成）

- ER_ID **stub_acre_flg** (const t_cflg *pk_cflg)
  イベントフラグの生成（ID番号自動生成）

- ER_ID **stub_acre_dtq** (const t_cdtq *pk_cdtq)
  データキューの生成（ID番号自動生成）

- ER_ID **stub_acre_mbx** (const t_cmbx *pk_cmbx)
  メールボックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mtx** (const t_cmtx *pk_cmtx)
  ミューテックスの生成（ID番号自動生成）

- ER_ID **stub_acre_mbf** (const t_cmbf *pk_cmbf)
  メッセージバッファの生成（ID番号自動生成）

- ER_ID **stub_acre_por** (const t_cpor *pk_cpor)
  ランデブポートの生成（ID番号自動生成）

- ER_ID **stub_acre_mpf** (const t_cmpf *pk_cmpf)
  固定長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_mpl** (const t_cmpl *pk_cmpl)
  可変長メモリプールの生成（ID番号自動生成）

- ER_ID **stub_acre_cyc** (const t_ccyc *pk_ccyc)
  周期ハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_alm** (const t_calm *pk_calm)
  アラームハンドラの生成（ID番号自動生成）

- ER_ID **stub_acre_isr** (const t_cisr *pk_cisr)
  割込みサービスルーチンの生成（ID番号自動生成）

TOPPERSカーネルの独自仕様

これらはTOPPERSカーネル独自のサービスコールだが必須の要素であり、他のカーネルを使用する場合でも対応するサービスコールを実装する 必要がある。

- bool **stub_vxsns_ctx** (VP_INT exinf)
  CPU例外発生時のコンテキストの参照.

- bool **stub_vxsns_loc** (VP_INT exinf)
  CPU例外発生時のCPUロック状態の参照.

- bool **stub_vxsns_dsp** (VP_INT exinf)
  CPU例外発生時のディスパッチ禁止状態の参照.

- bool **stub_vxsns_dpn** (VP_INT exinf)
  CPU例外発生時のディスパッチ保留状態の参照.

- bool **stub_vxsns_tex** (VP_INT exinf)
  CPU例外発生時のタスク例外処理禁止状態の参照.

- bool **stub_vsns_ini** ()
  カーネル動作状態の参照

TOPPERSカーネルの独自拡張

- ER **stub_vxget_tim** (SYSUTIM *p_sysutim)
  性能評価用システム時刻の参照

サービスコール管理機能

- ER **stub_def_svc** (FN fncd, const t_dsvc *pk_dsvc)
  拡張サービスコールの定義

- ER_UINT **stub_cal_svc** (FN fncd)
  拡張サービスコールの呼出し（パラメータなし）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1)
  拡張サービスコールの呼出し（パラメータ x 1）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2)
  拡張サービスコールの呼出し（パラメータ x 2）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3)
  拡張サービスコールの呼出し（パラメータ x 3）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4)
  拡張サービスコールの呼出し（パラメータ x 4）

- ER_UINT **stub_cal_svc** (FN fncd, VP_INT par1, VP_INT par2, VP_INT par3, VP_INT par4, VP_INT par5)
  拡張サービスコールの呼出し（パラメータ x 5）

解説

template<bool Exist = true> class toppers::raw_stub< Exist >

このサービスコールスタブは、ほとんど何の手も加えることなく、該当するサービスコールを呼び出す。

テンプレート引数*Exist* に*true* を指定した場合、カーネルがサポートしないサービスコールを呼び出す ような処理をコンパイル時にエラーにする。また、*Exist* に*false* を指定した場合、サポートされていない 可能性のあるサービスコールは*E_NOSPT* を返すように展開される。 ただし、*stub_exd_tsk* は返却値を持たないため、*std::exit(EXIT_FAILURE)* が呼び出され、プログラムが終了する。

Exist に***has_svc***<*tfn_サービスコール名*>*::value* を指定すれば、該当するサービスコールがサポートされていれば、 そのまま呼び出され、サポートされていない場合は*E_NOSPT* を返すようにすることができる。

クラス テンプレートtoppers::readyqueue< Diagnostics, Stub > の解説

レディキューの操作をカプセル化したテンプレートクラス

`#include <toppers/kernelstatus.hpp>`

Static Public メンバ関数

- ER **rotate** (PRI tskpri, **task_context_tag**=task_context)
  タスク優先順位の回転

- ER **rotate** (PRI tskpri, **non_task_context_tag**)
  タスク優先順位の回転（非タスクコンテキスト）

- ER **rotate** (PRI tskpri, **context_independent_tag**)
  タスク優先順位の回転（コンテキスト非依存）

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::readyqueue< Diagnostics, Stub >

readyqueue テンプレートクラスは、タスク優先順位の回転操作をカプセル化している。各メンバ関数は実行コンテキストをパラメータ化することで汎用性を持たせている。

クラス テンプレートtoppers::recursive_lock< LockObj, SenseKernel > の解説

ネスト可能なクリティカルセクションを生成するためのテンプレートクラス

`#include <toppers/sync.hpp>`

Public メンバ関数

- **recursive_lock** ()
  デフォルトコンストラクタ

- template<typename T> **recursive_lock** (T arg)
  パラメータ指定のコンストラクタ

- **~recursive_lock** () throw ()
  デストラクタ

解説

template<class LockObj, bool SenseKernel = false> class toppers::recursive_lock< LockObj, SenseKernel >

参照:

lock

例外安全なクリティカルセクションを、指定したロック機構を用いて生成する。 **lock** テンプレートクラスとは異なり、本クラスではネスティングが可能である。 誤使用を避けるため、このクラスではポーリングやタイムアウト指定はサポートしない。

テンプレート引数*LockObj* には、*semaphore* や*mutex* 、または*cpulock* 等を指定することができる。*LockObj* に*void* を指定した場合には、 実際にはクリティカルセクションが生成されない。

テンプレート引数*Sensekernel* は、クリティカルセクションを生成するにあたり、 予めカーネルの動作状態を*vsns_ini* によって参照するかどうかを指定することができる。*SenseKernel* が*true* であればカーネル動作状態が参照される。

注意:

デフォルトでは*SenseKernel* は*false* になっているため、 カーネルが動作している必要がある。

クラス テンプレートtoppers::rendezvous< Diagnostics, Stub > の解説

ランデブ番号をカプセル化したテンプレートクラス

`#include <toppers/rendezvous.hpp>`

Public 型

- typedef t_rrdv **reference**
  状態参照情報パケット型

Public メンバ関数

- **rendezvous** (RDVNO rdvno=0)
  ランデブ番号指定のコンストラクタ

- ER **refer** (**reference** *pk_robj) const
  ランデブの状態参照

- ER **reply** (const void *msg, UINT rmsgsz)
  ランデブの返答

- RDVNO **no** () const
  ランデブ番号の取得

- void **attach** (RDVNO rdvno)
  ランデブ番号の関連付け

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::rendezvous< Diagnostics, Stub >

参照:

rendezvousport

rendezvous テンプレートクラスは、ランデブ番号の機能をカプセル化し、 待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレートtoppers::rendezvousport< Id, Diagnostics, Stub > の解説

ランデブポートの機能をカプセル化したテンプレートクラス

`#include <toppers/rendezvous.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **rendezvousport** ()
  デフォルトコンストラクタ

- **rendezvousport** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **rendezvousport** (const **rendezvousport**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  ランデブポートの生成

- ER **create** (ATR poratr=TA_TFIFO, UINT maxcmsz=256, UINT maxrmsz=256)
  ランデブポートの生成（個別のフィールド指定）

- ER **destroy** ()
  ランデブポートの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  ランデブポートの状態参照

ランデブ通信

- ER_UINT **call** (RDVPTN calptn, void *msg, UINT cmsgsz, **forever_tag**=forever) const
  ランデブの呼出し

- ER_UINT **call** (RDVPTN calptn, void *msg, UINT cmsgsz, TMO tmout) const
  ランデブの呼出し（タイムアウト指定）

- template<class Diagnostics2, class Stub2> ER_UINT **accept** (RDVPTN acpptn, **rendezvous**< Diagnostics2, Stub2 > &rdvobj, void *msg, **forever_tag**=forever) const
  ランデブの受付

- template<class Diagnostics2, class Stub2> ER_UINT **accept** (RDVPTN acpptn, **rendezvous**< Diagnostics2, Stub2 > &rdvobj, void *msg, **polling_tag**) const
  ランデブの受付（ポーリング）

- template<class Diagnostics2, class Stub2> ER_UINT **accept** (RDVPTN acpptn, **rendezvous**< Diagnostics2, Stub2 > &rdvobj, void *msg, TMO tmout) const
  ランデブの受付（タイムアウト指定）

- template<class Diagnostics2, class Stub2> ER **forward** (RDVPTN calptn, **rendezvous**< Diagnostics2, Stub2 > &rdvobj, const void *msg, UINT cmsgsz) const
  ランデブの回送

フレンド

- ID **get_unsafe_id** (const **rendezvousport**< Id, Diagnostics, Stub > &**rendezvousport**)
  ランデブポートID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::rendezvousport< Id, Diagnostics, Stub >

参照:

rendezvous

rendezvousport テンプレートクラスは、ランデブポートの機能をカプセル化し、 待機方法をパラメータ化するためのインタフェースを提供する。

クラス テンプレートtoppers::semaphore< Id, Diagnostics, Stub > の解説

セマフォの機能をカプセル化したテンプレートクラス

`#include <toppers/semaphore.hpp>`

Public 型

- typedef controller::creation **creation**
  生成情報パケット型

- typedef controller::reference **reference**
  状態参照情報パケット型

Public メンバ関数

コンストラクタ

- **semaphore** ()
  デフォルトコンストラクタ

- **semaphore** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **semaphore** (const **semaphore**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  セマフォの生成

- ER **create** (ATR sematr=TA_TFIFO, UINT isemcnt=1, UINT maxsem=1)
  セマフォの生成（個別のフィールド指定）

- ER **destroy** ()
  セマフォの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  セマフォの状態参照

セマフォ資源の獲得と返却

- ER **signal** (**task_context_tag**=task_context) const
  セマフォ資源の返却

- ER **signal** (**non_task_context_tag**) const
  セマフォ資源の返却（非タスクコンテキスト）

- ER **signal** (**context_independent_tag**) const
  セマフォ資源の返却（コンテキスト非依存）

- ER **wait** (**forever_tag**=forever) const
  セマフォ資源の獲得

- ER **wait** (**polling_tag**) const
  セマフォ資源の獲得（ポーリング）

- ER **wait** (TMO tmout) const
  セマフォ資源の獲得（タイムアウト指定）

フレンド

- ID **get_unsafe_id** (const **semaphore**< Id, Diagnostics, Stub > &**semaphore**)
  セマフォID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::semaphore< Id, Diagnostics, Stub >

参照:

lock recursive_lock

semaphore テンプレートクラスは、セマフォの機能をカプセル化し、 コンテキストや待機方法をパラメータ化するためのインタフェースを提供する。 このクラスで提供されるインタフェースは、他の同期オブジェクトクラスと 交換可能になっている。

クラス テンプレートtoppers::syslog_tracer< Count, BufferSize, Diagnostics > の解説

syslogを用いたデータトレースクラス

`#include <toppers/trace.hpp>`

Public メンバ関数

- **syslog_tracer** (UINT prio=5u)
  コンストラクタ

- void **init** ()
  書式設定の初期化

- UINT **flags** () const
  現在設定されているフラグを返す。

- UINT **flags** (UINT fmtfl)
  フラグを新たに設定し、以前に設定されていたフラグを返す。

- UINT **setf** (UINT fmtfl)
  フラグの設定を追加し、以前に設定されていたフラグを返す。

- UINT **setf** (UINT fmtfl, UINT mask)
  mask で指定されたフィードのフラグの設定を追加し、以前に設定されていたフラグを返す。

- UINT **unsetf** (UINT mask)
  フラグの設定を解除し、以前に設定されていたフラグを返す。

- int **width** () const
  現在設定されている最小フィールド幅を返す。

- int **width** (int wide)
  最小フィールド幅を設定し、以前に設定されていた最小フィールド幅を返す。

- char **fill** () const
  現在設定されている充填文字を返す。

- char **fill** (char c)
  充填文字を設定し、以前に設定されていた充填文字を返す。

- **syslog_tracer**< Count, BufferSize, Diagnostics > & **flush** ()
  トレースバッファの内容を実際に出力する。

- **syslog_tracer**< Count, BufferSize, Diagnostics > & **put** (char c)
  トレースバッファへの1文字書き込み

- **syslog_tracer**< Count, BufferSize, Diagnostics > & **put** (const char *s)
  トレースバッファへの文字列書き込み

Static Public メンバ関数

- void **putter** (char c, void *p)
  ヘルパー関数のコールバック指定用1文字書き込み

解説

template<std::size_t Count = 32, std::size_t BufferSize = 256, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS> class toppers::syslog_tracer< Count, BufferSize, Diagnostics >

syslog_tracer テンプレートクラスは、 TOPPERSカーネルのシステムログを使用したデータトレース機能であり、概ね *std::ostream* のサブセットとしての機能を備えている。

名前空間の違いを無視すれば、 << 演算子と主なマニピュレータだけを用いてコーディングすることにより、後から容易に *std::ostream* に置き換えることができる。

メンバ関数の解説

template<std::size_t Count = 32, std::size_t BufferSize = 256, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS> void toppers::syslog_tracer< Count, BufferSize, Diagnostics >::putter (char *c*, void * *p*) [inline, static]

引数:

c: 書き込む文字
p: syslog_tracer<Count, BufferSize, Diagnostics>へのポインタ

注意:

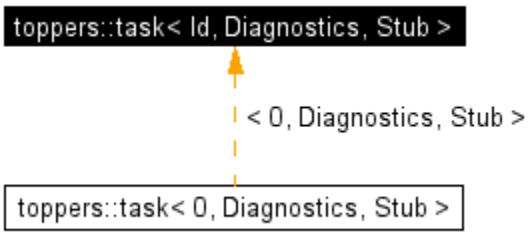
空間効率向上のため、*p* の型は*void** になっているが、 実際に指定する場合は必ずこのメンバー関数が属するクラスへの ポインタを指定する（テンプレート引数まで合致させること）。

クラス テンプレートtoppers::task< Id, Diagnostics, Stub > の解説

タスク機能をカプセル化したテンプレートクラス

`#include <toppers/task.hpp>`

toppers::task< Id, Diagnostics, Stub >に対する継承グラフ



Public 型

- typedef controller::creation **creation**
  タスク生成情報パケット型

- typedef controller::reference **reference**
  タスク状態参照情報パケット型

- typedef t_rtst **easy_reference**
  タスク状態参照情報パケット型（簡易版）

- typedef **task_body**< Diagnostics, Stub > **body**
  タスク本体定義用の基底クラス

Public メンバ関数

コンストラクタ

- **task** ()
  デフォルトコンストラクタ

- **task** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **task** (const **task**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート実引数違いの他のオブジェクトからの変換コンストラクタ

生成と削除

- ER **create** (const **creation** *pk_cobj)
  タスクの生成

- ER **create** (ATR tskatr, VP_INT exinf, void(***task**)(VP_INT), PRI itskpri=(TMIN_TPRI+TMAX_TPRI)/2, SIZE stksz=TOPPERS_STACKSIZE, VP stk=0)
  タスクの生成（個別のフィールド指定）

- template<class Body> ER **create** (ATR tskatr=TA_ACT, VP_INT exinf=0, PRI itskpri=(TMIN_TPRI+TMAX_TPRI)/2, SIZE stksz=TOPPERS_STACKSIZE, VP stk=0)
  タスクの生成（ボディ指定）

- ER **destroy** ()
  タスクの削除

状態参照

- ER **refer** (**reference** *pk_robj) const
  タスクの状態参照

- ER **refer** (**easy_reference** *pk_robj) const
  タスクの状態参照（簡易版）

タスク管理機能

- ER **activate** (**task_context_tag**=task_context) const
  タスクの起動

- ER **activate** (**non_task_context_tag**) const
  タスクの起動（非タスクコンテキスト）

- ER **activate** (**context_independent_tag**) const
  タスクの起動（コンテキスト非依存）

- ER **activate** (VP_INT exinf) const
  タスクの起動（起動コード指定）

- ER_UINT **cancel_activation** () const
  タスク起動要求のキャンセル

- ER **terminate** () const
  タスクの強制終了

- ER **change_priority** (PRI tskpri) const
  タスク優先度の変更

- ER **get_priority** (PRI *p_tskpri) const
  タスク優先度の参照

タスク付属同期機能

- ER **wakeup** (**task_context_tag**=task_context) const
  タスクの起床

- ER **wakeup** (**non_task_context_tag**) const
  タスクの起床（非タスクコンテキスト）

- ER **wakeup** (**context_independent_tag**) const
  タスクの起床（コンテキスト非依存）

- ER_UINT **cancel_wakeup** () const
  タスク起床要求のキャンセル

- ER **release_wait** (**task_context_tag**=task_context) const
  待ち状態の強制解除

- ER **release_wait** (**non_task_context_tag**) const
  待ち状態の強制解除（非タスクコンテキスト）

- ER **release_wait** (**context_independent_tag**) const
  待ち状態の強制解除（コンテキスト非依存）

- ER **suspend** () const
  強制待ち状態への移行

- ER **resume** (**normal_tag**=normal) const
  強制待ち状態からの再開

- ER **resume** (**forced_tag**) const
  強制待ち状態からの強制再開

フレンド

- ID **get_unsafe_id** (const **task**< Id, Diagnostics, Stub > &**task**)
  タスクID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::task< Id, Diagnostics, Stub >

参照:

task_body

task テンプレートクラスは、主として異なるタスクに対する操作をカプセル化している。自タスクに対する操作は、***task_body*** や ***taskexception_body*** のメンバ関数を使用のこと。

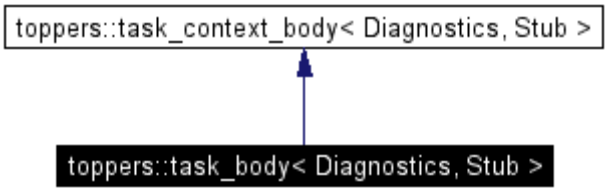
基本概念やテンプレート引数の意味については、**カーネルオブジェクト管理** を参照のこと。

クラス テンプレートtoppers::task_body< Diagnostics, Stub > の解説

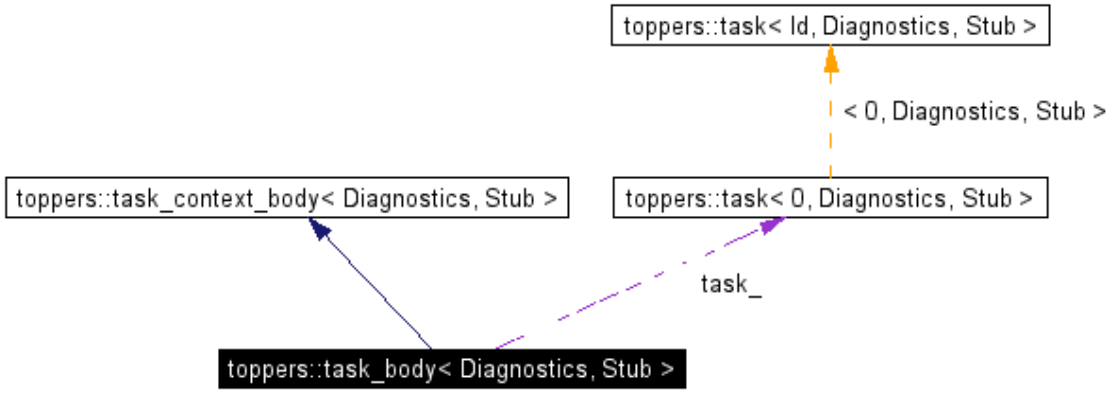
タスク本体のデータおよび動作を定義する基底クラス

`#include <toppers/task.hpp>`

toppers::task_body< Diagnostics, Stub >に対する継承グラフ



toppers::task_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **task_body** (VP_INT exinf)
  拡張情報を受取るコンストラクタ

- const **task**< 0, Diagnostics, Stub > & **get_task** () const
  自タスク管理オブジェクトの参照

- void **run** ()
  タスク本体動作の実行

- VP_INT **get_extended_info** () const
  タスクの拡張情報の参照

Static Public メンバ関数

- void **exit** ()
  タスクの終了

  例外:

  task_body::terminator

- void **exit_and_destroy** ()
  タスクの終了と削除

  例外:

  task_body::terminator

- ER **sleep** (**forever_tag**=forever)
  タスクの起床待ち

- ER **sleep** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **delay** (RELTIM dlytim)
  タスクの遅延

- ER **yield** ()
  タスクの切り替え

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context

非タスクコンテキストを指定する定数

- const **task_context_tag context_independent** = toppers::task_context
  現在のコンテキスト（すなわちタスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **~task_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::task_body< Diagnostics, Stub >

参照:

task task_entry task_body::terminator

タスクを作成する場合、***task_body*** テンプレートクラスから public 継承した クラスを定義し、その型を*task_entry* テンプレート関数のテンプレート引数 に渡す。

```
/*  task_body.h */
#ifndef TASK_BODY_H
#define TASK_BODY_H

#define MAIN_PRIORITY   5      /* メインタスクの優先度 */
```

```
#define STACKSIZE      0x2000

typedef struct
{
   int counter;
} MY_TASK_DATA;

#ifdef  __cplusplus
extern "C" {
#endif

extern MY_TASK_DATA my_task_data;

void my_task(VP_INT);

#ifdef  __cplusplus
}
#endif

#endif
```

```
//  task_body.cpp
#include <toppers/task.hpp>
#include "task_body.h"

MY_TASK_DATA my_task_data = { 0 };

class my_task_body : public toppers::task_body<>
{
public:
   explicit my_task_body(VP_INT exinf)
      : toppers::task_body<>(exinf)
   {
   }

   MY_TASK_DATA& get_data()
   {
      return *(MY_TASK_DATA*)get_extended_info();
   }

   void run()
   {
      for (;;)
      {
         delay(10);
         ++get_data().counter;
      }
   }
};

void my_task(VP_INT exinf)
{
   toppers::task_entry<my_task_body>(exinf);
}
```

```
/*  task_body.cfg   */
INCLUDE("¥"task_body.h¥"");

CRE_TSK(MY_TASK, { TA_HLNG | TA_ACT, (VP_INT)&my_task_data, my_task, MID_PRIORITY, STACK_SIZE,
NULL });

#include "./systask/cxxrt.cfg"
```

```
#include "./systask/timer.cfg"
#include "./systask/serial.cfg"
#include "./systask/logtask.cfg"
```

メンバ関数の解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> void toppers::task_body< Diagnostics, Stub >::exit () [inline, static]

参照:

task_entry

自タスクを終了させる。 自動記憶域に生成されたインスタンスのデストラクタを確実に呼出すため、 直接*ext_tsk* を呼出すのではなく、例外をスローしている。

e task_entry を使用する場合は内部で例外がキャッチされるが、タスクの エントリ関数を自作する場合は関数のトップレベルで例外をキャッチする 必要がある。

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> void toppers::task_body< Diagnostics, Stub >::exit_and_destroy () [inline, static]

参照:

task_entry

自タスクを終了させた後、削除する。 自動記憶域に生成されたインスタンスのデストラクタを確実に呼出すため、 直接*exd_tsk* を呼出すのではなく、例外をスローしている。

e task_entry を使用する場合は内部で例外がキャッチされるが、タスクの エントリ関数を自作する場合は関数のトップレベルで例外をキャッチする 必要がある。

クラス toppers::task_body< Diagnostics, Stub >::terminator の解説

自タスクを終了させる場合にスローする例外クラス

`#include <toppers/task.hpp>`

Public メンバ関数

- **terminator** (void(*exit_proc)())
  コンストラクタ

- void(* **get_exit_proc** ())()
  終了関数の参照

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::task_body< Diagnostics, Stub >::terminator

参照:

task_body::exit

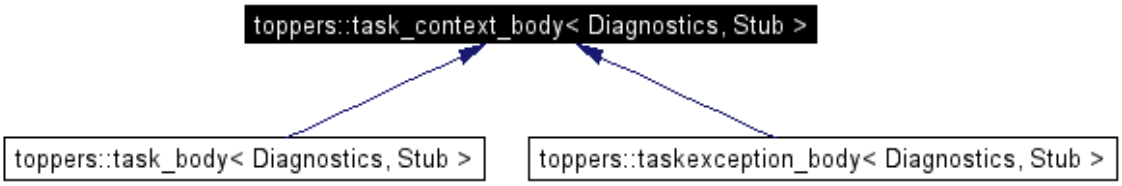
task_body::terminater クラスは、***task_body::exit*** または***task_body::exit_and_destroy*** メンバ関数からスローされる例外クラスである。 このクラスのコンストラクタで指定された終了関数が、例外オブジェクトがキャッチされた後に実行される。

クラス テンプレートtoppers::task_context_body< Diagnostics, Stub > の解説

タスクコンテキストの本体定義用基底クラス

`#include <toppers/context.hpp>`

toppers::task_context_body< Diagnostics, Stub >に対する継承グラフ



Public メンバ関数

- void **run** ()
  タスク本体動作の実行

- VP_INT **get_extended_info** () const
  タスクの拡張情報の参照

Static Public メンバ関数

- ER **sleep** (**forever_tag**=forever)
  タスクの起床待ち

- ER **sleep** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **delay** (RELTIM dlytim)
  タスクの遅延

- ER **yield** ()
  タスクの切り替え

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **task_context_tag context_independent** = toppers::task_context
  現在のコンテキスト（すなわちタスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **task_context_body** (VP_INT exinf)
  コンストラクタ

- **~task_context_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics, class Stub> class toppers::task_context_body< Diagnostics, Stub >

task_context_body テンプレートクラスは、タスクコンテキスト用ボディクラスの基底クラスとして使用される。

タスクコンテキストにおいて、自タスクに対する主な操作はこのクラスでカプセル化されている。 ***task_context_body*** テンプレートクラスから派生したクラスでは、*context_independent* 定数は *task_context* 定数と同じであり、シンタックスを汎用化するために使用することができる。

クラス テンプレート**toppers::taskexception< Diagnostics, Stub >** の解説

タスク例外処理機能をカプセル化したテンプレートクラス

`#include <toppers/taskexception.hpp>`

Public 型

- typedef t_dtex **definition**
  定義情報パケット型

- typedef t_rtex **reference**
  状態参照情報パケット型

- typedef **taskexception_body**< Diagnostics, Stub > **body**
  ボディクラス

Static Public メンバ関数

タスク例外処理ルーチンの定義

- template<ID Id, class Diagnostics2, class Stub2> ER **define** (const **task**< Id, Diagnostics2, Stub2 > &host, const **definition** *pk_cobj)
  タスク例外処理ルーチンの定義

- template<ID Id, class Diagnostics2, class Stub2> ER **define** (const **task**< Id, Diagnostics2, Stub2 > &host, ATR texatr, void(*texrtn)(TEXPTN, VP_INT))
  タスク例外処理ルーチンの定義（個別のフィールド指定）

- template<class Body, ID Id, class Diagnostics2, class Stub2> ER **define** (const **task**< Id, Diagnostics2, Stub2 > &host, ATR texatr=0)
  タスク例外処理ルーチンの定義（ボディ指定）

- ER **define** (const **definition** *pk_cobj)
  自タスクに対するタスク例外処理ルーチンの定義

- ER **define** (ATR texatr, void(*texrtn)(TEXPTN, VP_INT))
  自タスクに対するタスク例外処理ルーチンの定義（個別のフィールド指定）

- template<class Body> ER **define** (ATR texatr=0)
  自タスクに対するタスク例外処理ルーチンの定義（ボディ指定）

- template<ID Id, class Diagnostics2, class Stub2> ER **undefine** (const **task**< Id, Diagnostics2, Stub2 > &host)
  タスク例外処理ルーチンの定義取り消し

- ER **undefine** ()
  自タスクに対するタスク例外処理ルーチンの定義取り消し

状態参照

- template<ID Id, class Diagnostics2, class Stub2> ER **refer** (const **task**< Id, Diagnostics2, Stub2 > &host, **reference** *pk_robj)
  タスク例外処理の状態参照

- ER **refer** (**reference** *pk_robj)
  自タスクに対するタスク例外処理の状態参照

例外要求

- template<ID Id, class Diagnostics2, class Stub2> ER **raise** (const **task**< Id, Diagnostics2, Stub2 > &host, TEXPTN rasptn, **task_context_tag**=task_context)
  タスク例外処理の要求

- template<ID Id, class Diagnostics2, class Stub2> ER **raise** (const **task**< Id, Diagnostics2, Stub2 > &host, TEXPTN rasptn, **non_task_context_tag**)
  タスク例外処理の要求（非タスクコンテキスト）

- template<ID Id, class Diagnostics2, class Stub2> ER **raise** (const **task**< Id, Diagnostics2, Stub2 > &host, TEXPTN rasptn, **context_independent_tag**)
  タスク例外処理の要求（コンテキスト非依存）

- ER **raise** (TEXPTN rasptn, **task_context_tag**=task_context)
  自タスクに対するタスク例外処理の要求

- ER **raise** (TEXPTN rasptn, **non_task_context_tag**)
  自タスクに対するタスク例外処理の要求（非タスクコンテキスト）

- ER **raise** (TEXPTN rasptn, **context_independent_tag**)
  自タスクに対するタスク例外処理の要求（コンテキスト非依存）

禁止と許可

- ER **disable** ()
  タスク例外処理の禁止

- ER **enable** ()
  タスク例外処理の許可

- bool **sense** ()
  タスク例外処理禁止状態の参照

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::taskexception< Diagnostics, Stub >

参照:

taskexception_body

警告:

C++ではタスク例外処理は必ずしも安全ではない。使用に際しては慎重に検討すること。

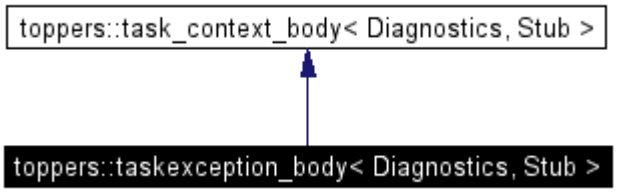
taskexception テンプレートクラスは、μITRON 4.0仕様のタスク例外処理機能をカプセル化している。 このクラスのメンバ関数は全て静的メンバ関数として実装されている。 また、ほとんどのメンバ関数は*task* テンプレートクラスへの参照を引数として渡す汎用のものと、 *task* への参照を渡さない自タスク用のものがある。

クラス テンプレートtoppers::taskexception_body< Diagnostics, Stub > の解説

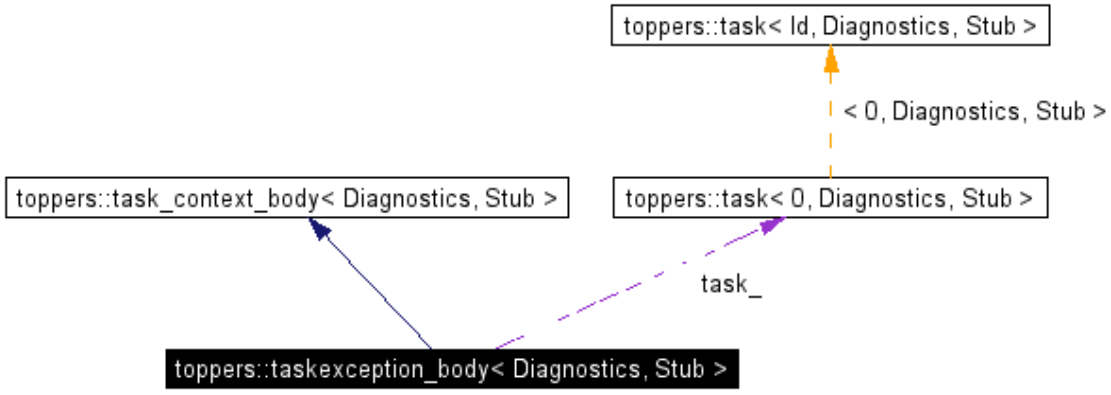
タスク例外処理ルーチンの本体動作を定義する基底クラス

`#include <toppers/taskexception.hpp>`

toppers::taskexception_body< Diagnostics, Stub >に対する継承グラフ



toppers::taskexception_body< Diagnostics, Stub >のコラボレーション図



Public メンバ関数

- **taskexception_body** (TEXPTN texptn, VP_INT exinf)
  コンストラクタ

- const **task**< 0, Diagnostics, Stub > & **get_task** () const
  自タスク管理オブジェクトの参照

- void **run** ()
  タスク本体動作の実行

- VP_INT **get_extended_info** () const
  タスクの拡張情報の参照

Static Public メンバ関数

- bool **filter** (TEXPTN texptn)
  タスク例外処理マスク情報によるフィルタリング

- ER **change_mask** (TEXPTN mask)
  タスク例外処理マスク情報の変更

- ER **get_mask** (TEXPTN *p_mask)
  タスク例外処理マスク情報の参照

- ER **sleep** (**forever_tag**=forever)
  タスクの起床待ち

- ER **sleep** (TMO tmout)
  タスクの起床待ち（タイムアウト指定）

- ER **delay** (RELTIM dlytim)
  タスクの遅延

- ER **yield** ()
  タスクの切り替え

Static Public 変数

- const **task_context_tag task_context** = toppers::task_context
  タスクコンテキストを指定する定数

- const **non_task_context_tag non_task_context** = toppers::non_task_context
  非タスクコンテキストを指定する定数

- const **task_context_tag context_independent** = toppers::task_context

現在のコンテキスト（すなわちタスクコンテキスト）を指定する定数

Protected 型

- typedef Stub **stub**
  サービスコールスタブ

Protected メンバ関数

- **~taskexception_body** ()
  デストラクタ

Static Protected メンバ関数

- ID **get_task_id** ()
  実行中のタスクID番号を参照する

解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::taskexception_body< Diagnostics, Stub >

参照:

taskexception taskexception_entry

メンバ関数の解説

template<class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> bool toppers::taskexception_body< Diagnostics, Stub >::filter (TEXPTN *texptn*) `[inline, static]`

タスク例外処理のマスク機能を使わない場合は、派生クラスで filter を再定義し、常に*true* を返すようにすればよい。

クラス **toppers::variable_new_pool** の解説

グローバルな new および delete 演算子による可変長メモリプール

```
#include <toppers/allocator.hpp>
```

Public メンバ関数

- ER **get** (UINT size, VP *p_blk)
  可変長メモリブロックの獲得（サイズ指定との互換用）

- ER **get** (UINT size, VP *p_blk, int sync)
  可変長メモリブロックの獲得（サイズ指定および同期方法指定との互換用）

- ER **release** (VP blk)
  可変長メモリブロックの解放

解説

variable_new_pool クラスは、グローバルな new および delete 演算子を用いて、 可変長メモリプールの体裁を持たせたクラスである。 本来メモリプールを使用すべきところにこのクラスを指定することで、グローバルな new および delete 演算子を用いた動作に変更することができる。

クラス テンプレートtoppers::variable_simple_pool< TotalSize > の解説

排他制御のない可変長メモリプール

`#include <toppers/allocator.hpp>`

Public メンバ関数

- **variable_simple_pool** ()
  コンストラクタ

- ER **get** (UINT size, VP *p_blk)
  可変長メモリブロックの獲得

- ER **get** (UINT size, VP *p_blk, int sync)
  可変長メモリブロックの獲得（同期方法指定との互換用）

- ER **release** (VP blk)
  可変長メモリブロックの解放

解説

template<std::size_t TotalSize = 256> class toppers::variable_simple_pool< TotalSize >

e **variable_simple_pool** テンプレートクラスは、テンプレート引数*TotalSize* で指定したサイズの配列を 先頭から分割するだけの単純な可変長メモリプールである。

e **variable_simple_pool** は、構造を単純化することで高速化を図っているため、排他制御は行っていない。

注意:

このメモリプールによって一度割り付けたメモリブロックは、メモリプールが削除されるまで再利用できない。

クラス テンプレートtoppers::variable_sized_new< Host, Pool > の解説

指定メモリプールを用いた new および delete 演算子

`#include <toppers/allocator.hpp>`

Public 型

- typedef void(* **handler_type**)()
  new 演算子エラー処理関数型

Public メンバ関数

- **handler_type set_new_handler** (**handler_type** handler)
  new演算子に失敗した際の処理関数の設定

Static Public メンバ関数

- void * **operator new** (std::size_t size) throw (std::bad_alloc)
  new演算子

- void **operator delete** (void *p) throw ()
  delete演算子

- template<typename Sync> void * **operator new** (std::size_t size, Sync sync) throw (std::bad_alloc)
  同期方法指定付き new 演算子

- template<typename Sync> void **operator delete** (void *p, Sync) throw ()
  同期方法指定付きnew演算子に対応するdelete演算子

- void * **operator new** (std::size_t size, const std::nothrow_t &nt) throw ()
  例外を投げないnew演算子

- void **operator delete** (void *p, const std::nothrow_t &) throw ()
  例外を投げないnew演算子に対応するdelete演算子

- template<typename Sync> void * **operator new** (std::size_t size, const std::nothrow_t &, Sync sync) throw ()
  例外を投げない同期方法指定付きnew演算子

- template<typename Sync> void **operator delete** (void *p, const std::nothrow_t &, Sync sync) throw ()
  例外を投げない同期方法指定付きnew演算子に対応するdelete演算子

- void * **operator new[]** (std::size_t size) throw (std::bad_alloc)
  new演算子（配列用）

- void **operator delete[]** (void *p) throw ()
  delete演算子（配列用）

- template<typename Sync> void * **operator new[]** (std::size_t size, Sync sync) throw (std::bad_alloc)
  同期方法指定付きnew演算子（配列用）

- template<typename Sync> void **operator delete[]** (void *p, Sync) throw ()
  同期方法指定付きnew演算子に対応するdelete演算子（配列用）

- void * **operator new[]** (std::size_t size, const std::nothrow_t &nt) throw ()
  例外を投げないnew演算子（配列用）

- void **operator delete[]** (void *p, const std::nothrow_t &nt) throw ()
  例外を投げないnew演算子に対応するdelete演算子（配列用）

- template<typename Sync> void * **operator new[]** (std::size_t size, const std::nothrow_t &nt, Sync sync) throw ()
  例外を投げない同期方法指定付きnew演算子（配列用）

- template<typename Sync> void **operator delete[]** (void *p, const std::nothrow_t &nt, Sync sync) throw ()
  例外を投げない同期方法指定付きnew演算子に対応するdelete演算子（配列用）

Protected メンバ関数

- **~variable_sized_new** ()
  デストラクタ

解説

template<class Host, class Pool> class toppers::variable_sized_new< Host, Pool >

variable_sized_new は、テンプレート引数*Pool* で指定した可変長メモリプールを用いた new および delete 演算子を定義している。

variable_sized_new から public 継承することにより、そのクラスの new および delete 演算子をグローバルな new および delete 演算子と置き換えることができる。

テンプレート引数*Host* には、***variable_sized_new*** から派生させるクラスそのものを指定すること。

コンストラクタとデストラクタの解説

template<class Host, class Pool> toppers::variable_sized_new< Host, Pool >::~variable_sized_new () [inline, protected]

variable_sized_newのポインタを用いて派生クラスがdeleteされることがないよう protected属性とする。

メンバ関数の解説

template<class Host, class Pool> template<typename Sync> void* toppers::variable_sized_new< Host, Pool >::operator new (std::size_t *size*, Sync *sync*) throw (std::bad_alloc) [inline, static]

引数:

sync 同期方法。forever、polling、およびタイムアウト時間を指定できる。

クラス テンプレートtoppers::vmempool< Id, Diagnostics, Stub > の解説

可変長メモリプールの機能をカプセル化したテンプレートクラス

`#include <toppers/vmempool.hpp>`

Public メンバ関数

コンストラクタ

- **vmempool** ()
  デフォルトコンストラクタ

- **vmempool** (ID id)
  ID番号指定のコンストラクタ.

- template<ID Id2, class Diagnostics2, class Stub2> **vmempool** (const **vmempool**< Id2, Diagnostics2, Stub2 > &other)
  テンプレート引数違いの変換コンストラクタ

生成と削除

- ER **create** (const creation *pk_cobj)
  可変長メモリプールの生成

- ER **create** (ATR mplatr=TA_TFIFO, UINT mplsz=1024, VP mpl=0)
  可変長メモリプールの生成（個別のフィールド指定）

- ER **destroy** ()
  可変長メモリプールの削除

状態参照

- ER **refer** (reference *pk_robj) const
  可変長メモリプールの状態参照

メモリブロックの獲得と解放

- ER **get** (UINT blksz, VP *p_blk, **forever_tag**=forever) const
  可変長メモリブロックの獲得

- ER **get** (UINT blksz, VP *p_blk, **polling_tag**) const
  可変長メモリブロックの獲得（ポーリング）

- ER **get** (UINT blksz, VP *p_blk, TMO tmout) const
  可変長メモリブロックの獲得（タイムアウト指定）

- ER **release** (VP blk) const
  可変長メモリブロックの解放

フレンド

- ID **get_unsafe_id** (const **vmempool**< Id, Diagnostics, Stub > &**vmempool**)
  可変長メモリプールID番号の取得

解説

template<ID Id = 0, class Diagnostics = TOPPERS_DEFAULT_DIAGNOSTICS, class Stub = TOPPERS_DEFAULT_STUB> class toppers::vmempool< Id, Diagnostics, Stub >

参照:

fmempool

vmempool テンプレートクラスは、可変長メモリプールをカプセル化するとともに、 他のメモリプールと交換可能なインタフェースを提供している。

注意:

このテンプレートクラスは拡張サービスコールを使用しているため、 スタンダードプロファイル準拠のカーネル上ではコンパイルエラーになる。

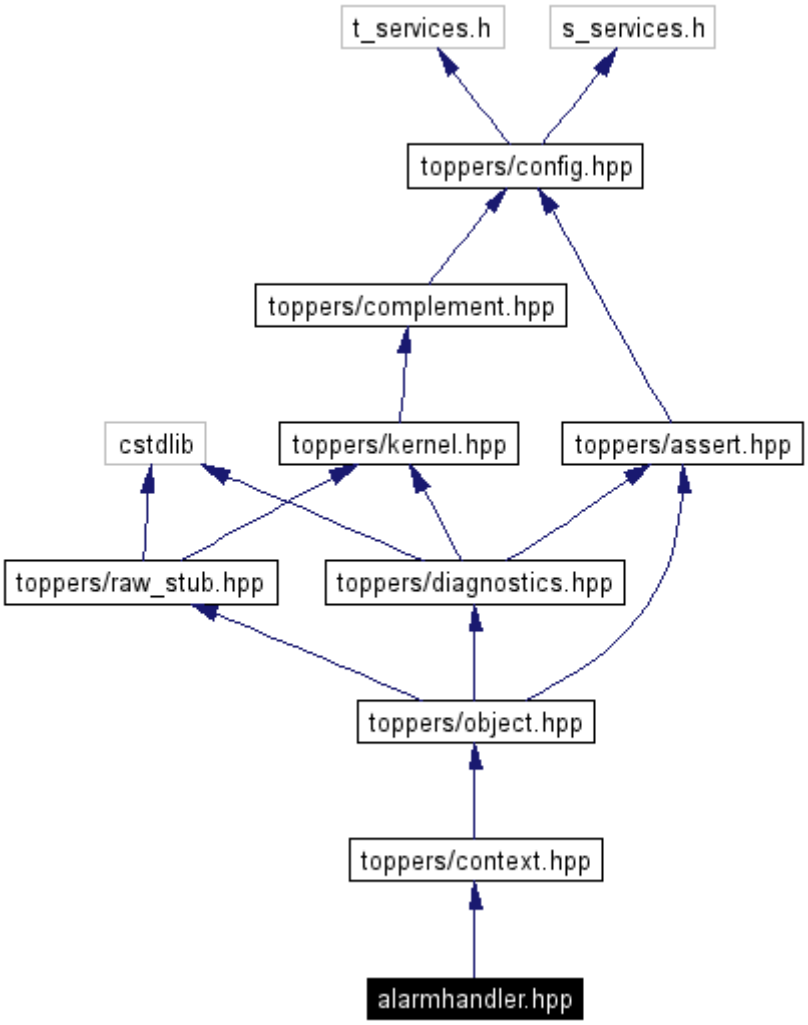
TOPPERS C++ Wrapper Library ファイルの解説

toppers/alarmhandler.hpp の解説

アラームハンドラを管理・制御するクラスの定義

```
#include <toppers/context.hpp>
```

alarmhandler.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

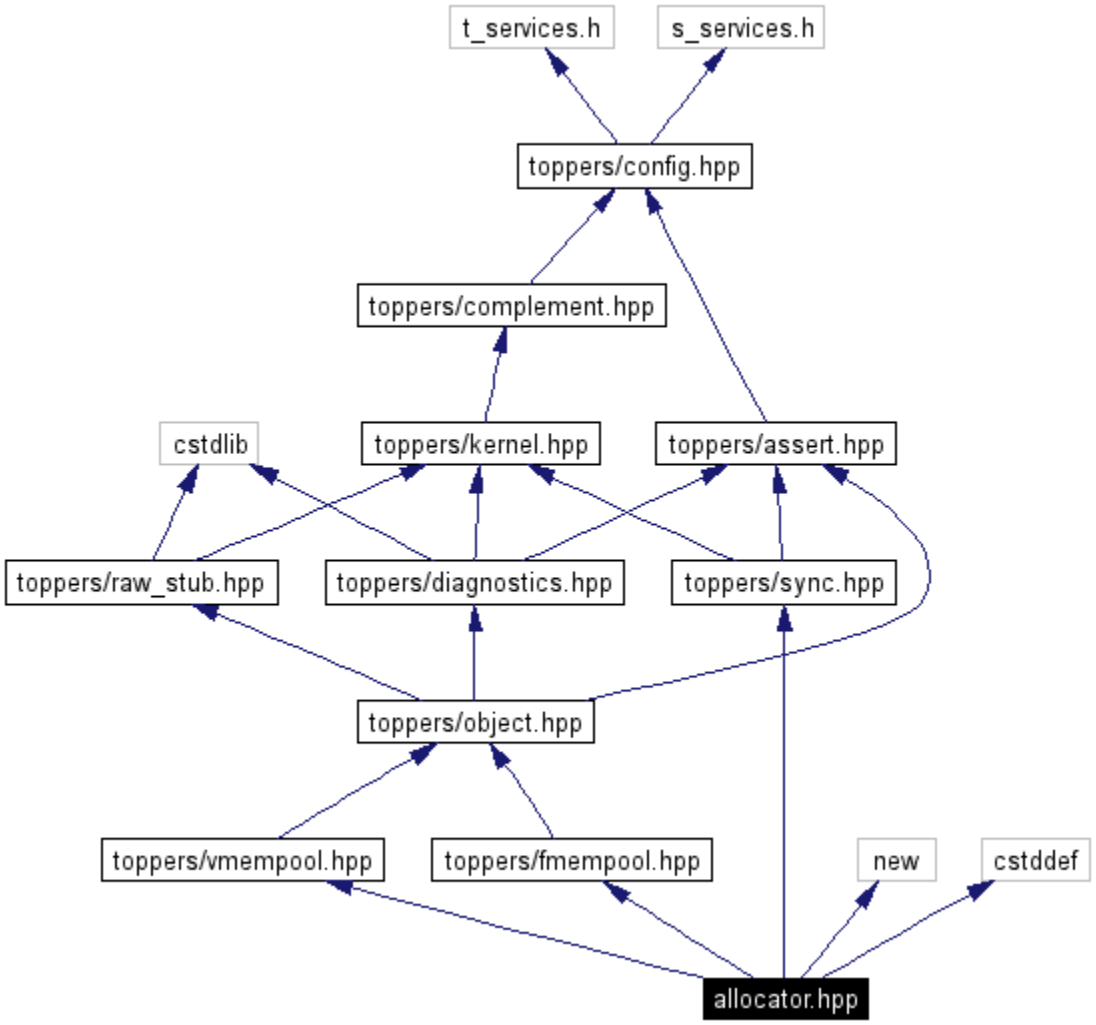
```
class alarmhandler<Id, Diagnostics, Stub>;
class alarmhandler_body<Diagnostics, Stub>;
void alarmhandler_entry<Body>(VP_INT);
```

toppers/allocator.hpp の解説

メモリアロケータに関するクラスの定義　このヘッダファイルでは以下のクラスを定義している。

```
#include <toppers/fmempool.hpp>
#include <toppers/vmempool.hpp>
#include <toppers/sync.hpp>
#include <new>
#include <cstddef>
```

allocator.hppのインクルード依存関係図



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

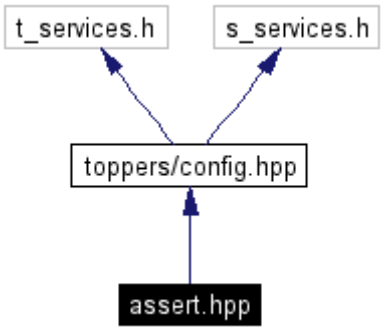
```
    class allocator<T, Pool>;
    class fixed_sized_allocator<Size, Pool>;
    class fixed_sized_new<Host, Pool, Allocator>;
    class variable_sized_new<Host, Pool>;
    class fixed_simple_pool<N, Size>;
    class variable_simple_pool<TotalSize>;
    class fixed_new_pool<Size>;
    class variable_new_pool;
```

toppers/assert.hpp の解説

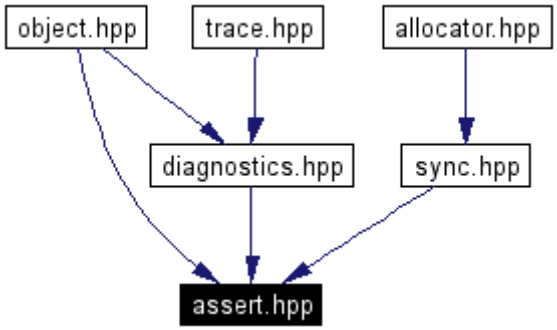
アサーション機能のためのマクロ定義

```
#include <toppers/config.hpp>
```

assert.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**
- namespace **toppers::detail**

マクロ定義

- #define **TOPPERS_ASSERT**(expr) ((void)((expr) ? 0 : ::toppers::detail::assertion_failed(#expr, __FILE__, __LINE__)))
  実行時アサーション

- #define **TOPPERS_STATIC_ASSERT**(expr) typedef int TOPPERS_JOIN(toppers_diagnostics_, __LINE__)[toppers::static_assertion<!!(expr)>::okay]
  静的アサーション

解説

このヘッダファイルでは以下のマクロを定義している。

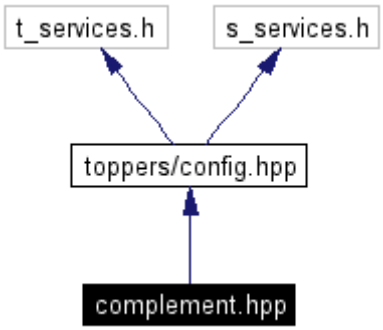
```
TOPPERS_ASSERT(expr);
TOPPERS_STATIC_ASSERT(expr);
```

toppers/complement.hpp の解説

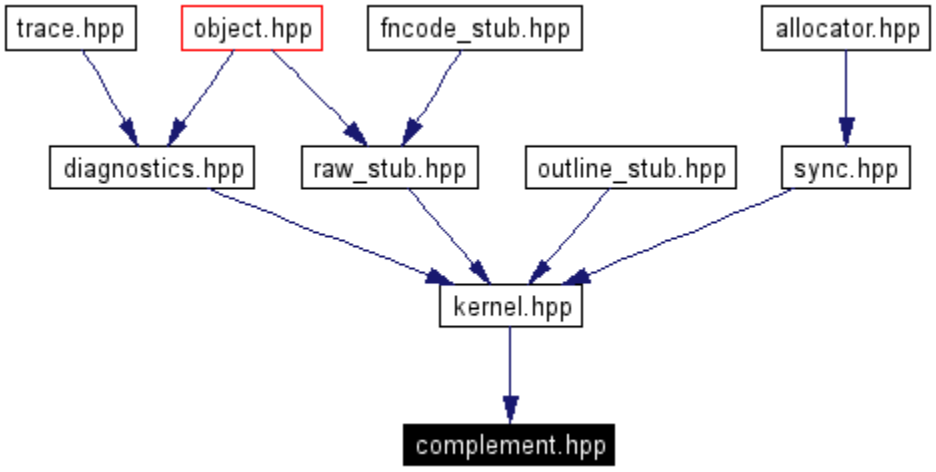
カーネル間の差異を補完・吸収するための宣言・定義

```
#include <toppers/config.hpp>
```

complement.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

- namespace **toppers_private**

解説

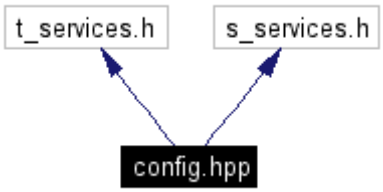
　このヘッダファイルは、拡張サービスコールで必要になるデータ型等、全てのカーネルで 必ずしも定義されているとは限らないものを仮定義することで、カーネル間の差異を吸収し、 ライブラリ内の実装を簡便化することを目的としている。

toppers/config.hpp の解説

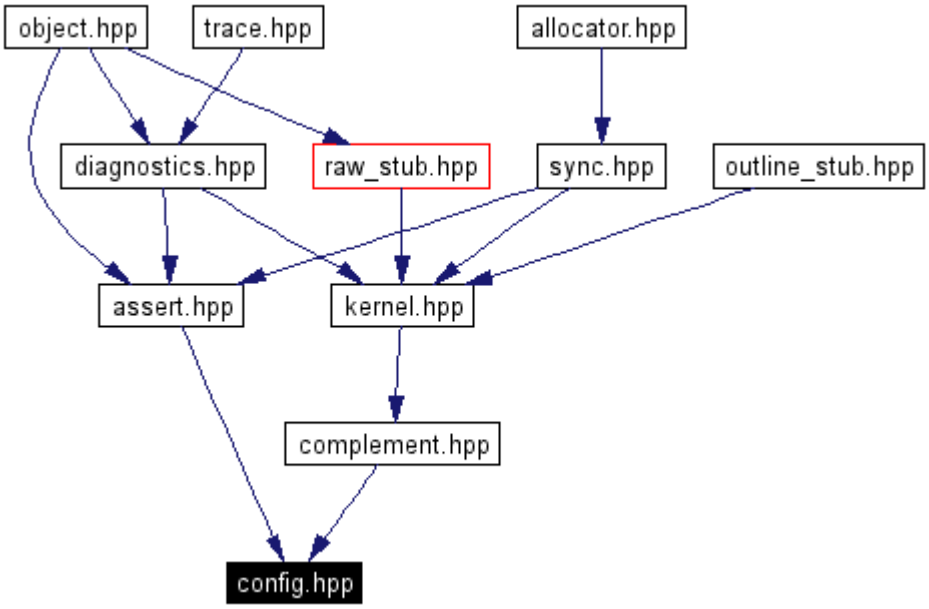
使用するカーネルやコンパイラ等の環境に依存する設定

```
#include <t_services.h>
#include <s_services.h>
```

config.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



解説

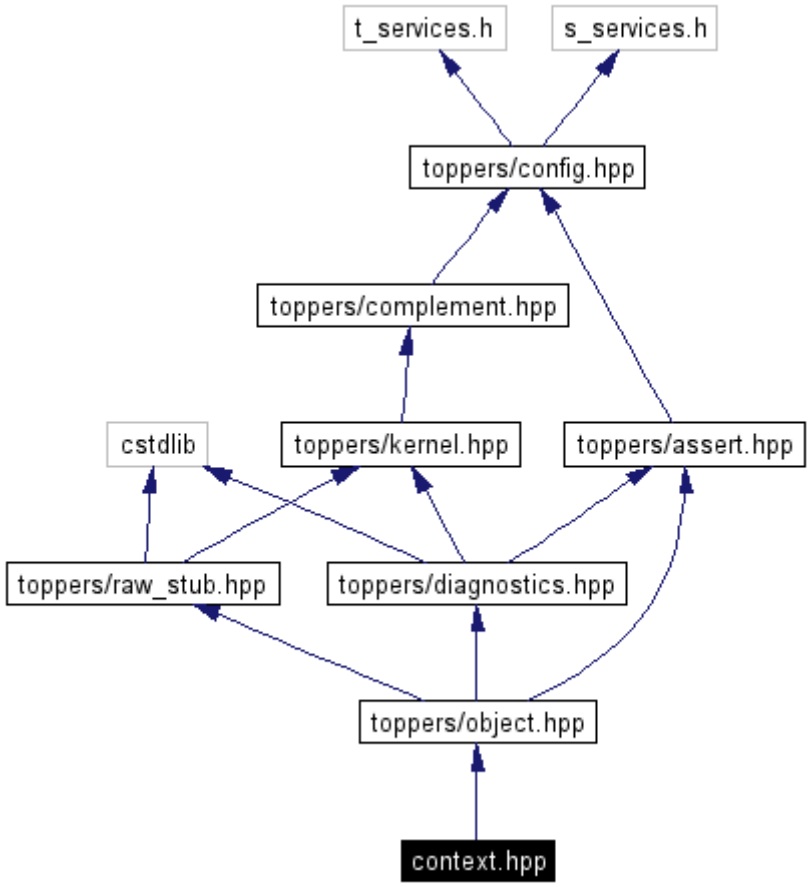
　このヘッダファイルは、使用するカーネルの実装に依存する部分の設定を行う。TOPPERSカーネル以外（またはTOPPERS/JSP 1.4以前の古いTOPPERSカーネル）を使用する場合、 そのカーネルに合わせて適宜このヘッダファイルを修正する必要がある。

toppers/context.hpp の解説

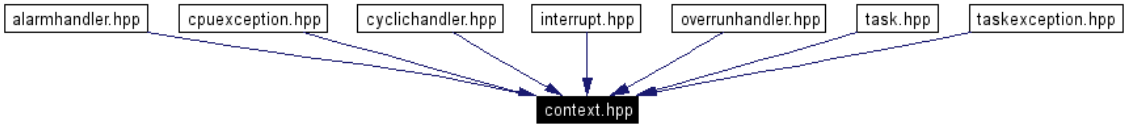
タスクおよび非タスクコンテキストの本体動作に関する基本定義

```
#include <toppers/object.hpp>
```

context.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**
- namespace **toppers::detail**

マクロ定義

- #define **TOPPERS_BIND_HANDLER**(func) ::toppers::detail::bind_context_handler<__typeof__(&(func)), &(func)>(&(func))
  任意型の関数のハンドラ関数への結びつけ

解説

　このヘッダファイルでは以下のテンプレートクラスおよびマクロを定義している。

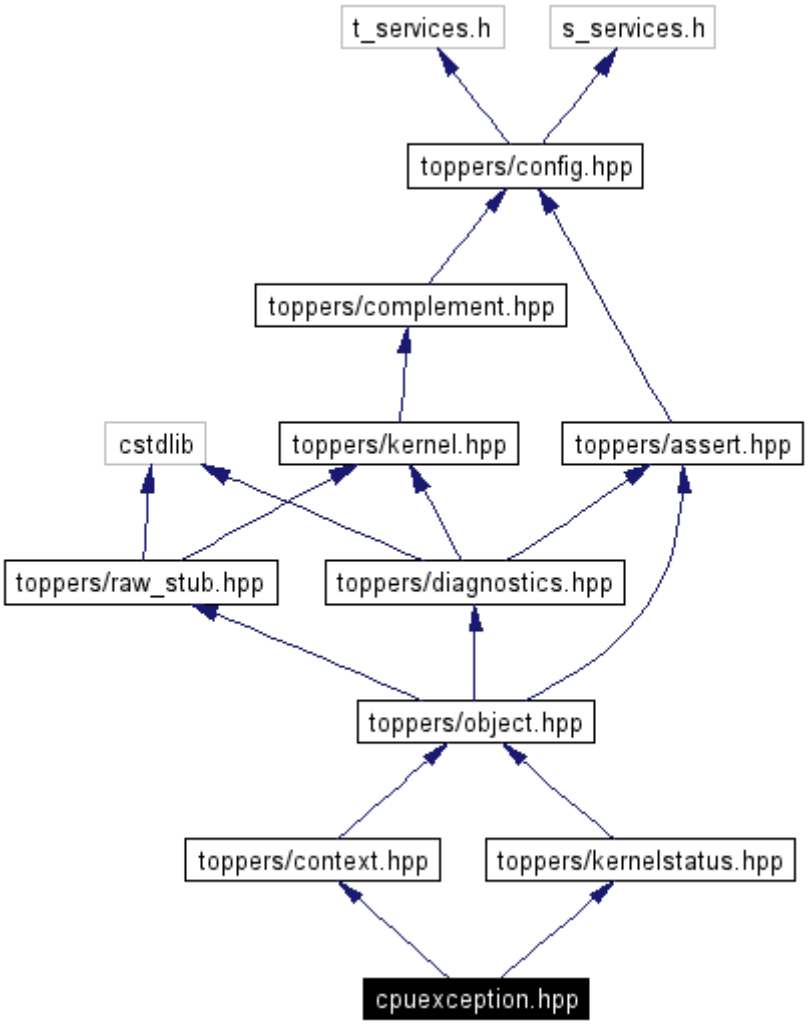
```
class task_context_body<Diagnostics, Stub>;
class non_task_context_body<Diagnostics, Stub>;
TOPPERS_BIND_HANDLER(func);
```

toppers/cpuexception.hpp の解説

CPU例外ハンドラを管理・制御するクラスの定義.

```
#include <toppers/context.hpp>
#include <toppers/kernelstatus.hpp>
```

cpuexception.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

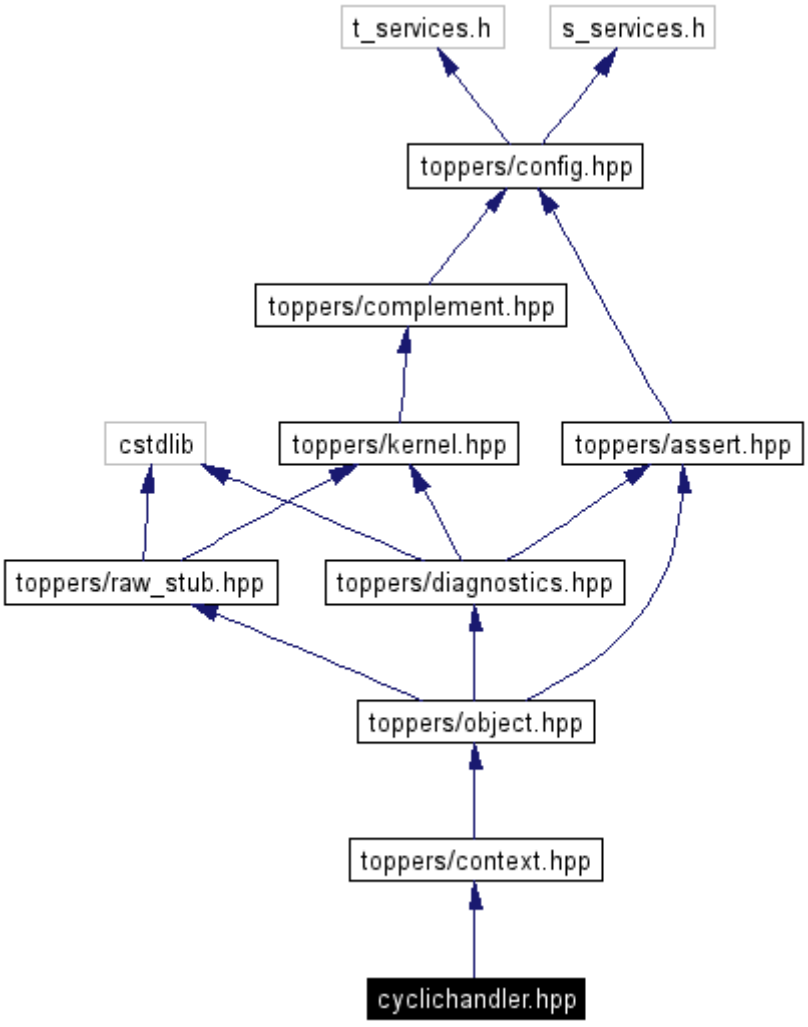
```
class cpuexception<Diagnostics, Stub>;
class cpuexception_body<Diagnostics, Stub>;
void cpuexception_entry<Body>(VP);
```

toppers/cyclichandler.hpp の解説

周期ハンドラを管理・制御するクラスの定義

```
#include <toppers/context.hpp>
```

cyclichandler.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

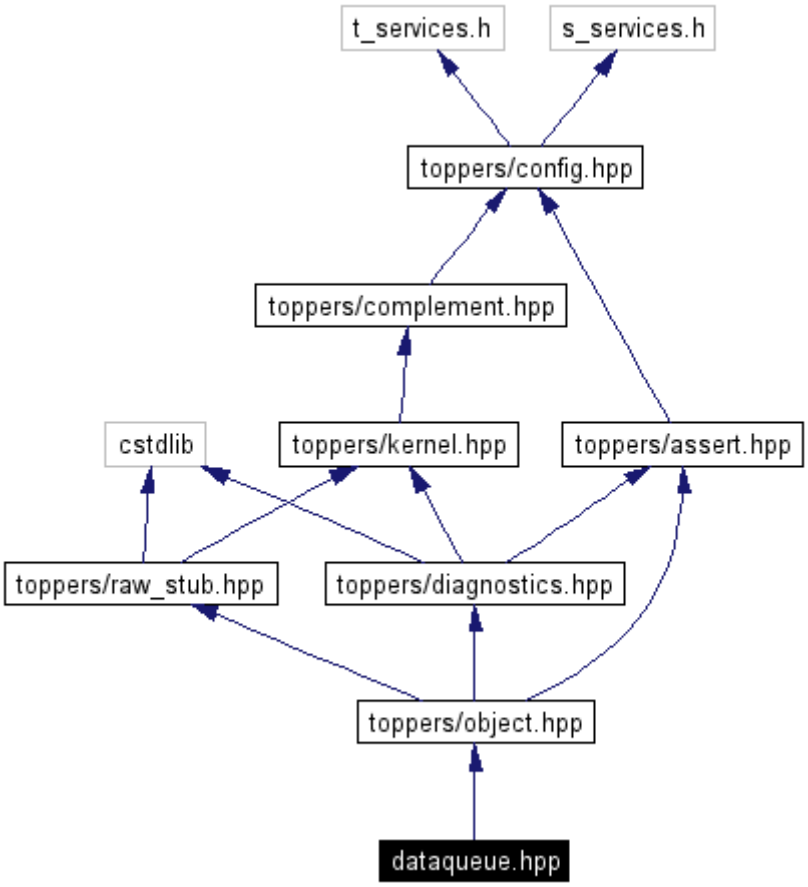
```
    class cyclichandler<Id, Diagnostics, Stub>;
    class cyclichandler_body<Diagnostics, Stub>;
    void cyclichandler_entry<Body>(VP_INT);
```

toppers/dataqueue.hpp の解説

データキューを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

dataqueue.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

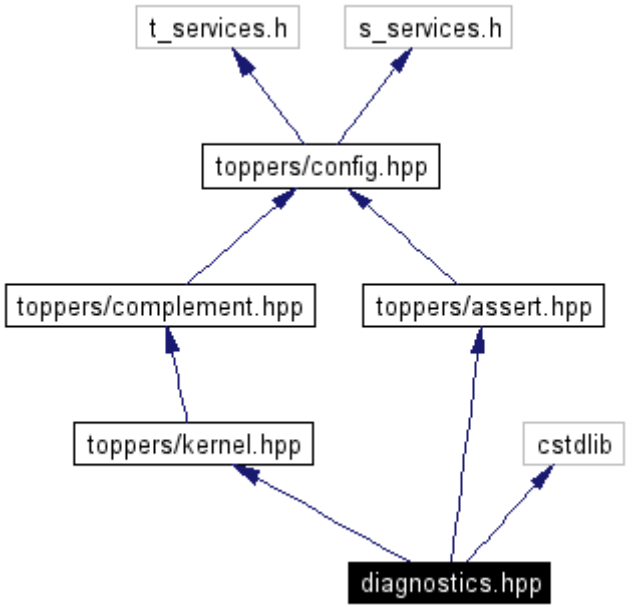
```
   class dataqueue<Id, Diagnostics, Stub>;
```

toppers/diagnostics.hpp の解説

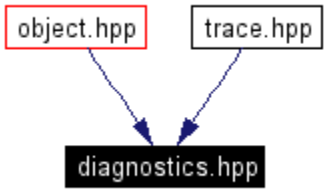
プログラム診断クラスの定義

```
#include <toppers/kernel.hpp>
#include <toppers/assert.hpp>
#include <cstdlib>
```

diagnostics.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

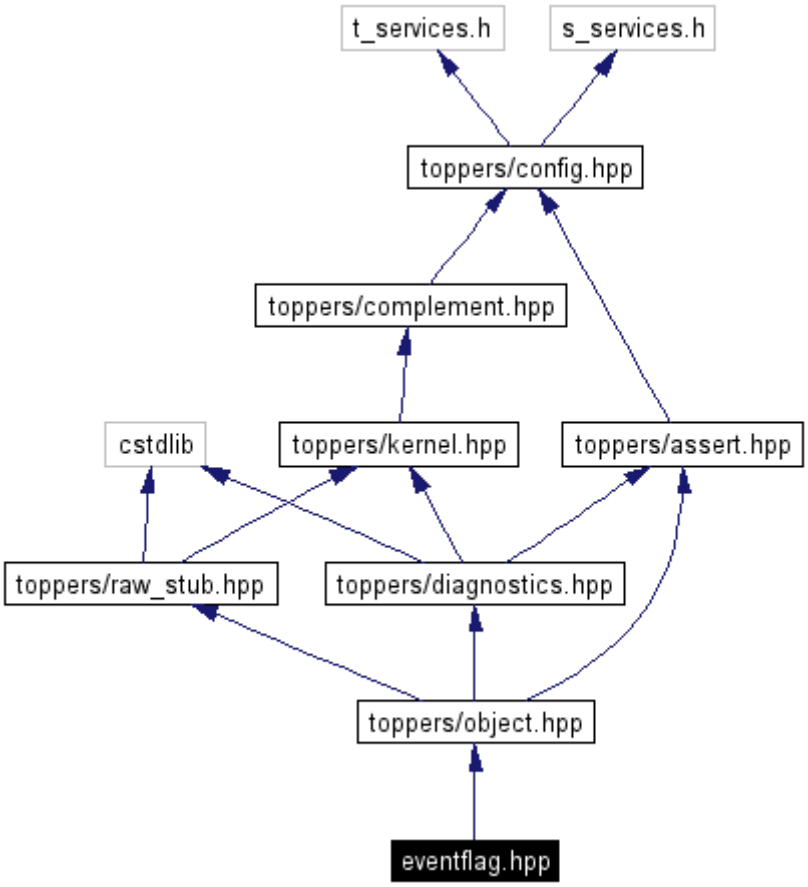
```
class null_diagnostics;
class error_handler_diagnostics<T>;
class polymorphic_diagnostics<T>;
```

toppers/eventflag.hpp の解説

イベントフラグの管理・制御を行うクラスの定義

```
#include <toppers/object.hpp>
```

eventflag.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

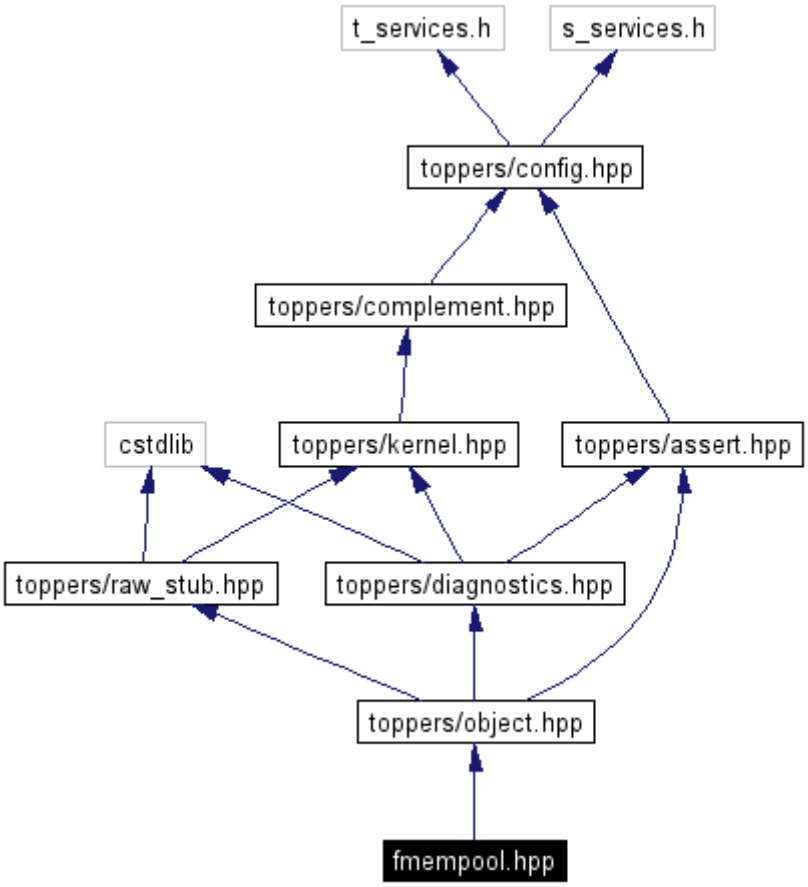
```
class eventflag<Id, Diagnostics, Stub>;
```

toppers/fmempool.hpp の解説

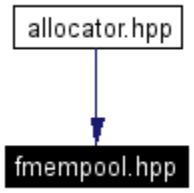
固定長メモリプールを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

fmempool.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

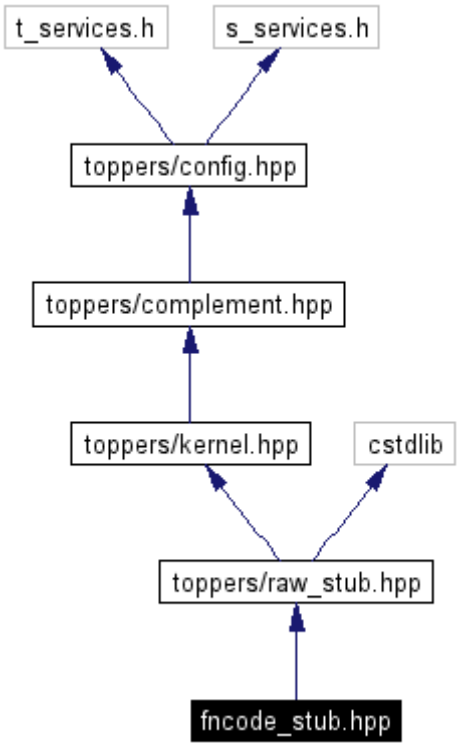
```
    class fmempool<Id, Diagnostics, Stub>;
```

toppers/fncode_stub.hpp の解説

機能コードを用いたサービスコールスタブの定義

```
#include <toppers/raw_stub.hpp>
```

fncode_stub.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のクラスを定義している。

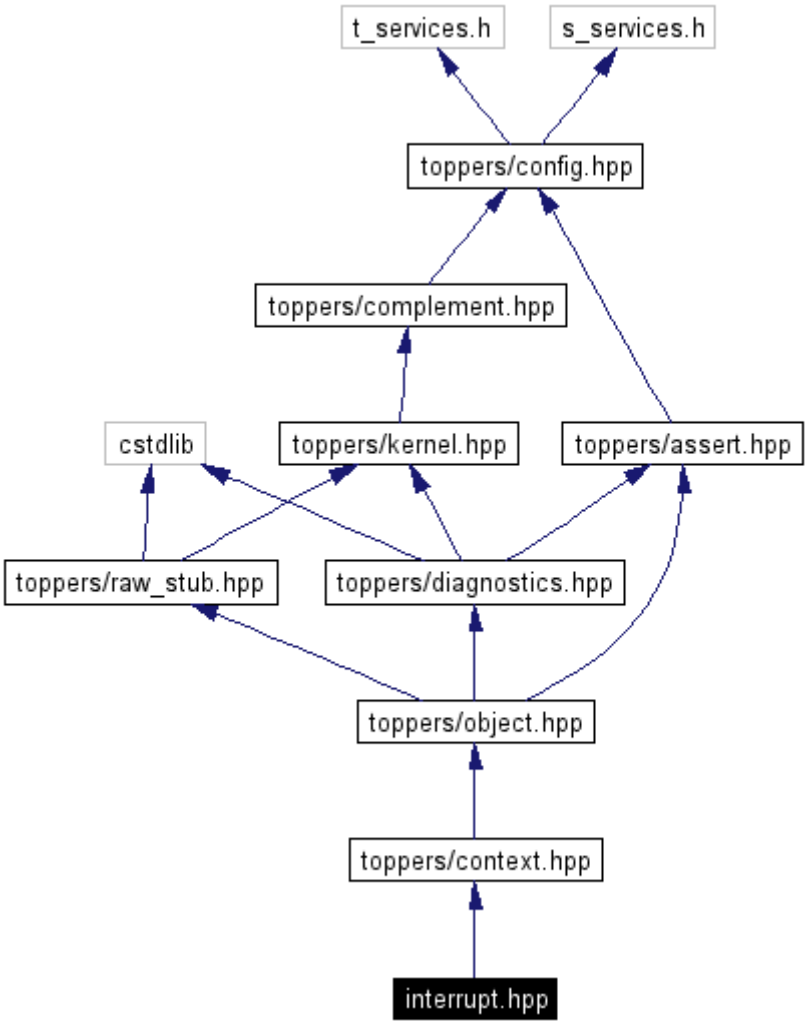
```
class fncode_stub<Stub>;
```

toppers/interrupt.hpp の解説

割込み管理に関するクラスの定義

```
#include <toppers/context.hpp>
```

interrupt.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

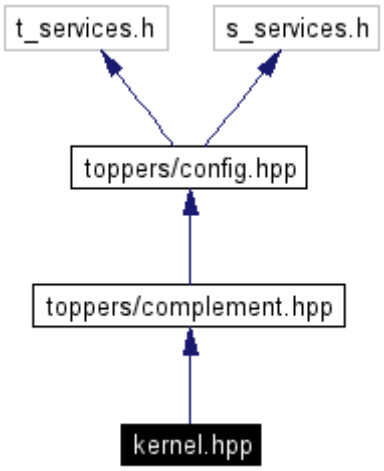
```
class interrupt<Diagnostics, Stub>;
class interrupthandler<Diagnostics, Stub>;
class interrupthandler_body<Diagnostics, Stub>;
void interrupthandler_entry<Body>();
class isr<Id, Diagnostics, Stub>;
class isr_body<Diagnostics, Stub>;
void isr_entry<Body>(VP_INT);
```

toppers/kernel.hpp の解説

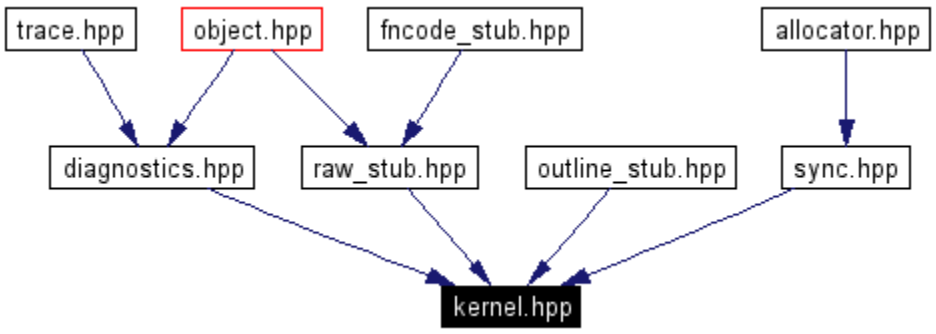
カーネル機能をライブラリから使用するための補助的な定義

```
#include <toppers/complement.hpp>
```

kernel.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下の型およびテンプレート関数を定義している。

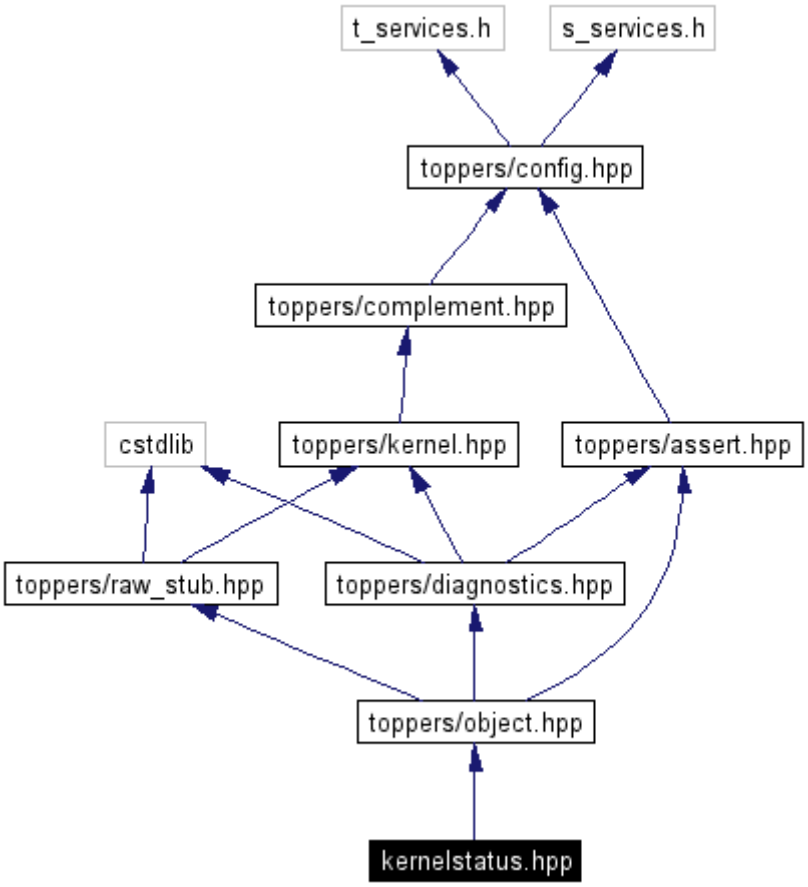
```
enum function_code_type;
enum object_type;
enum task_context_tag
enum non_task_context_tag;
enum context_independent_tag;
enum forever_tag;
enum polling_tag;
enum normal_tag;
enum forced_tag;
struct has_svc<Fncd>;
struct has_obj<ObjType>;
bool sense_kernel<Sense>();
```

toppers/kernelstatus.hpp の解説

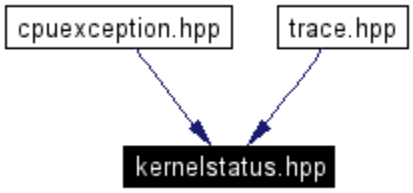
カーネル状態を管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

kernelstatus.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

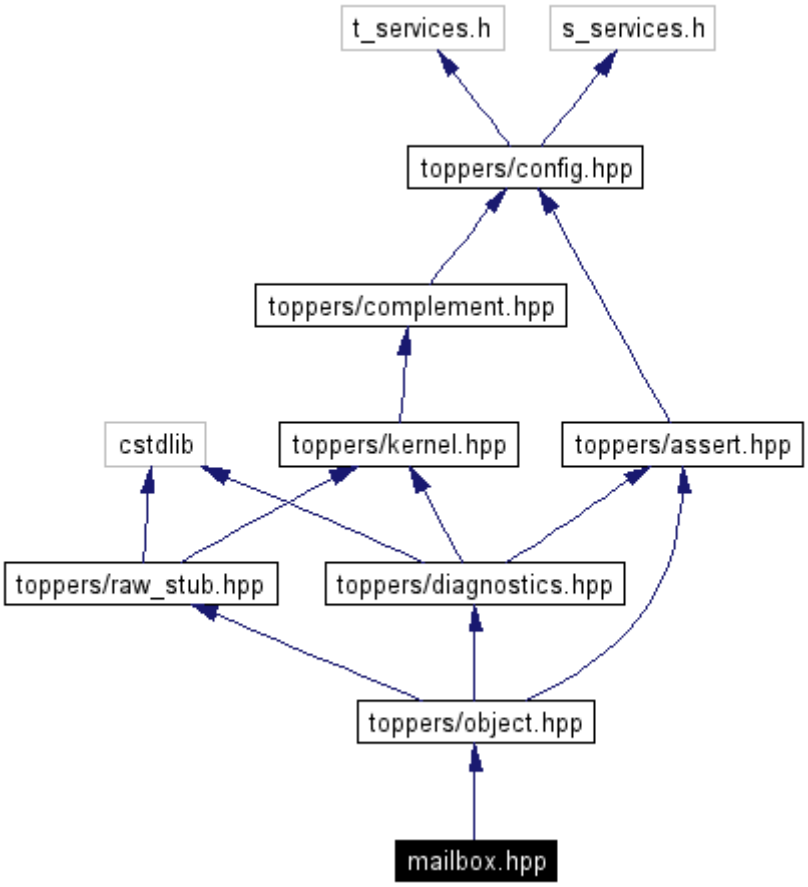
```
class kernelstatus<Diagnostics, Stub>;
class readyqueue<Diagnostics, Stub>;
class cpulock<Diagnostics, Stub>;
class dispatcher<Diagnostics, Stub>;
```

toppers/mailbox.hpp の解説

メールボックスを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

mailbox.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

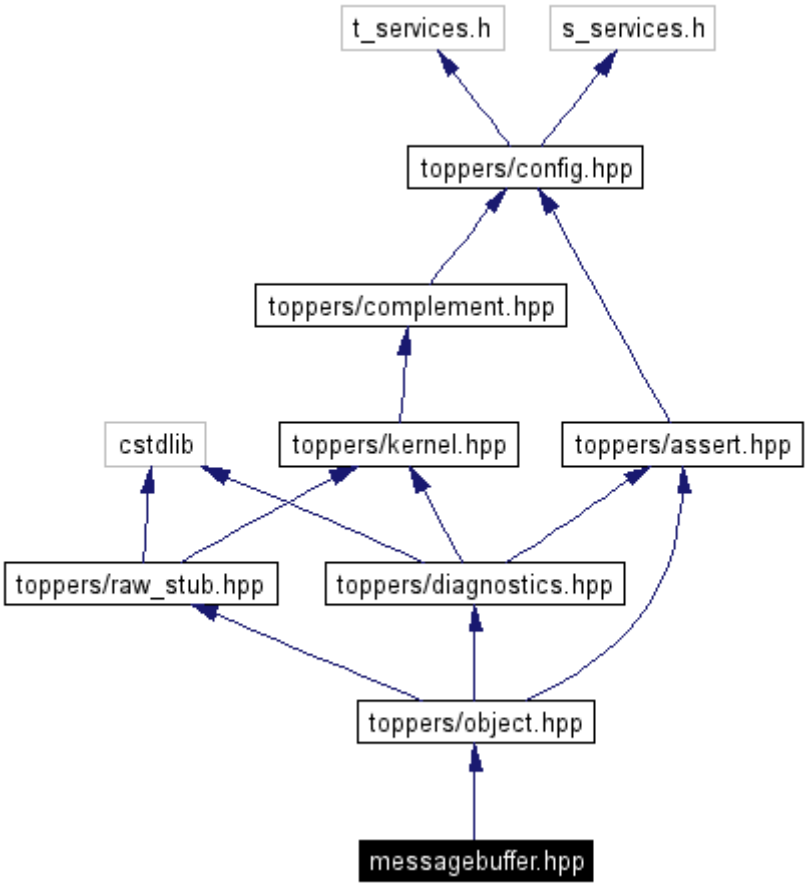
```
   class mailbox<Id, Diagnostics, Stub>;
```

toppers/messagebuffer.hpp の解説

メッセージバッファを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

messagebuffer.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

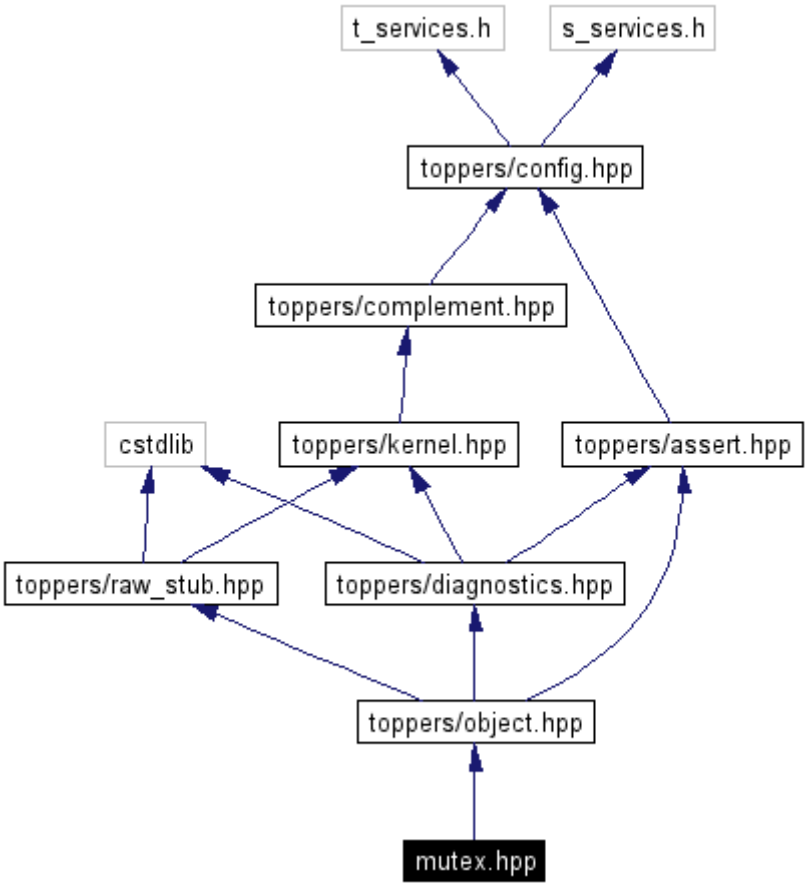
```
class messagebuffer<Id, Diagnostics, Stub>;
```

toppers/mutex.hpp の解説

ミューテックスを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

mutex.hppのインクルード依存関係図



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

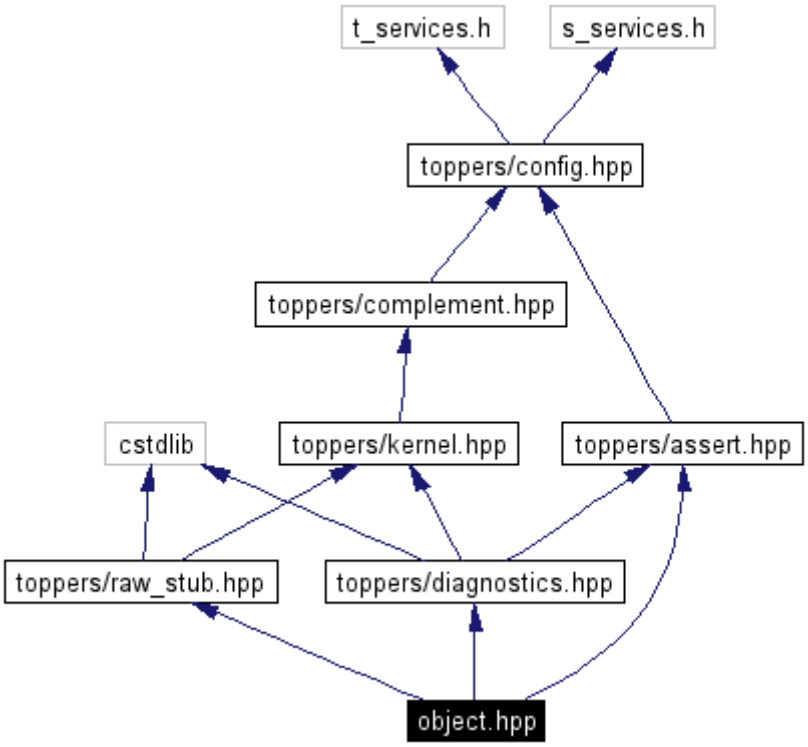
```
class mutex<Id, Diagnostics, Stub>;
```

toppers/object.hpp の解説

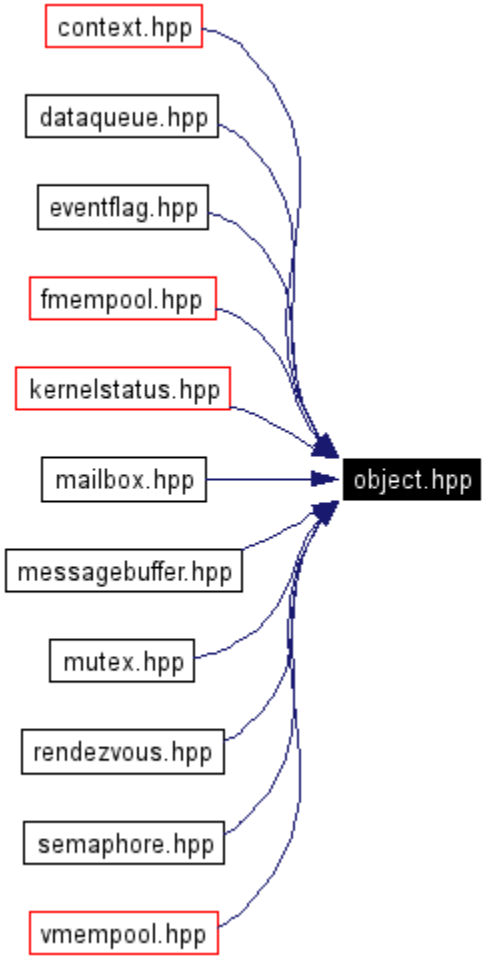
各カーネルオブジェクトで共通に利用するクラス定義

```
#include <toppers/raw_stub.hpp>
#include <toppers/assert.hpp>
#include <toppers/diagnostics.hpp>
```

object.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

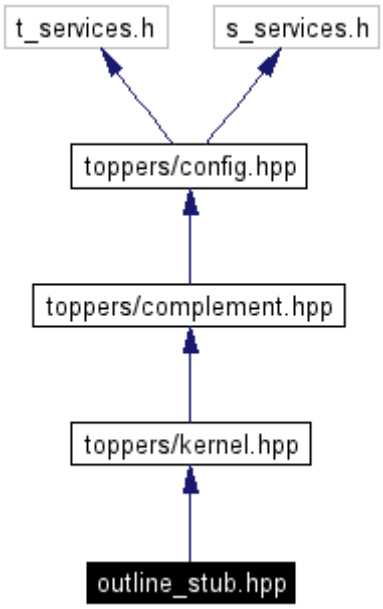
```
class object_id<Id>;
class object_controller<Id, Object, Stub>;
```

toppers/outline_stub.hpp の解説

外部関数を用いたサービスコールスタブの定義

```
#include <toppers/kernel.hpp>
```

outline_stub.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のクラスを定義している。

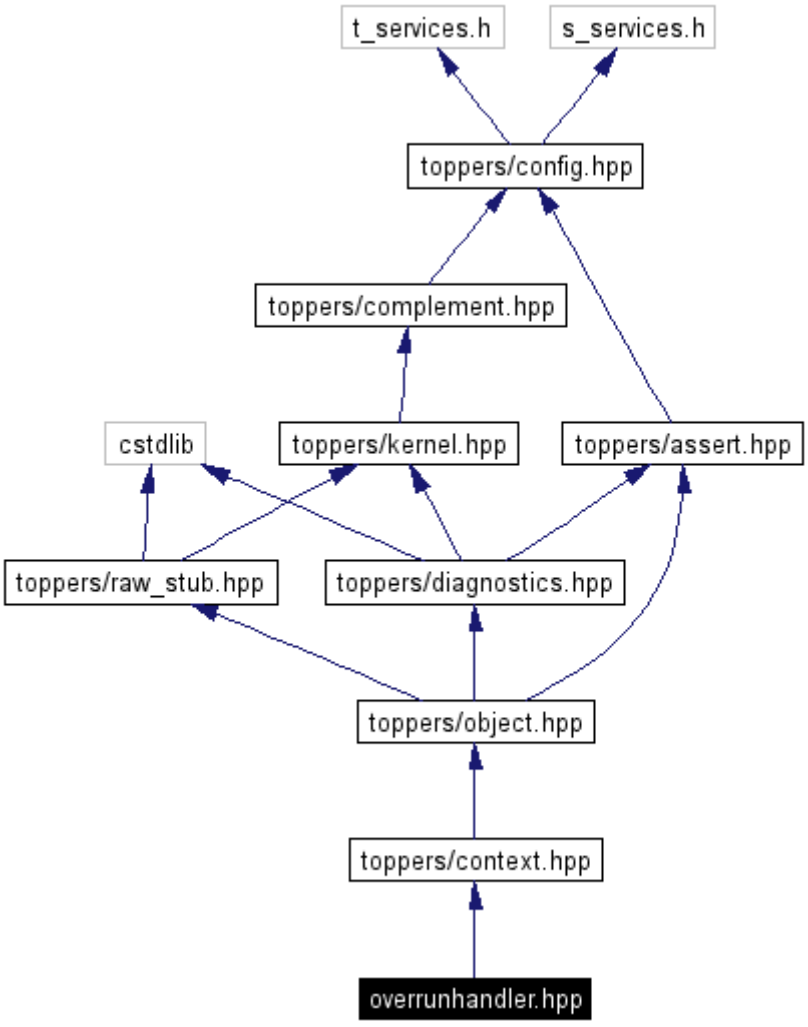
```
class outline_stub;
```

toppers/overrunhandler.hpp の解説

オーバーランハンドラを管理・制御するクラスの定義

```
#include <toppers/context.hpp>
```

overrunhandler.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

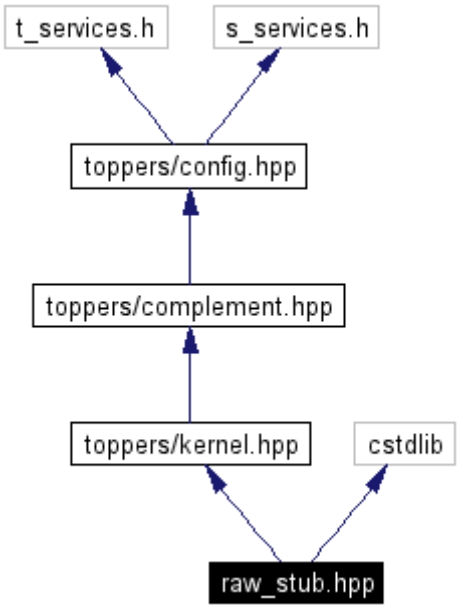
```
class overrunhandler<Diagnostics, Stub>;
class overrunhandler_body<Diagnostics, Stub>;
void overrunhandler_entry<Body>(ID, VP_INT);
```

toppers/raw_stub.hpp の解説

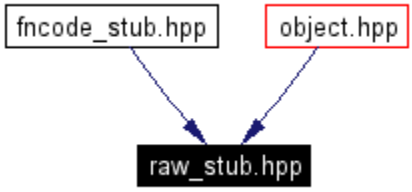
最も単純なサービスコールスタブの定義

```
#include <toppers/kernel.hpp>
#include <cstdlib>
```

raw_stub.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

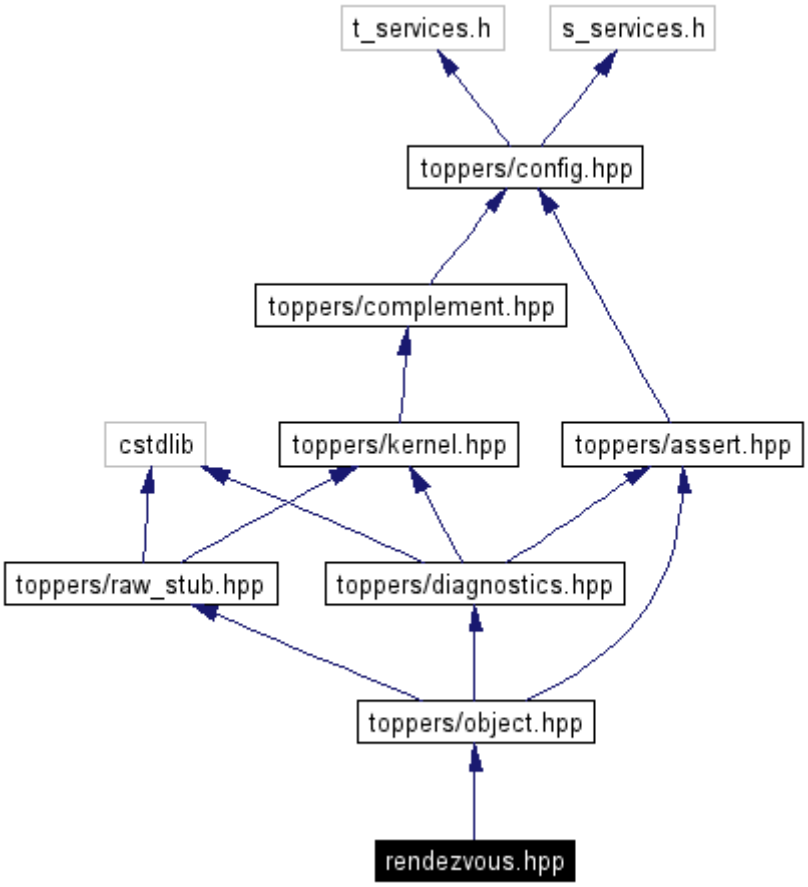
```
class raw_stub<Exist>;
```

toppers/rendezvous.hpp の解説

ランデブを管理・制御するためのクラスの定義

```
#include <toppers/object.hpp>
```

rendezvous.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

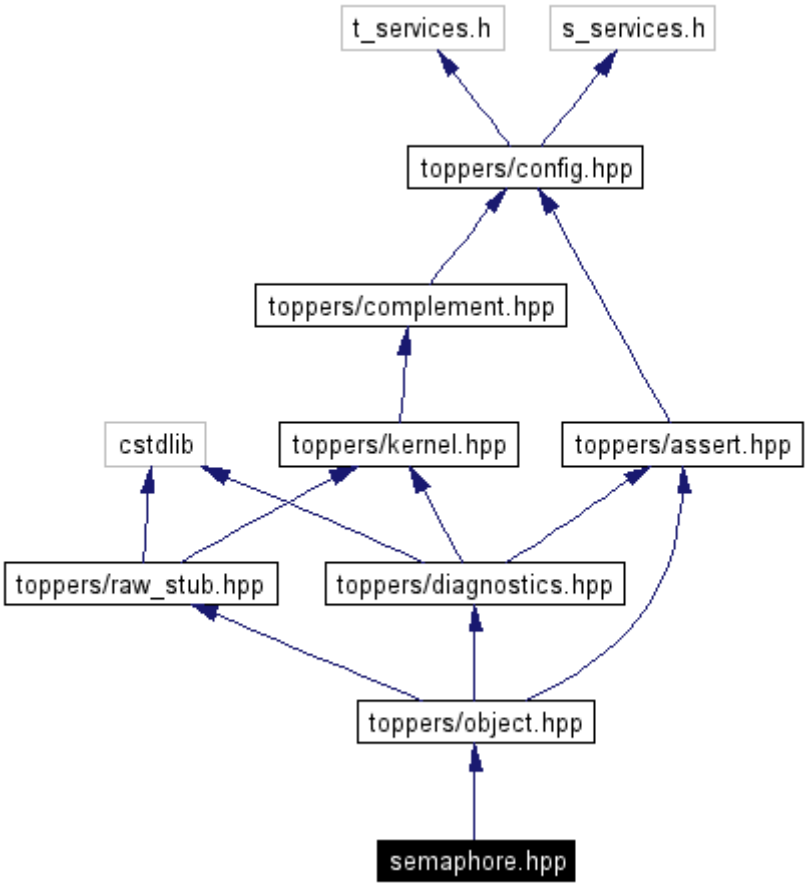
```
   class rendezvous<Diagnostics, Stub>;
   class rendezvousport<Id, Diagnostics, Stub>;
```

toppers/semaphore.hpp の解説

セマフォを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

semaphore.hppのインクルード依存関係図



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

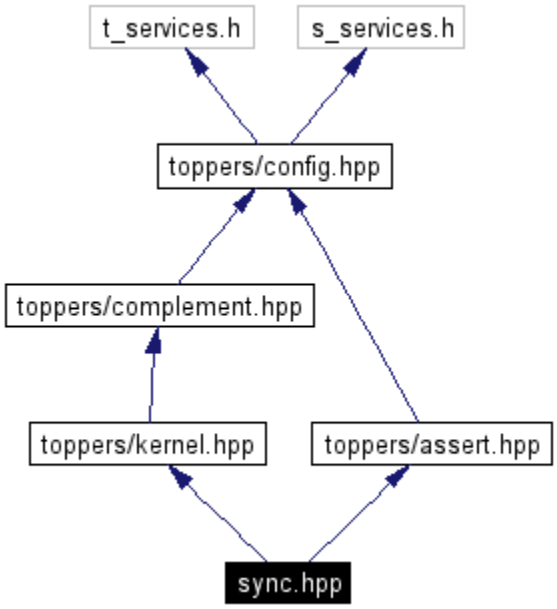
```
class semaphore<Id, Diagnostics, Stub>;
```

toppers/sync.hpp の解説

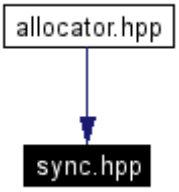
同期機構に関するクラスの定義

```
#include <toppers/kernel.hpp>
#include <toppers/assert.hpp>
```

sync.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**
- namespace **toppers::detail**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

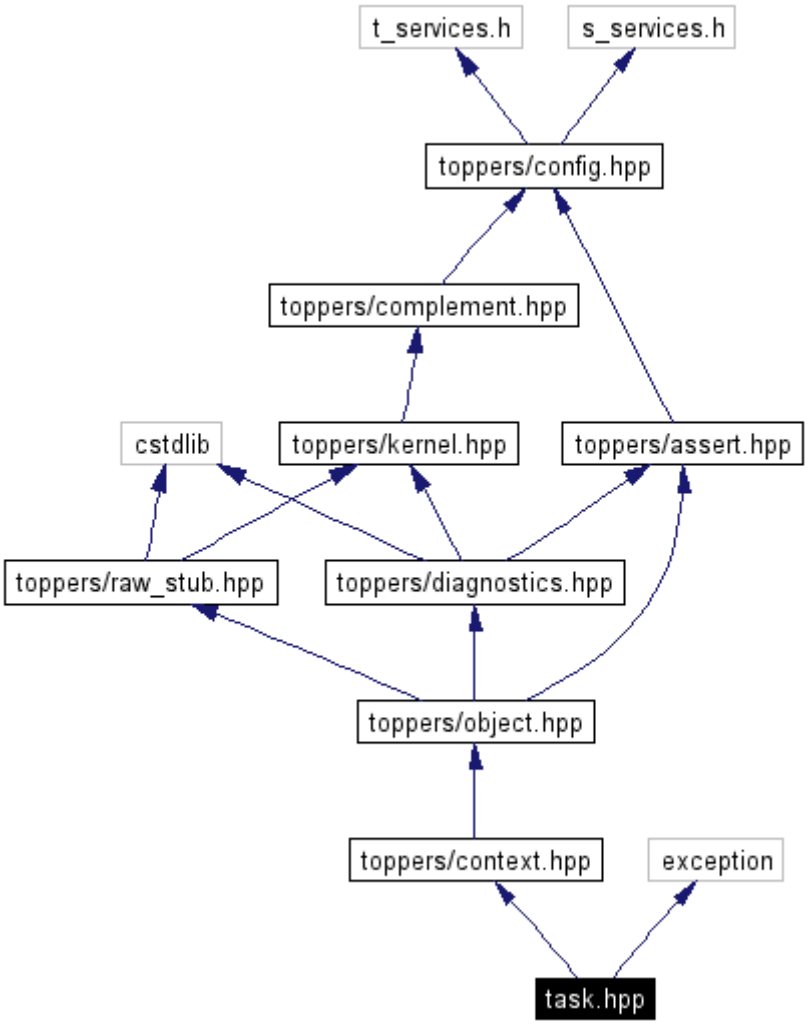
```
class lock<LockObj, SenseKernel>;
class recursive_lock<LockObj, SenseKernel>;
T& sync_increment<LockObj, T>(T&);
T& sync_decrement<LockObj, T>(T&);
void sync_swap<LockObj, T>(T&, T&);
```

toppers/task.hpp の解説

タスクを管理・制御するためのクラス定義

```
#include <toppers/context.hpp>
#include <exception>
```

task.hppのインクルード依存関係図



名前空間

- namespace **toppers**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

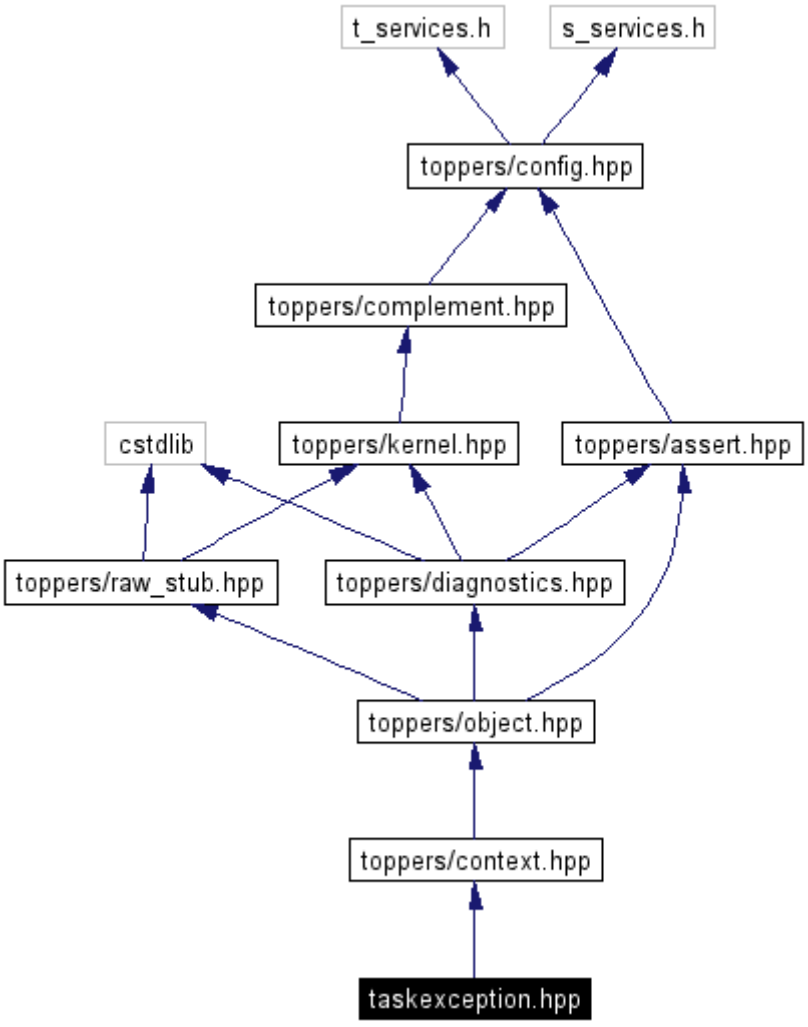
```
   class task<Id, Diagnostics, Stub>;
   class task_body<Diagnostics, Stub>;
   void task_entry<Body>(VP_INT);
```

toppers/taskexception.hpp の解説

タスク例外処理を管理・制御するクラスの定義

```
#include <toppers/context.hpp>
```

taskexception.hppのインクルード依存関係図



名前空間

- namespace **toppers**

- namespace **toppers::detail**

解説

　このヘッダファイルでは以下のテンプレートクラスおよびテンプレート関数を定義している。

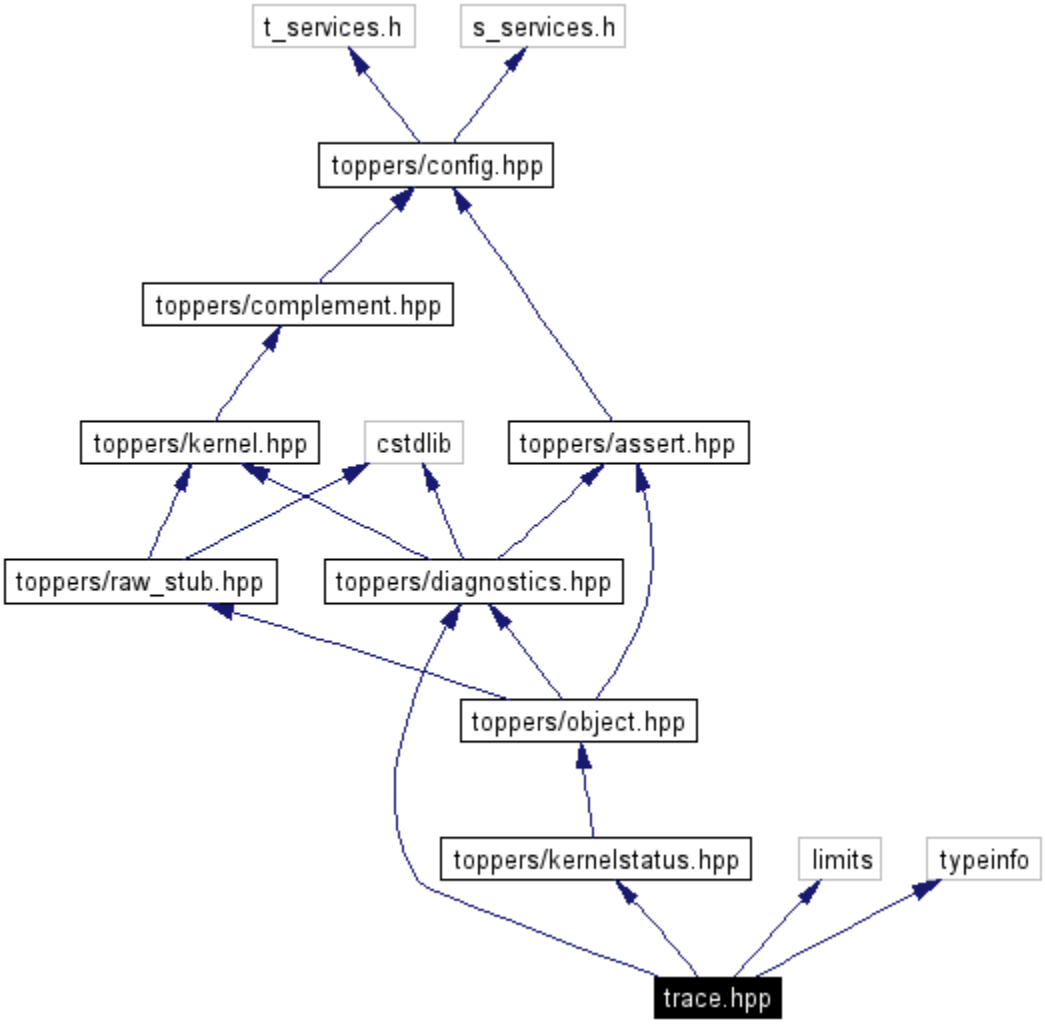
```
class taskexception<Diagnostics, Stub>;
class taskexception_body<Diagnostics, Stub>;
void taskexception_entry<Body>(TEXPTN, VP_INT);
```

toppers/trace.hpp の解説

実行時トレースに関する宣言・定義

```
#include <toppers/diagnostics.hpp>
#include <toppers/kernelstatus.hpp>
#include <limits>
#include <typeinfo>
```

trace.hppのインクルード依存関係図



名前空間

- namespace **toppers**
- namespace **toppers::detail**

マクロ定義

- #define **TOPPERS_TYPENAME**(type)　toppers::type_name<__typeof__(type)>::get()
  型名を表す文字列定数の取得

解説

　このヘッダファイルでは以下のテンプレートクラス、テンプレート関数およびマクロを定義している。

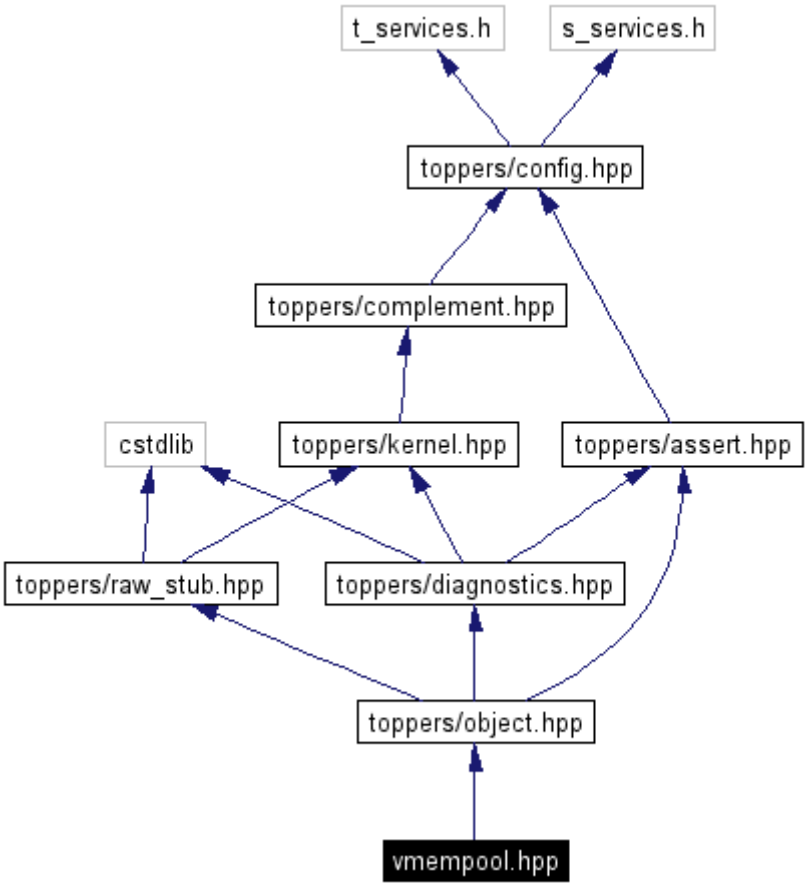
```
class null_tracer;
class syslog_tracer<Count, BufferSize, Diagnostics>;
Tracer& operator<<(Tracer&, T);
TOPPERS_TYPENAME(type);
```

toppers/vmempool.hpp の解説

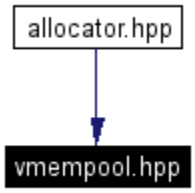
可変長メモリプールを管理・制御するクラスの定義

```
#include <toppers/object.hpp>
```

vmempool.hppのインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



名前空間

- namespace **toppers**

解説

このヘッダファイルでは以下のテンプレートクラスを定義している。

```
    class vmempool<Id, Diagnostics, Stub>;
```

TOPPERS C++ Wrapper Library ページの解説

ライセンス

```
TOPPERS/C++ Binding
    Toyohashi Open Platform for Embedded Real-Time Systems/
    C++ Binding

Copyright (C) 2004 by TAKAGI Nobuhisa

上記著作権者は，以下の (1)～(4) の条件か，Free Software Foundation
によって公表されている GNU General Public License の Version 2 に記
述されている条件を満たす場合に限り，本ソフトウェア（本ソフトウェア
を改変したものを含む．以下同じ）を使用・複製・改変・再配布（以下，
利用と呼ぶ）することを無償で許諾する．
(1) 本ソフトウェアをソースコードの形で利用する場合には，上記の著作
    権表示，この利用条件および下記の無保証規定が，そのままの形でソー
    スコード中に含まれていること．
(2) 本ソフトウェアを，ライブラリ形式など，他のソフトウェア開発に使
    用できる形で再配布する場合には，再配布に伴うドキュメント（利用
    者マニュアルなど）に，上記の著作権表示，この利用条件および下記
    の無保証規定を掲載すること．
(3) 本ソフトウェアを，機器に組み込むなど，他のソフトウェア開発に使
    用できない形で再配布する場合には，次のいずれかの条件を満たすこ
    と．
  (a) 再配布に伴うドキュメント（利用者マニュアルなど）に，上記の著
      作権表示，この利用条件および下記の無保証規定を掲載すること．
  (b) 再配布の形態を，別に定める方法によって，TOPPERSプロジェクトに
      報告すること．
(4) 本ソフトウェアの利用により直接的または間接的に生じるいかなる損
    害からも，上記著作権者およびTOPPERSプロジェクトを免責すること．

本ソフトウェアは，無保証で提供されているものである．上記著作権者お
よびTOPPERSプロジェクトは，本ソフトウェアに関して，その適用可能性も
含めて，いかなる保証も行わない．また，本ソフトウェアの利用により直
接的または間接的に生じたいかなる損害に関しても，その責任を負わない．
```

コーディング規約

`TOPPERS C++ Wrapper Library' は、下記のコーディング規約に基づいて 実装されている。

ソースファイルの命名:

- サフィックス（拡張子）は、ヘッダファイルは *`.hpp'*、*それ以外は　`.cpp' とする。*
- 原則として小文字のみを使用する。
- 原則としてソースファイル内で定義される代表的なクラス名をファイル名とする。

識別子の命名:

- C++の予約識別子は使用しない。
- マクロ名は大文字で、単語の区切りは下線とし、プレフィックス *`TOPPERS_'* を付加する。
- テンプレート仮引数は単語の頭文字を大文字、それ以外は小文字とする。
- 局所的かつユーザが使用することがないものについては、下線一文字の識別子を用いてもよい。
- 上記以外の識別子は小文字で、単語の区切りは下線とする。
- ライブラリが、実装の都合上、グローバル名前空間に配置する一般識別子は *`toppers_'* を付加する。
- ライブラリが、実装の都合上、内部的に使用しているマクロ名は下線で終わる。
- クラスのメンバ変数は下線で終わる。
- 特別な理由がない限り、識別子に用いる単語は省略しない。

名前空間:

- ライブラリの各要素は、原則として*`toppers'* 名前空間内に配置する。
- 実装の都合上、*`toppers'* 名前空間に配置できないものについては、*`toppers_private'* 名前空間に配置し、グローバル名前空間には極力配置しない。
- 実装の詳細に関する要素は、*`toppers::detail'* 名前空間に配置する。

チュートリアル

このページは`TOPPERS C++ Wrapper Library'を導入するための手引きとなるチュートリアルである。

1. **環境設定**
2. **Hello, World!**

環境設定

TOPPERSカーネルでの使用

`TOPPERS C++ Wrapper Library'は、通常の使用では、C++のソースファイルからヘッダファイルをインクルードする だけで使用することができる。 そのため、必要な環境設定は wrapper ディレクトリにインクルードパスを通すだけでよい。

TOPPERSカーネル以外、または旧バージョンのTOPPERSカーネルでの使用

最新のTOPPERSカーネル以外を使用する場合には何らかの移植作業が必要になる。 本ライブラリは、そうしたカーネルをサポートしていない。

1. src ディレクトリ下のソースをメイクして libcxxwrap.a を構築する。
2. toppers/config.hpp を適切に編集し、カーネルの実装に依存するマクロ定義を行う。
3. 必要に応じて、各ソースコードの修正を行う。

Hello, World!

```
/*  hello.cfg   */
INCLUDE(¥"hello.h¥");

CRE_TSK(HELLO_TASK, { TA_HLNG | TA_ACT, 0, hello_task, 5, 0x1000, NULL });

/* 以下、カーネル依存の設定 */
```

```
/*  hello.h */
#ifdef  __cplusplus
extern "C" {
#endif

void hello_task(VP_INT);

#ifdef  __cplusplus
}
#endif
```

```
/*  hello.cpp   */
#include <toppers/task.hpp>
#include <toppers/trace.hpp>
#include "hello.h"
#include "kernel_id.h"

class hello_body : public topper::task_body<>
{
public:
   explicit hello_body(VP_INT exinf)
      : toppers_task_body<>(exinf)
   {
   }

   void run()
   {
      using namespace topper;
      syslog_tracer<> tr;

      tr << "Hello, World!" << endl;
   }
}

void hello_task(VP_INT exinf)
```

```
{
   toppers::task_entry<hello_body>(exinf);
}
```

TODO一覧

メンバ TOPPERS_BIND_HANDLER (func)

将来のバージョンでは可能な限りGCC以外のコンパイラにも対応させる。

索引

~fixed_sized_new
  toppers::fixed_sized_new, 70
~variable_sized_new
  toppers::variable_sized_new, 197
alarmhandler_entry
  KernelObjectManager, 11
context_independent
  toppers, 31
context_independent_tag
  toppers, 31
cpuexception_entry
  KernelObjectManager, 11
cyclichandler_entry
  KernelObjectManager, 11
Diagnostics
  TOPPERS_ASSERT, 16
  TOPPERS_STATIC_ASSERT, 16
exit
  toppers::task_body, 180
exit_and_destroy
  toppers::task_body, 180
filter
  toppers::taskexception_body, 191
forced
  toppers, 31
forced_tag
  toppers, 31
forever
  toppers, 31
forever_tag
  toppers, 31
interrupthandler_entry
  KernelObjectManager, 12
isr_entry
  KernelObjectManager, 12
KernelObjectManager
  alarmhandler_entry, 11
  cpuexception_entry, 11
  cyclichandler_entry, 11
  interrupthandler_entry, 12
  isr_entry, 12
  overrunhandler_entry, 12
  sense_kernel, 13
  task_entry, 13
  taskexception_entry, 13
  TOPPERS_BIND_HANDLER, 10
non_task_context
  toppers, 31
non_task_context_tag
  toppers, 31
normal
  toppers, 32
normal_tag
  toppers, 32
operator new
  toppers::fixed_sized_new, 71
  toppers::variable_sized_new, 197
overrunhandler_entry
  KernelObjectManager, 12
polling
  toppers, 32
polling_tag
  toppers, 32
putter
  toppers::syslog_tracer, 170
sense_kernel
  KernelObjectManager, 13
sync_decrement
  Syncronization, 18
sync_increment
  Syncronization, 18
sync_swap
  Syncronization, 19
Syncronization
  sync_decrement, 18
  sync_increment, 18
  sync_swap, 19
task_context
  toppers, 32
task_context_tag
  toppers, 32
task_entry
  KernelObjectManager, 13
taskexception_entry
  KernelObjectManager, 13
toppers, 20
  context_independent, 31
  context_independent_tag, 31
  forced, 31
  forced_tag, 31
  forever, 31
  forever_tag, 31
  non_task_context, 31
  non_task_context_tag, 31
  normal, 32
  normal_tag, 32

polling, 32
polling_tag, 32
task_context, 32
task_context_tag, 32
toppers/alarmhandler.hpp, 200
toppers/allocator.hpp, 202
toppers/assert.hpp, 204
toppers/complement.hpp, 206
toppers/config.hpp, 208
toppers/context.hpp, 210
toppers/cpuexception.hpp, 212
toppers/cyclichandler.hpp, 214
toppers/dataqueue.hpp, 216
toppers/diagnostics.hpp, 218
toppers/eventflag.hpp, 220
toppers/fmempool.hpp, 222
toppers/fncode_stub.hpp, 224
toppers/interrupt.hpp, 226
toppers/kernel.hpp, 228
toppers/kernelstatus.hpp, 230
toppers/mailbox.hpp, 232
toppers/messagebuffer.hpp, 234
toppers/mutex.hpp, 236
toppers/object.hpp, 238
toppers/outline_stub.hpp, 240
toppers/overrunhandler.hpp, 241
toppers/raw_stub.hpp, 243
toppers/rendezvous.hpp, 245
toppers/semaphore.hpp, 247
toppers/sync.hpp, 249
toppers/task.hpp, 251
toppers/taskexception.hpp, 253
toppers/trace.hpp, 255
toppers/vmempool.hpp, 257
toppers::alarmhandler, 33
toppers::alarmhandler_body, 36
toppers::allocator, 39
toppers::allocator::rebind, 42
toppers::cpuexception, 43
toppers::cpuexception_body, 45
toppers::cpulock, 47
toppers::cyclichandler, 49
toppers::cyclichandler_body, 52
toppers::dataqueue, 55
toppers::dispatcher, 58
toppers::dispatcher::pending, 59
toppers::error_handler_diagnostics, 60
use_exception, 61
toppers::eventflag, 62
toppers::fixed_new_pool, 65
toppers::fixed_simple_pool, 66
toppers::fixed_sized_allocator, 68
toppers::fixed_sized_new, 69
~fixed_sized_new, 70
operator new, 71
toppers::fmempool, 72
toppers::fncode_stub, 74
toppers::has_obj, 89
value, 89
toppers::has_svc, 90
value, 90
toppers::interrupt, 91
toppers::interrupthandler, 92
toppers::interrupthandler_body, 94
toppers::isr, 96
toppers::isr_body, 98
toppers::kernelstatus, 100
toppers::kernelstatus::context, 102
toppers::lock, 103
toppers::mailbox, 105
toppers::messagebuffer, 107
toppers::mutex, 109
toppers::non_task_context_body, 112
toppers::null_diagnostics, 114
use_exception, 115
toppers::null_tracer, 116
toppers::object_controller, 117
toppers::object_id, 118
toppers::outline_stub, 119
toppers::overrunhandler, 135
toppers::overrunhandler_body, 137
toppers::polymorphic_diagnostics, 140
use_exception, 142
toppers::raw_stub, 143
toppers::readyqueue, 158
toppers::recursive_lock, 159
toppers::rendezvous, 161
toppers::rendezvousport, 163
toppers::semaphore, 166
toppers::syslog_tracer, 169
putter, 170
toppers::task, 172
toppers::task_body, 176
exit, 180
exit_and_destroy, 180
toppers::task_body::terminator, 181
toppers::task_context_body, 182
toppers::taskexception, 185
toppers::taskexception_body, 188
filter, 191
toppers::variable_new_pool, 192
toppers::variable_simple_pool, 193
toppers::variable_sized_new, 195
~variable_sized_new, 197

operator new, 197
toppers::vmempool, 198
TOPPERS_ASSERT
Diagnostics, 16
TOPPERS_BIND_HANDLER
KernelObjectManager, 10
TOPPERS_STATIC_ASSERT
Diagnostics, 16
use_exception
toppers::error_handler_diagnostics, 61
toppers::null_diagnostics, 115
toppers::polymorphic_diagnostics, 142
value
toppers::has_obj, 89
toppers::has_svc, 90
カーネルオブジェクト管理, 3
サービスコールスタブ, 2
プログラム診断, 15
メモリアロケータ, 14
同期機構, 17