MATE

1.0

Generated by Doxygen 1.8.5

Fri Jun 22 2018 11:09:16

Contents

Chapter 1

Deprecated List

Member DiProcess::DiProcess ()

Member Model::Application::Start ()

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 ACProxy Class Reference

Creates a connection with AC and acts as an interface with it. This class has a socket object to represent the connection and a PTPProtocol object for making requests in a common language.

`#include <ACProxy.h>`

Public Member Functions

- ACProxy (std::string const &host, int const port)

  Constructor, creates a connection with the given host:port.
- void StartApplication (char const *appPath, int argc, char const **argv, char const *analyzerHost)

  Starts the execution of the application in the AC host.
- void AddInstr (int tid, int eventId, std::string const &fName, InstrPlace place, int nAttrs, Attribute *attrs)

  Requests for adding an instruction in the target application.
- void AddInstr (int tid, int eventId, std::string const &fName, InstrPlace place, int nAttrs, Attribute *attrs, int nPapi, std::string *PapiMetrics)

  Requests for adding an instruction in the target application.
- void RemoveInstr (int tid, int eventId, InstrPlace place)

  Requests for removing an instruction from the target application.
- void LoadLibrary (int tid, std::string const &libPath)

  Requests for loading a library in the target application.
- void SetVariableValue (int tid, std::string const &varName, AttributeValue const &varValue, Breakpoint *brkpt)

  Requests for changing the value of a variable in the target application.
- void ReplaceFunction (int tid, std::string const &oldFunc, std::string const &newFunc, Breakpoint *brkpt)

  Requests for changing all the instances of a function from the target application.
- void InsertFunctionCall (int tid, std::string const &funcName, int nAttrs, Attribute *attrs, std::string const &destFunc, InstrPlace destPlace, Breakpoint *brkpt)

  Requests for the insertion of a function call in a point of the target application.
- void OneTimeFuncCall (int tid, std::string const &funcName, int nAttrs, Attribute *attrs, Breakpoint *brkpt)

  Requests for the call of a function in this point of the target application execution.
- void RemoveFuncCall (int tid, std::string const &funcName, std::string const &callerFunc, Breakpoint *brkpt)

  Requests for removing all the calls to a function in the target application.
- void FuncParamChange (int tid, std::string const &funcName, int paramIdx, int newValue, int *requiredOldValue, Breakpoint *brkpt)

  Requests for the changing of the value of a certain parameter in one function of the target application.

4.1.1 Detailed Description

Creates a connection with AC and acts as an interface with it. This class has a socket object to represent the connection and a PTPProtocol object for making requests in a common language.

Its methods encapsulate the requests in the adequate kind of request object and use the protocol to serialize and write them in the socket.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ACProxy::ACProxy (std::string const & *host,* int const *port*) [inline]

Constructor, creates a connection with the given host:port.

Parameters

host	host were the target AC is running
port	port for the AC process in the host

4.1.3 Member Function Documentation

4.1.3.1 void ACProxy::AddInstr (int *tid,* int *eventId,* std::string const & *fName,* InstrPlace *place,* int *nAttrs,* Attribute * *attrs*)

Requests for adding an instruction in the target application.

Parameters

tid	identifier of the thread in which we will add the instruction
eventId	identifier of the event
fName	name of the function in which we will add the instruction
place	place in the function where we will add the instruction values are: instrUnknown, ipFuncEntry & ipFuncExit
nAttrs	number of Attributes
attrs	Attributes array

4.1.3.2 void ACProxy::AddInstr (int *tid,* int *eventId,* std::string const & *fName,* InstrPlace *place,* int *nAttrs,* Attribute * *attrs,* int *nPapi,* std::string * *PapiMetrics*)

Requests for adding an instruction in the target application.

Parameters

tid	identifier of the thread in which we will add the instruction
eventId	identifier of the event
fName	name of the function in which we will add the instruction
place	place in the function where we will add the instruction values are: instrUnknown, ipFuncEntry & ipFuncExit
nAttrs	number of Attributes
attrs	Attributes array
nPapi	number of Papi Metrics
PapiMetrics	array of Papi Metrics

4.1.3.3 void ACProxy::FuncParamChange (int *tid,* std::string const & *funcName,* int *paramIdx,* int *newValue,* int * *requiredOldValue,* **Breakpoint** * *brkpt*)

Requests for the changing of the value of a certain parameter in one function of the target application.

Parameters

tid	identifier of the thread in which the function is placed
funcName	name of the function
paramIdx	position of the parameter in the parameter list
newValue	new value for the argument
requiredOld-Value	old value required to change for the new one
brkpt	—

4.1.3.4 void ACProxy::InsertFunctionCall (int *tid,* std::string const & *funcName,* int *nAttrs,* **Attribute** * *attrs,* std::string const & *destFunc,* InstrPlace *destPlace,* **Breakpoint** * *brkpt*)

Requests for the insertion of a function call in a point of the target application.

Parameters

tid	identifier of the thread in which the call will be placed
funcName	name of the function
nAttrs	number of attributes
attrs	attributes vector
destFunc	name of the destination function
destPlace	point where the call will be placed
brkpt	—

4.1.3.5 void ACProxy::LoadLibrary (int *tid,* std::string const & *libPath*)

Requests for loading a library in the target application.

Parameters

tid	identifier of the thread in which we will load the library library path

4.1.3.6 void ACProxy::OneTimeFuncCall (int *tid,* std::string const & *funcName,* int *nAttrs,* **Attribute** * *attrs,* **Breakpoint** * *brkpt*)

Requests for the call of a function in this point of the target application execution.

Parameters

tid	identifier of the thread in which the call will be placed
funcName	name of the function
nAttrs	number of attributes
attrs	attributes vector
brkpt	—

4.1.3.7 void ACProxy::RemoveFuncCall (int *tid*, std::string const & *funcName*, std::string const & *callerFunc*, Breakpoint * *brkpt*)

Requests for removing all the calls to a function in the target application.

Parameters

tid	identifier of the thread in which the call will be removed
funcName	name of the function to be removed
callerFunc	function which makes the call
brkpt	—

4.1.3.8 void ACProxy::RemoveInstr (int *tid*, int *eventId*, InstrPlace *place*)

Requests for removing an instruction from the target application.

Parameters

tid	identifier of the thread in which we will remove the instruction
enventId	identifier of the associated event
place	place in the function where the instruction will be removed

4.1.3.9 void ACProxy::ReplaceFunction (int *tid*, std::string const & *oldFunc*, std::string const & *newFunc*, Breakpoint * *brkpt*)

Requests for changing all the instances of a function from the target application.

Parameters

tid	identifier of the thread in which the function is placed
oldFunc	name of the function to be changed
newFunc	name of the new function
brkpt	—

4.1.3.10 void ACProxy::SetVariableValue (int *tid*, std::string const & *varName*, AttributeValue const & *varValue*, Breakpoint * *brkpt*)

Requests for changing the value of a variable in the target application.

Parameters

tid	identifier of the thread in which the variable is placed
varName	name of the variable to change
varValue	new value for the variable

brkpt	—

4.1.3.11 void ACProxy::StartApplication (char const * *appPath*, int *argc*, char const ** *argv*, char const * *analyzerHost*)

Starts the execution of the application in the AC host.

Parameters

appPath	path to the application executable
argc	number of arguments to the application main
argv	argument vector to the application main
analyzerHost	node where the analyzer is executed

The documentation for this class was generated from the following files:

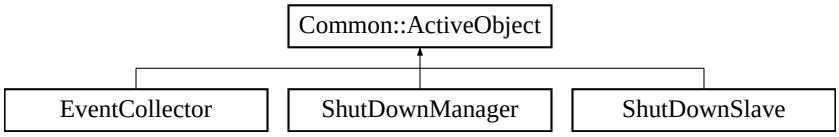
- Analyzer/ACProxy.h
- Analyzer/ACProxy.cpp

4.2 Common::ActiveObject Class Reference

Abstract class, encapsulates OS thread (pthreads) POSIX compatible.

`#include <ActiveObject.h>`

Inheritance diagram for Common::ActiveObject:



Public Member Functions

- ActiveObject ()
  Constructor.
- virtual ~ActiveObject ()
  Destructor.
- void Kill ()
  Stops the thread execution.

Protected Member Functions

- virtual void **InitThread** ()=0
- virtual void **Run** ()=0
- virtual void **FlushThread** ()=0
- void Resume ()
  Continues with the execution of the thread.

Protected Attributes

- int **_isDying**

4.2.1 Detailed Description

Abstract class, encapsulates OS thread (pthreads) POSIX compatible.

Last thing in the constructor of a class derived from ActiveObject must be a call to *_thread.Resume();*

Inside the loop the Run method must keep checking *_isDying*:

```
if (_isDying)
    return;
```

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

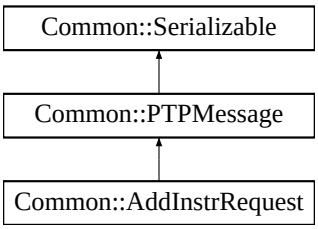
- Common/ActiveObject.h
- Common/ActiveObject.cpp

4.3 Common::AddInstrRequest Class Reference

Represents message sent when analyzer requests to add instrumentation.

`#include <PTPMsg.h>`

Inheritance diagram for Common::AddInstrRequest:



Public Member Functions

- AddInstrRequest (int pid=0, int eventId=0, std::string const &funcName=std::string(), InstrPlace place=ipFuncEntry, int nAttrs=0, Attribute ∗attrs=0, int nPapi=0, std::string ∗PapiMetrics=0)

  Constructor.
- ∼AddInstrRequest ()

  Destructor.
- PTPMsgType GetType () const

  Returns type of message (PTPAddInstr).
- int GetPid () const

  Returns the process id.

- InstrPlace GetInstrPlace () const

  Returns the place where the instruction should be added.
- std::string const & GetFunctionName () const

  Returns function name.
- int GetEventId () const

  Returns the event id.
- Attribute * GetAttributes () const

  Returns array of attributes.
- int GetAttrsCount () const

  Returns number of attributes the function has.
- std::string * GetMetrics () const

  Returns array of Metrics.
- int GetMetricsCount () const

  Returns number of metrics the function has.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

4.3.1 Detailed Description

Represents message sent when analyzer requests to add instrumentation.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Common::AddInstrRequest::AddInstrRequest (int *pid* = 0, int *eventId* = 0, std::string const & *funcName* = std::string(), InstrPlace *place* = ipFuncEntry, int *nAttrs* = 0, Attribute * *attrs* = 0, int *nPapi* = 0, std::string * *PapiMetrics* = 0) [inline]

Constructor.

Parameters

pid	Id of the process where the instrumentation will be added, default 0.
eventId	Event id, default 0.
funcName	Name of the function to modify, default "".
place	Place where the instrumentation will be added, default ipFuncEntry.

nAttrs	Number of attributes the function has, default 0.
attrs	Attribute array, default 0.

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.4 Common::Address Class Reference

Encapsulates a socket address of the AF_INET family.

`#include <Address.h>`

Public Member Functions

- Address (std::string const &host, int port)

  Constructor.
- Address (int port)

  Constructor.
- Address ()

  Constructor.
- operator struct sockaddr * ()

  Returns a pointer to the sockaddr intern structure.
- operator struct sockaddr_in * ()

  Returns a pointer to the sockaddr_in intern structure.
- socklen_t GetSize () const

  Returns size of current address. Number of bytes the address uses in memory.
- std::string GetHostName () const

  Returns name of host.

4.4.1 Detailed Description

Encapsulates a socket address of the AF_INET family.

This class contains methods to initialize the address of a socket.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Common::Address::Address (std::string const & *host*, int *port*)

Constructor.

Parameters

host	Host where the socket will be located.
port	Port used.

Exceptions

SysException	

4.4.2.2 Address::Address (int *port*)

Constructor.

Uses the INADDR_ANY address.

Parameters

port	Port used.

4.4.2.3 Address::Address ( )

Constructor.

Initializes an empty address, setting the memory of the object to 0.

4.4.3 Member Function Documentation

4.4.3.1 string Address::GetHostName ( ) const

Returns name of host.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/Address.h
- Common/Address.cpp

4.5 Analyzer Class Reference

Analyzes events from a given set and determines if there are problems within them. The class also has the capabilities to add or remove instrumentation from the code.

`#include <Analyzer.h>`

Public Member Functions

- Analyzer (EventList &list)

  Constructor.
- void AnalyzeEvent ()

  Analyzes an event and, if it finds a problem, makes tuning actions.
- void Instrument ()

  Requests to add instrumentation in the application so as to get information from it.

- void RemoveInstr ()

  Requests to remove instrumentation.
- void Tune ()

  Requests to modify the application in order to improve its behavior.

4.5.1 Detailed Description

Analyzes events from a given set and determines if there are problems within them. The class also has the capabilities to add or remove instrumentation from the code.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Analyzer::Analyzer (EventList & *list*) [inline]

Constructor.

Parameters

list	of events to be analyzed

The documentation for this class was generated from the following files:

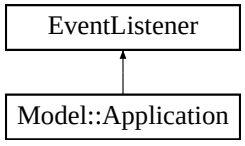
- Analyzer/Analyzer.h
- Analyzer/Analyzer.cpp

4.6 Model::Application Class Reference

Represents tuned application in the analyzer. Holds identificative information of the application, the tasks that form it (and which one is the master), the host where they are running, the places from where we are getting events and handlers to both: tasks and hosts. Provides methods to:

`#include <AppModel.h>`

Inheritance diagram for Model::Application:



Public Member Functions

- Application (char const *appPath, int argc, char const **argv)

  Constructor.
- string GetName () const

  Name getter.
- int NumActiveTasks () const

  Number of tasks getter.
- Tasks & GetTasks ()

  Tasks getter.
- Hosts & GetHosts ()

  Hosts getter.
- Task * GetMasterTask ()

  Master task getter.
- Status GetStatus () const
- void Start ()

  Starts the application.
- int AddEvent (Event const &e)

  Adds a definition of a new event to be traced in all running tasks of the application.
- int RemoveEvent (int eventId, InstrPlace place)

  Removes previously added event from all running tasks.
- int LoadLibrary (string const &libPath)

  Loads a shared library to all running tasks. This enables the Analyzer to load any additional code required for the tuning.
- int SetVariableValue (string const &varName, AttributeValue const &varValue, Breakpoint *brkpt)

  Modifies a value of a specified variable in a given set of tasks.
- int ReplaceFunction (string const &oldFunc, string const &newFunc, Breakpoint *brkpt)

  Replaces all calls to a function with calls to another one in a given set of tasks.
- int InsertFunctionCall (string const &funcName, int nAttrs, Attribute *attrs, string const &destFunc, InstrPlace destPlace, Breakpoint *brkpt)

  Inserts a new function invocation code at a given location in a given set of tasks.
- int OneTimeFuncCall (string const &funcName, int nAttrs, Attribute *attrs, Breakpoint *brkpt)

  Inserts a new function invocation code in a given set of tasks and calls it once.
- int RemoveFuncCall (string const &funcName, string const &callerFunc, Breakpoint *brkpt)

  Removes all calls to a given function from the given caller function in a given set of tasks. For example this method can be used to remove all flush() function calls from a debug() function.
- int FuncParamChange (string const &funcName, int paramIdx, int newValue, int *requiredOldValue, Breakpoint *brkpt)

  Sets the value of an input parameter of a given function in a given set of tasks. This parameter value is modified before the function body is invoked. There exists the possibility to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.
- void SetTaskHandler (TaskHandler &h)

  Installs a callback function that is called when a new task is started or an existing one is terminated.
- void SetHostHandler (HostHandler &h)

  Installs a callback function that is called when a new host is added to the virtual machine or an existing one is removed.
- int ProcessEvents (bool block=true)

  Processes application events (ECP).
- void OnEvent (ECPMessage *msg)

  This method is called in the context of Event Collector thread.
- void OnFatalError ()

  This method is called when fatal EventCollector error occurs. Application changes its status to stAborted.

Protected Member Functions

- void ProcessEvent (ECPMessage *msg)

  Takes the proper actions depending on the kind of message received.
- void DispatchEvent (EventMsg const &msg)

  Finds the sender-task corresponding object and dispatches its event (see task).
- Host & AddHost (string const &name)

  Creates & adds a new host to the host list of the application.
- void AddTask (int pid, int mpiRank, string const &name, Host &h)

  Adds a task to the application list.
- void RemoveTask (int tid)

  Removes a task when an unregistered message is received (see processEvent).

4.6.1 Detailed Description

Represents tuned application in the analyzer. Holds identificative information of the application, the tasks that form it (and which one is the master), the host where they are running, the places from where we are getting events and handlers to both: tasks and hosts. Provides methods to:

- Retrieve application information
- Monitoring: add/remove events to trace.
- Tuning: loading libraries, changing variables & parameter values, adding/removing function calls and calling them explicitly.

Basically the monitoring a tuning methods call to the corresponding methods in AppTask for all the tasks that conform the application.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Application::Application (char const * *appPath,* int *argc,* char const ** *argv*)

Constructor.

Parameters

appPath	Path to the executable of the application.
argc	Number of arguments of the application.
argv	Arguments of the application.

4.6.3 Member Function Documentation

4.6.3.1 int Application::AddEvent (Event const & *e*)

Adds a definition of a new event to be traced in all running tasks of the application.

Parameters

e	Event to be traced.

Returns

Number of tasks where the event tracing was added.

4.6.3.2 Host & Application::AddHost (string const & *name*) [protected]

Creates & adds a new host to the host list of the application.

Parameters

name	Name of the host

Returns

Reference to the created host

4.6.3.3 void Application::AddTask (int *pid*, int *mpiRank*, string const & *name*, Host & *h*) [protected]

Adds a task to the application list.

Parameters

pid	Process identifier of the task.
mpiRank	MPI identifier of the task
name	Process name
host	Host where the task is running

4.6.3.4 void Application::DispatchEvent (EventMsg const & *msg*) [protected]

Finds the sender-task corresponding object and dispatches its event (see task).

Parameters

msg	Message that contains an event request from an AC.

4.6.3.5 int Application::FuncParamChange (string const & *funcName*, int *paramIdx*, int *newValue*, int * *requiredOldValue*, Breakpoint * *brkpt*)

Sets the value of an input parameter of a given function in a given set of tasks. This parameter value is modified before the function body is invoked. There exists the possibility to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

Parameters

funcName	Name of the function
paramIdx	Id of the parameter to change
newValue	New value for the parameter
requiredOld-Value	Required old value of the parameter to change it
brkpt	—

Returns

Number of tasks where the parameter was changed.

4.6.3.6 Hosts& Model::Application::GetHosts ( ) [inline]

Hosts getter.

Returns

A collection of Host objects that form the virtual machines

4.6.3.7 Task* Model::Application::GetMasterTask ( ) [inline]

Master task getter.

Returns

A reference to the master task of the application

4.6.3.8 string Model::Application::GetName ( ) const [inline]

Name getter.

Returns

Name of the running program

4.6.3.9 Status Model::Application::GetStatus ( ) const [inline]

Returns

The application status information

4.6.3.10 Tasks& Model::Application::GetTasks ( ) [inline]

Tasks getter.

Returns

A collection of Task objects

4.6.3.11 int Application::InsertFunctionCall (string const & *funcName*, int *nAttrs*, Attribute * *attrs*, string const & *destFunc*, InstrPlace *destPlace*, Breakpoint * *brkpt*)

Inserts a new function invocation code at a given location in a given set of tasks.

Parameters

funcName	Name of the function to call.
nAttrs	Number of parameters of the function.
attrs	Values for each parameter.
destFunc	Function where the calls will be placed.
destPlace	Point of the function where the calls will be placed.
brkpt	—

Returns

Number of tasks where the function calls were added.

4.6.3.12 int Application::LoadLibrary (string const & *libPath*)

Loads a shared library to all running tasks. This enables the Analyzer to load any additional code required for the tuning.

Parameters

libPath	Path to the library.

Returns

Number of tasks where the library is loaded.

4.6.3.13 int Model::Application::NumActiveTasks ( ) const [inline]

Number of tasks getter.

Returns

Number of tasks actually running

4.6.3.14 int Application::OneTimeFuncCall (string const & *funcName*, int *nAttrs*, Attribute * *attrs*, Breakpoint * *brkpt*)

Inserts a new function invocation code in a given set of tasks and calls it once.

Parameters

funcName	Name of the function to call
nAttrs	Number of arguments of the function
attrs	Values for each argument of the function
brkpt	—

Returns

Number of tasks where the function was called.

4.6.3.15 void Application::OnEvent (ECPMessage * *msg*) [virtual]

This method is called in the context of Event Collector thread.

Parameters

msg	Pointer to a message object that must be deleted by a receiver.

Implements EventListener.

4.6.3.16 void Application::ProcessEvent (ECPMessage * *msg*) [protected]

Takes the proper actions depending on the kind of message received.

- Register: adds the host where the new task was created and creates a task object to represent it.
- Unregister: removes the task from the list of task.
- Event: calls DispatchEvent to handle it.

Parameters

msg	Message that contains a request from an AC.

4.6.3.17 int Application::ProcessEvents (bool *block* = `true`)

Processes application events (ECP).

Parameters

Block	indicates if the function blocks and waits for next event.

Returns

Number of processed events

4.6.3.18 int Application::RemoveEvent (int *eventId*, InstrPlace *place*)

Removes previously added event from all running tasks.

Parameters

eventId	Id of the event
place	Location of the function where the event is recorded

Returns

number of tasks where the event was removed.

4.6.3.19 int Application::RemoveFuncCall (string const & *funcName*, string const & *callerFunc*, Breakpoint * *brkpt*)

Removes all calls to a given function from the given caller function in a given set of tasks. For example this method can be used to remove all flush() function calls from a debug() function.

Parameters

funcName	Name of the function
callerFunc	Function that calls the function that will be removed
brkpt	—

Returns

Number of tasks where the function call is removed.

4.6.3.20 void Application::RemoveTask (int *tid*) `[protected]`

Removes a task when an unregistered message is received (see processEvent).

Parameters

tid	process (thread) identifier of the task.

4.6.3.21 int Application::ReplaceFunction (string const & *oldFunc*, string const & *newFunc*, Breakpoint * *brkpt*)

Replaces all calls to a function with calls to another one in a given set of tasks.

Parameters

oldFunc	Name of the function to replace.
newFunc	Name of the new function.
brkpt	—

Returns

Number of tasks where the function calls were changed.

4.6.3.22 void Application::SetHostHandler (HostHandler & *h*)

Installs a callback function that is called when a new host is added to the virtual machine or an existing one is removed.

Parameters

h	Handler for the new hosts.

4.6.3.23 void Application::SetTaskHandler (TaskHandler & *h*)

Installs a callback function that is called when a new task is started or an existing one is terminated.

Parameters

h	Handler for the new tasks.

4.6.3.24 int Application::SetVariableValue (string const & *varName,* AttributeValue const & *varValue,* Breakpoint * *brkpt*)

Modifies a value of a specified variable in a given set of tasks.

application process.

Parameters

varName	Name of the variable.
varValue	New value for the variable.
brkpt	—

Returns

Number of tasks where the values were changed.

4.6.3.25 void Application::Start ( )

Starts the application.

Deprecated

The documentation for this class was generated from the following files:

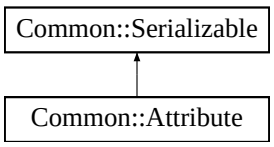
- Analyzer/AppModel.h
- Analyzer/AppModel.cpp

4.7 Common::Attribute Class Reference

Contains the necessary information of an attribute to be inserted in a program.

`#include <Utils.h>`

Inheritance diagram for Common::Attribute:



Public Member Functions

- Attribute (Attribute const &a)

  Copy constructor.
- Attribute ()

  Constructor.
- void Serialize (Serializer &out) const

  Sends the data serialized.
- void DeSerialize (DeSerializer &in)

  Gets the data deserialized.
- string GetSourceString () const

  Returns the string of the source.
- string GetTypeString () const

  Returns the type of the attribute.
- void Dump () const

  Logs the information of the attribute on the System Log.

Static Public Member Functions

- static string GetTypeString (AttrValueType type)

  Given a value of the enumerator AttrValueType returns the type in a string.

Public Attributes

- AttrSource **source**
- AttrValueType **type**
- string **id**

4.7.1 Detailed Description

Contains the necessary information of an attribute to be inserted in a program.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Common::Attribute::Attribute ( ) [inline]

Constructor.

Creates a default Attribute object of the integer type.

The documentation for this class was generated from the following files:

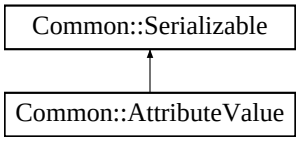
- Common/Utils.h
- Common/Utils.cpp

4.8 Common::AttributeValue Class Reference

Contains the vale of an attribute.

`#include <Utils.h>`

Inheritance diagram for Common::AttributeValue:



Public Member Functions

- AttributeValue ()
  Constructor.
- AttributeValue (AttributeValue const &av)
  Copy constructor.
- void operator= (AttributeValue const &av)
  Assignation operator, copies content of the given object.
- AttrValueType GetType () const
  Returns type of the attribute.
- void SetType (AttrValueType attrType)
  Sets the type of the attribute.
- void SetStrValue (std::string attrStrValue)
  Sets the string value of the attribute.
- int GetIntValue () const
  Gets the integer value.
- std::string GetStringValue () const
  Gets the string value.
- short GetShortValue () const

Gets the short value.

- float GetFloatValue () const

  Gets the float value.

- double GetDoubleValue () const

  Gets the double value.

- char GetCharValue () const

  Gets the char value.

- void * GetValueBuffer ()

  Gets the pointer to the buffer.

- int GetSize () const

  Gets the size of the value in memory.

- string ToString () const

  Returns the value in a string.

- void Serialize (Serializer &out) const

  Sends the data serialized.

- void DeSerialize (DeSerializer &in)

  Gets the data deserialized.

Public Attributes

- union {
  int **intValue**
  short **shortValue**
  float **floatValue**
  double **doubleValue**
  char **charValue**
  };

4.8.1 Detailed Description

Contains the vale of an attribute.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.8.2 Member Function Documentation

4.8.2.1 char AttributeValue::GetCharValue ( ) const [inline]

Gets the char value.

Exceptions

Exception	

4.8.2.2 double AttributeValue::GetDoubleValue ( ) const [inline]

Gets the double value.

Exceptions

Exception	

4.8.2.3 float AttributeValue::GetFloatValue ( ) const [inline]

Gets the float value.

Exceptions

Exception	

4.8.2.4 int AttributeValue::GetIntValue ( ) const [inline]

Gets the integer value.

Exceptions

Exception	

4.8.2.5 short AttributeValue::GetShortValue ( ) const [inline]

Gets the short value.

Exceptions

Exception	

4.8.2.6 std::string AttributeValue::GetStringValue ( ) const [inline]

Gets the string value.

Exceptions

Exception	

The documentation for this class was generated from the following files:

- Common/Utils.h
- Common/Utils.cpp

4.9 auto_iterator< T > Class Template Reference

Class that implements the *auto_ptrs* operators.

`#include <auto_vector.h>`

Public Member Functions

- **auto_iterator** (auto_ptr< T > *pp)
- bool **operator!=** (auto_iterator< T > const &it) const
- auto_iterator const & **operator++** (int)
- auto_iterator **operator++** ()
- T * **operator*** ()
- T const * **operator*** () const
- T * **operator->** ()

4.9.1 Detailed Description

template<class T>class auto_iterator< T >

Class that implements the *auto_ptrs* operators.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

The documentation for this class was generated from the following file:

- Common/auto_vector.h

4.10 auto_vector< T > Class Template Reference

Class with the methods to create, handle and destroy an array of *auto_ptr* elements.

`#include <auto_vector.h>`

Public Types

- typedef auto_iterator< T > **iterator**

Public Member Functions

- auto_vector (size_t capacity=0)

  Constructor.
- ~auto_vector ()

  Destructor.
- T const * **operator[]** (size_t i) const
- T * **operator[]** (size_t i)
- void assign (size_t i, auto_ptr< T > &p)

  Assigns p to the ith position in the auto_vector array _arr.
- void remove_direct (size_t i, T *p)

Deletes position i in the array.

- void Dump ()

  Prints the contents in the auto_vector array _arr.
- void clear ()

  Deletes all contents in the auto_vector array _arr.
- void push_back (auto_ptr< T > &p)

  Pushes an element p at the end of the auto_vector array _arr.
- auto_ptr< T > pop_back ()

  Pops the last element of the auto_vector array _arr.
- auto_ptr< T > acquire (size_t i)

  Gets the contents from vector _arr in the given position.
- iterator begin () const

  Returns a pointer to the first position in the array _arr.
- iterator end () const

  Returns a pointer to the last position in the array _arr.
- int size () const

  Returns the size of _arr.

4.10.1 Detailed Description

template<class T>class auto_vector< T >

Class with the methods to create, handle and destroy an array of *auto_ptr* elements.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.10.2 Constructor & Destructor Documentation

4.10.2.1 template<class T> auto_vector< T >::auto_vector (size_t *capacity* = 0) [inline], [explicit]

Constructor.

Parameters

capacity	Vector capacity. By default this is 0

4.10.3 Member Function Documentation

4.10.3.1 template<class T> auto_ptr<T> auto_vector< T >::acquire (size_t *i*) [inline]

Gets the contents from vector _arr in the given position.

Parameters

i	Position of the element we want to get from _arr

Returns

Contents of the *ith* element in _arr

4.10.3.2 template<class T> void auto_vector< T >::assign (size_t *i*, auto_ptr< T > & *p*) [inline]

Assigns *p* to the *ith* position in the auto_vector array _arr.

Parameters

i	Position where to make the assignment
p	Data that will be assigned

4.10.3.3 template<class T> iterator auto_vector< T >::begin ( ) const [inline]

Returns a pointer to the first position in the array _arr.

Returns

First position in _arr

4.10.3.4 template<class T> iterator auto_vector< T >::end ( ) const [inline]

Returns a pointer to the last position in the array _arr.

Returns

Ending position in _arr

4.10.3.5 template<class T> auto_ptr<T> auto_vector< T >::pop_back ( ) [inline]

Pops the last element of the auto_vector array _arr.

Returns

Last element in _arr

4.10.3.6 template<class T> void auto_vector< T >::push_back (auto_ptr< T > & *p*) [inline]

Pushes an element *p* at the end of the auto_vector array _arr.

Parameters

p	

4.10.3.7 template<class T> void auto_vector< T >::remove_direct (size_t *i*, T * *p*) [inline]

Deletes position *i* in the array.

Parameters

i	
p	

4.10.3.8 template<class T> int auto_vector< T >::size ( ) const [inline]

Returns the size of _arr.

Returns

Number of elements in _arr

The documentation for this class was generated from the following file:

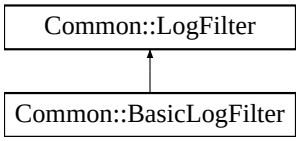
- Common/auto_vector.h

4.11 Common::BasicLogFilter Class Reference

Filters LogEntry objects to be inserted in a log.

`#include <Syslog.h>`

Inheritance diagram for Common::BasicLogFilter:



Public Member Functions

- BasicLogFilter ()
  Constructor.
- BasicLogFilter (int mask)
  Constructor.
- bool Accept (LogEntry const &entry) const
  Returns true if the log is accepted, false otherwise.

4.11.1 Detailed Description

Filters LogEntry objects to be inserted in a log.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Common::BasicLogFilter::BasicLogFilter (int *mask*) [inline]

Constructor.

Parameters

mask	Severity mask.

The documentation for this class was generated from the following files:

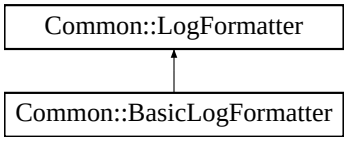
- Common/Syslog.h
- Common/Syslog.cpp

4.12 Common::BasicLogFormatter Class Reference

Formats LogEntry objects to be inserted in a log.

`#include <Syslog.h>`

Inheritance diagram for Common::BasicLogFormatter:



Public Member Functions

- BasicLogFormatter ()
  Constructor.
- BasicLogFormatter (Config &cfg)
  Constructor.
- std::string GetLogHeader () const
  Returns a string containing the log header.
- std::string GetLogFooter () const
  Returns a string containing the log footer.
- std::string Format (LogEntry const &entry) const
  Returns a string containing the LogEntry object formatted.
- void ShowTimestamp (bool value)
  Enables the timestamp view.
- void ShowSeverity (bool value)
  Enables the severity view.
- void ShowChannel (bool value)
  Enables the channel view.

4.12.1 Detailed Description

Formats LogEntry objects to be inserted in a log.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.12.2 Constructor & Destructor Documentation

4.12.2.1 BasicLogFormatter::BasicLogFormatter (Config & *cfg*)

Constructor.

Parameters

cfg	A Config object containing initial settings of the log formatter.

The documentation for this class was generated from the following files:

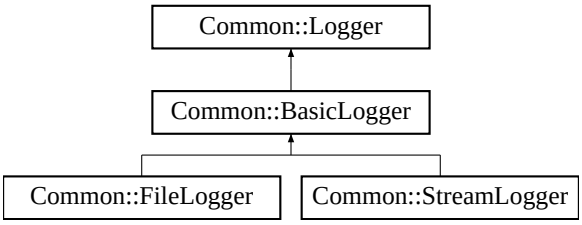
- Common/Syslog.h
- Common/Syslog.cpp

4.13 Common::BasicLogger Class Reference

Stores information of events in a system.

`#include <Syslog.h>`

Inheritance diagram for Common::BasicLogger:



Public Member Functions

- BasicLogger ()
  Constructor.
- void SetFilter (LogFilterPtr &filter)
  Sets the LogFilter to be used by the logger.
- LogFilter const * GetFilter () const
  Returns the LogFilter the logger uses.
- void SetFormatter (LogFormatterPtr &formatter)
  Sets the LogFormatter to be used by the logger.
- LogFormatter const * GetFormatter () const
  Returns the LogFormatter the logger uses.
- bool Accept (LogEntry const &entry) const
  Inserts an entry to the log.

Protected Attributes

- LogFilterPtr **_filter**
- LogFormatterPtr **_formatter**

4.13.1 Detailed Description

Stores information of events in a system.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.13.2 Member Function Documentation

4.13.2.1 bool BasicLogger::Accept (LogEntry const & *entry*) const

Inserts an entry to the log.

Returns

True if the insert was successful, false otherwise.

The documentation for this class was generated from the following files:

- Common/Syslog.h
- Common/Syslog.cpp

4.14 BatchData Class Reference

Statistics of a single batch.

`#include <FactoringStats_nw.h>`

Public Member Functions

- BatchData (int batchIdx)

  Constructor.
- ~BatchData ()

  Destructor.
- void OnNewBatch (int numChunks)

  Sets the number of chunks to numChunks
- WorkerData & GetWorkerData (int workerTid)

  Getter of the data in worker given by workerTid. If it does not exist, creates a new one and returns it.
- WorkerData & NewWorkerData (int workerTid)

Creates a new worker data in workerTid and returns it. If it already exists, returns it.

- bool IsComplete () const

  Checks if the current BatchData is complete.
- bool IsActualize () const

  Getter of the _flagActualize private var.
- void SetActualize ()

  Sets _flagActualize to 1.
- bool AreWorkersComplete () const

  *Checks if all the workers **in BatchData** are complete.*
- WorkerData ** AllocWorkersArray ()

  Allocates a dynamic array of WorkerData and returns it.
- double DeviationComputingTime ()

  Not implemented.
- double MeanComputingTime ()

  Not implemented.
- int GetNumChunks () const

  Getter of _numChunks.
- double GetMeanStats ()

  Calculates and returns the mean task processing time.
- double GetStdStats ()

  Calculates and returns the standard deviation of the tasks.
- void SizeTaskReceived (int sizeTasks)

  Adds sizeTasks to _TotalTaskReceived.
- int GetSizeTaskReceived () const

  _TotalTaskReceived getter
- ModelParam GetModelParam ()

  Returns the ModelParam object with TotalDataVolume, TotalDataSendW and TotalCompTime updated.

4.14.1 Detailed Description

Statistics of a single batch.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 BatchData::BatchData (int *batchIdx*)

Constructor.

Parameters

batchIdx	

4.14.3 Member Function Documentation

4.14.3.1 WorkerData ** BatchData::AllocWorkersArray ( )

Allocates a dynamic array of WorkerData and returns it.

Returns

WorkerData array

4.14.3.2 bool BatchData::AreWorkersComplete ( ) const

Checks if all the workers **in BatchData** are complete.

Returns

boolean

4.14.3.3 double BatchData::DeviationComputingTime ( )

Not implemented.

Returns

4.14.3.4 double BatchData::GetMeanStats ( )

Calculates and returns the mean task processing time.

Returns

mean time in **seconds?**

4.14.3.5 ModelParam BatchData::GetModelParam ( )

Returns the ModelParam object with TotalDataVolume, TotalDataSendW and TotalCompTime updated.

Returns

ModelParam object

4.14.3.6 int BatchData::GetNumChunks ( ) const [inline]

Getter of _numChunks.

Returns

_numChunks

4.14.3.7 int BatchData::GetSizeTaskReceived ( ) const [inline]

_TotalTaskReceived getter

Returns

_TotalTaskReceived

4.14.3.8 double BatchData::GetStdStats ( )

Calculates and returns the standard deviation of the tasks.

Returns

standard deviation

4.14.3.9 WorkerData & BatchData::GetWorkerData (int *workerTid*)

Getter of the data in worker given by workerTid. If it does not exist, creates a new one and returns it.

Parameters

workerTid	Worker ID

Returns

WorkerData object

4.14.3.10 bool BatchData::IsActualize ( ) const [inline]

Getter of the _flagActualize private var.

Returns

returns _flagActualize

4.14.3.11 bool BatchData::IsComplete ( ) const [inline]

Checks if the current BatchData is complete.

Returns

boolean

4.14.3.12 double BatchData::MeanComputingTime ( )

Not implemented.

Returns

4.14.3.13 WorkerData & BatchData::NewWorkerData (int *workerTid*)

Creates a new worker data in workerTid and returns it. If it already exists, returns it.

Parameters

workerTid	Worker ID

Returns

WorkerData Object

4.14.3.14 void BatchData::OnNewBatch (int *numChunks*) [inline]

Sets the number of chunks to *numChunks*

Parameters

numChunks	Number of chunks

4.14.3.15 void BatchData::SizeTaskReceived (int *sizeTasks*) [inline]

Adds *sizeTasks* to _TotalTaskReceived.

Parameters

sizeTasks	

The documentation for this class was generated from the following files:

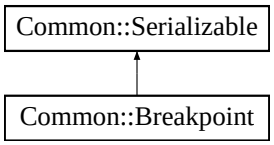
- Analyzer/FactoringStats_nw.h
- Analyzer/FactoringStats_nw.cpp

4.15 Common::Breakpoint Class Reference

Denotes place in a function (on the entry or at the end).

`#include <Utils.h>`

Inheritance diagram for Common::Breakpoint:



Public Member Functions

- Breakpoint ()

  Constructor.
- Breakpoint (Breakpoint const &b)

  Copy Constructor.
- void Serialize (Serializer &out) const

  Serializes the breakpoint through the given Serializer.
- void DeSerialize (DeSerializer &in)

  Deserializes the breakpoint from the given DeSerializer.

Public Attributes

- std::string **funcName**
- InstrPlace **place**

4.15.1 Detailed Description

Denotes place in a function (on the entry or at the end).

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

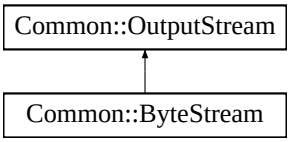
- Common/Utils.h

4.16 Common::ByteStream Class Reference

Stores stream of bytes.

`#include <ByteStream.h>`

Inheritance diagram for Common::ByteStream:



Public Member Functions

- ByteStream (char *buf, size_t bufSize)
  Constructor.
- ByteStream (size_t bufSize)
  Constructor. Creates an intern buffer.
- void Write (char const *buf, size_t bufSize)
  Adds the content of the buffer to the stream.
- char const * GetData () const
  Returns pointer to the intern buffer.
- size_t GetDataSize () const
  Returns size of the stream.
- void Reset ()
  Clears the stream.

4.16.1 Detailed Description

Stores stream of bytes.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Common::ByteStream::ByteStream (char * *buf*, size_t *bufSize*) [inline]

Constructor.

Parameters

buf	Intern buffer to be used.
bufSize	Size of the intern buffer.

4.16.2.2 Common::ByteStream::ByteStream (size_t *bufSize*) [inline]

Constructor. Creates an intern buffer.

Parameters

bufSize	Size of the intern buffer.

4.16.3 Member Function Documentation

4.16.3.1 char const* Common::ByteStream::GetData ( ) const [inline]

Returns pointer to the intern buffer.

When used the returned pointer should always use the GetDataSize() method to iterate over the buffer.

Returns

Read-only pointer to the intern buffer.

4.16.3.2 size_t Common::ByteStream::GetDataSize ( ) const [inline]

Returns size of the stream.

Returns

Buffer size.

4.16.3.3 void ByteStream::Write (char const * *buf*, size_t *bufSize*) [virtual]

Adds the content of the buffer to the stream.

Parameters

buf	Buffer to read.
bufSize	Size of the given buffer.

Implements Common::OutputStream.

The documentation for this class was generated from the following files:

- Common/ByteStream.h
- Common/ByteStream.cpp

4.17 CommandLine Class Reference

Encapsulates methods to interact with the user of analyzer. Basically reads the arguments of the analyzer and parses them to get the configuration file and the objective application with its parameters. Once read, encapsulates the information and provides accessors to it. There are two formats Analyzer can be called:

`#include <cmdline.h>`

Public Member Functions

- CommandLine (int argc, char **argv)

  Constructor.
- bool IsOk () const

  Returns the value of _isOk.
- int GetArgc () const

  Getter for the _argc variable.
- char ** GetArgv () const

  Getter for the _argv variable.
- int GetAppArgc () const

  Getter for the _appArgc variable.
- char * GetAppPath () const

  Getter for the _appPath variable.
- char ** GetAppArgv () const

  Getter for the _appArgv variable.
- bool HasConfig () const

  Checks if there's a path for the configuration file, if not returns 0.
- char * GetConfigFileName () const

  Getter for the _configFile variable.
- void DisplayHelp () const

  Prints help message on the terminal. Tells the user how to introduce the necessary arguments.
- CommandLine (int argc, char **argv)

  Constructor, parses the arguments provided to analyzer.
- bool IsOk () const

  Status of the arguments getter.
- int GetArgc () const

  Number of arguments getter.
- char ** GetArgv () const

  Arguments getter.
- bool HasConfig () const

  Determines if the user has chosen his own configuration file.
- char * GetConfigFile () const

  Configuration file getter.
- char const * GetAppPath () const

  Application path getter.
- int GetAppArgc () const

  Application number of arguments getter.
- char const ** GetAppArgv () const

  Application arguments getter.
- void DisplayHelp () const

  Explains the user which arguments can be provided to analyzer.

4.17.1 Detailed Description

Encapsulates methods to interact with the user of analyzer. Basically reads the arguments of the analyzer and parses them to get the configuration file and the objective application with its parameters. Once read, encapsulates the information and provides accessors to it. There are two formats Analyzer can be called:

Checks for the necessary data in the arguments passed to main and parses them.

- Analyzer <AppPath> [<AppArgs>]
- Analyzer -config file.ini <App> [<AppArgs>]

Notes

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.17.2 Constructor & Destructor Documentation

4.17.2.1 CommandLine::CommandLine (int *argc*, char ** *argv*) [inline]

Constructor, parses the arguments provided to analyzer.

Parameters

argc	number of arguments for analyzer
argv	arguments for analyzer

4.17.3 Member Function Documentation

4.17.3.1 int CommandLine::GetAppArgc ( ) const [inline]

Getter for the _appArgc variable.

Returns

Size of the arguments vector for the app.

4.17.3.2 int CommandLine::GetAppArgc ( ) const [inline]

Application number of arguments getter.

Returns

The number of arguments of the target application.

4.17.3.3 char** CommandLine::GetAppArgv ( ) const [inline]

Getter for the _appArgv variable.

Returns

Vector that contains the app's arguments.

4.17.3.4 char const** CommandLine::GetAppArgv ( ) const [inline]

Application arguments getter.

Returns

The arguments of the target application.

4.17.3.5 char* CommandLine::GetAppPath ( ) const [inline]

Getter for the _appPath variable.

Returns

Path to the executable of the app.

4.17.3.6 char const* CommandLine::GetAppPath ( ) const [inline]

Application path getter.

Returns

The path of the target application.

4.17.3.7 int CommandLine::GetArgc ( ) const [inline]

Getter for the _argc variable.

Returns

Size of the vector of arguments.

4.17.3.8 int CommandLine::GetArgc ( ) const [inline]

Number of arguments getter.

Returns

The number of arguments provided to analyzer.

4.17.3.9 char** CommandLine::GetArgv ( ) const [inline]

Getter for the _argv variable.

Returns

Vector of arguments.

4.17.3.10 char** CommandLine::GetArgv ( ) const [inline]

Arguments getter.

Returns

The arguments provided to analyzer.

4.17.3.11 char* CommandLine::GetConfigFile ( ) const [inline]

Configuration file getter.

Returns

The configuration file of Analyzer.

4.17.3.12 char* CommandLine::GetConfigFileName ( ) const [inline]

Getter for the _configFile variable.

Returns

Path for the configuration file.

4.17.3.13 bool CommandLine::HasConfig ( ) const [inline]

Determines if the user has chosen his own configuration file.

Returns

If the user provided a specific configuration file.

4.17.3.14 bool CommandLine::HasConfig ( ) const [inline]

Checks if there's a path for the configuration file, if not returns 0.

Returns

Path for the configuration file.

4.17.3.15 bool CommandLine::IsOk ( ) const [inline]

Returns the value of _isOk.

Returns

Boolean variable that is true if the configuration has been parsed correctly.

4.17.3.16 bool CommandLine::IsOk ( ) const [inline]

Status of the arguments getter.

Returns

if the arguments provided to analyzer are correct or not.

The documentation for this class was generated from the following files:

- AC/cmdline.h
- Analyzer/cmdline.h

4.18 Common::Config Class Reference

Manages a configuration of the system.

`#include <Config.h>`

Classes

- class KeyIterator

  Iterates over the keys of a Config object.

Public Member Functions

- Config ()

  Constructor.
- std::string const & GetStringValue (std::string const &section, std::string const &key) const

  Returns string value of the entry specified by the parameters.
- int GetIntValue (std::string const &section, std::string const &key) const

  Returns integer value of the entry specified by the parameters.
- int GetIntValue (std::string const &section, std::string const &key, int defaultValue) const

  Returns integer value of the entry specified by the parameters.
- bool GetBoolValue (std::string const &section, std::string const &key) const

  Returns boolean value of the entry specified by the parameters.
- bool GetBoolValue (std::string const &section, std::string const &key, bool defaultValue) const

  Returns boolean value of the entry specified by the parameters.
- bool Contains (std::string const &section, std::string const &key) const

  Finds an entry on the configuration.
- KeyIterator GetKeys (std::string const &section) const

  Returns an iterator of the keys inside the requested section.
- void AddEntry (std::string const &section, std::string const &key, std::string const &value)

  Adds a new entry to the configuration.

Friends

- class **KeyIterator**
- class **ConfigReader**

4.18.1 Detailed Description

Manages a configuration of the system.

The configuration is based on section and keys, and the format is the following:

```
[section]
key = value
key = value
...

[newsection]
key = value
...
```

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2000

4.18.2 Member Function Documentation

4.18.2.1 void Common::Config::AddEntry (std::string const & *section,* std::string const & *key,* std::string const & *value*) [inline]

Adds a new entry to the configuration.

Parameters

section	Section of the new entry.
key	Key of the new entry.
value	Value of the new entry.

4.18.2.2 bool Common::Config::Contains (std::string const & *section,* std::string const & *key*) const [inline]

Finds an entry on the configuration.

Parameters

section	Section to find the entry.
key	Key to find the entry.

Returns

True if the entry was found, false otherwise.

4.18.2.3 bool Common::Config::GetBoolValue (std::string const & *section,* std::string const & *key*) const

Returns boolean value of the entry specified by the parameters.

Parameters

section	Section to find the value.
key	Key to find the value.

Returns

Boolean containing the requested value.

Exceptions

ConfigException	

4.18.2.4 bool Common::Config::GetBoolValue (std::string const & *section*, std::string const & *key*, bool *defaultValue*) const [inline]

Returns boolean value of the entry specified by the parameters.

If the configuration doesn't contain the specified entry, returns the default value.

Parameters

section	Section to find the value.
key	Key to find the value.
defaultValue	Value returned if the requested entry is not inside the configuration.

Returns

Boolean containing the requested value.

4.18.2.5 int Common::Config::GetIntValue (std::string const & *section*, std::string const & *key*) const

Returns integer value of the entry specified by the parameters.

Parameters

section	Section to find the value.
key	Key to find the value.

Exceptions

ConfigException	

4.18.2.6 int Common::Config::GetIntValue (std::string const & *section*, std::string const & *key*, int *defaultValue*) const [inline]

Returns integer value of the entry specified by the parameters.

If the configuration doesn't contain the specified entry, returns the default value.

Parameters

section	Section to find the value.
key	Key to find the value.

defaultValue	Value returned if the requested entry is not inside the configuration.

Returns

Integer containing the requested value.

4.18.2.7 KeyIterator Common::Config::GetKeys (std::string const & *section*) const [inline]

Returns an iterator of the keys inside the requested section.

Parameters

section	Section requested.

Returns

Iterator to the keys inside the section.

4.18.2.8 std::string const& Common::Config::GetStringValue (std::string const & *section*, std::string const & *key*) const [inline]

Returns string value of the entry specified by the parameters.

Parameters

section	Section to find the value.
key	Key to find the value.

Exceptions

ConfigException	

The documentation for this class was generated from the following file:

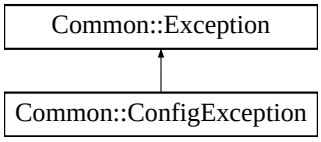
- Common/Config.h

4.19 Common::ConfigException Class Reference

Config, ConfigReader and ConfigMap exceptions.

`#include <ConfigException.h>`

Inheritance diagram for Common::ConfigException:



Public Member Functions

- ConfigException (std::string const &msg, std::string const &objName=std::string())

  Constructor.
- void Display (std::ostream &os) const

Displays exception message on the given output stream.

- void Display () const

  Displays exception message on the standard error output.

- std::string GetReason () const

  Returns a string containing the error message.

Additional Inherited Members

4.19.1 Detailed Description

Config, ConfigReader and ConfigMap exceptions.

Version

1.0b

Since

1.0b

Author

Noel De Martin, 2011

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Common::ConfigException::ConfigException (std::string const & *msg*, std::string const & *objName* = std::string ()) [inline]

Constructor.

Parameters

msg	Exception message.
objName	Name of the object causing the exception, "" by default.

4.19.3 Member Function Documentation

4.19.3.1 void Common::ConfigException::Display (std::ostream & *os*) const [virtual]

Displays exception message on the given output stream.

Parameters

os	Output stream to display the message.

Reimplemented from Common::Exception.

4.19.3.2 string ConfigException::GetReason ( ) const

Returns a string containing the error message.

Returns

String with the error.

The documentation for this class was generated from the following files:

- Common/ConfigException.h
- Common/ConfigException.cpp

4.20 Common::ConfigHelper Class Reference

Static class that contains methods to manage Config objects.

`#include <Config.h>`

Static Public Member Functions

- static Config ReadFromFile (std::string const &fileName)

  Returns a Config object loaded from the given file.

4.20.1 Detailed Description

Static class that contains methods to manage Config objects.

Version

1.0b

Since

1.0b

Author

Noel De Martin, 2011

4.20.2 Member Function Documentation

4.20.2.1 static Config Common::ConfigHelper::ReadFromFile (std::string const & *fileName*) [inline], [static]

Returns a Config object loaded from the given file.

Exceptions

ConfigException	

The documentation for this class was generated from the following file:

- Common/Config.h

4.21 Common::ConfigMap Class Reference

Contains and manages a collection of Config objects.

`#include <ConfigMap.h>`

Classes

- class Iterator

  Iterates over a ConfigMap object.

Public Member Functions

- ConfigMap ()

  Constructor.
- bool Add (std::string const &section, std::string const &key, std::string const &value)

  Adds a new value to the map.
- std::string const & GetValue (std::string const &section, std::string const &key) const

  Returns a requested value on the map.
- bool Contains (std::string const &section, std::string const &key) const

  Looks for the entry specified by the parameters of the function.
- int GetSize () const

  Returns size of the map.

Friends

- class **Iterator**

4.21.1 Detailed Description

Contains and manages a collection of Config objects.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2000

4.21.2 Member Function Documentation

4.21.2.1 bool ConfigMap::Add (std::string const & *section*, std::string const & *key*, std::string const & *value*)

Adds a new value to the map.

If the entry already exists on the map returns false.

Parameters

section	Section of the new entry.
key	Key of the new entry.

value	Value of the new entry.

Returns

True if the insertion was successful, false otherwise.

4.21.2.2 bool ConfigMap::Contains (std::string const & *section*, std::string const & *key*) const

Looks for the entry specified by the parameters of the function.

Parameters

section	Section to find the entry.
key	Key to find the entry.

Returns

True if the entry was found, false otherwise.

4.21.2.3 string const & ConfigMap::GetValue (std::string const & *section*, std::string const & *key*) const

Returns a requested value on the map.

Parameters

section	Section to find the value.
key	Key to find the value.

Exceptions

ConfigException	

The documentation for this class was generated from the following files:

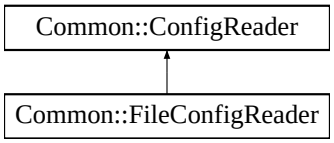
- Common/ConfigMap.h
- Common/ConfigMap.cpp

4.22 Common::ConfigReader Class Reference

Abstract class, generates Config objects from reading sources.

`#include <ConfigReader.h>`

Inheritance diagram for Common::ConfigReader:



Public Member Functions

- ConfigReader ()

  Constructor.
- virtual Config **Read** ()=0

Protected Member Functions

- void AnalyzeLine (Config &config, std::string const &line)

  Loads the information of the line into the Config object.

4.22.1 Detailed Description

Abstract class, generates Config objects from reading sources.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2000

The documentation for this class was generated from the following files:

- Common/ConfigReader.h
- Common/ConfigReader.cpp

4.23 Controller Class Reference

Provides the logic and controls the execution flow of the application.

`#include <Ctrl.h>`

Public Member Functions

- Controller (CommandLine &cmdLine)

  Constructor.
- ~Controller ()

  Destructor.
- void Run ()

  Initializes all the necessary fields and starts the main loop of the AC.
- void Interrupt ()

  Sets the _fInterrupted variable to 1.
- Controller (CommandLine &cmdLine, std::string const &cfgFile)

  Constructor, sets the command line for the user, determines the configuration for the application and prepares the system log.
- void Run (ShutDownManager *sdm)

  Manages the execution flow of the application. The execution flow of analyzer is:

4.23.1 Detailed Description

Provides the logic and controls the execution flow of the application.

Contains the main functionality of the AC, including its main loop which runs until all tuning operations have been finished.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.23.2 Constructor & Destructor Documentation

4.23.2.1 Controller::Controller (CommandLine & *cmdLine*)

Constructor.

Parameters

cmdLine	Class that provides commandline communications with the user.

4.23.3 Member Function Documentation

4.23.3.1 void Controller::Run (ShutDownManager * *sdm*)

Manages the execution flow of the application. The execution flow of analyzer is:

- create DTAPI, initialize collector, etc.
- create application model
- initialize all tunlets
- start application
- handle events
- destroy tunlets
- destroy app model

4.23.3.2 void Controller::Run ( )

Initializes all the necessary fields and starts the main loop of the AC.

Creates a TaskManager objects and a Reactor and PTPAcceptor which will provide event handling and tuning capabilities.

The documentation for this class was generated from the following files:

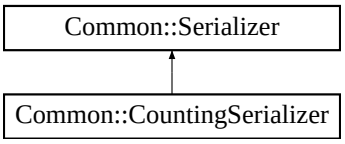
- AC/Ctrl.h
- Analyzer/Ctrl.h
- AC/Ctrl.cpp
- Analyzer/Ctrl.cpp

4.24 Common::CountingSerializer Class Reference

Stores the size of serialized data.

`#include <NetSer.h>`

Inheritance diagram for Common::CountingSerializer:



Public Member Functions

- CountingSerializer ()
  Constructor.
- int_t GetSize () const
  Returns size of the serialized data.
- void PutLong (long_t l)
  Adds the size of a serialized long.
- void PutDouble (double_t d)
  Adds the size of a serialized double.
- void PutBool (bool_t b)
  Adds the size of a serialized boolean.
- void PutShort (short_t s)
  Adds the size of a serialized short.
- void PutByte (byte_t b)
  Adds the size of a serialized byte.
- void PutChar (char_t c)
  Adds the size of a serialized char.
- void PutString (std::string const &str)
  Adds the size of a serialized string.
- void PutInt (int_t i)
  Adds the size of a serialized integer.
- void PutBuffer (char const *buffer, int bufferSize)
  Adds the size of a serialized buffer.

4.24.1 Detailed Description

Stores the size of serialized data.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- Common/NetSer.h

4.25 curState Class Reference

Struct that stores the iteration, batch and number of tuples.

Public Attributes

- int **iter**
- int **batch**
- int **numTuples**

4.25.1 Detailed Description

Struct that stores the iteration, batch and number of tuples.

The documentation for this class was generated from the following files:

- Analyzer/FactoringTunlet_nw-XFire.cpp
- Analyzer/FactoringTunlet_nw.cpp

4.26 Common::DateTime Class Reference

Holds a timestamp.

`#include <DateTime.h>`

Public Member Functions

- DateTime ()

  Constructor, sets the current date and time.
- int GetYear () const

  Returns year represented by this date.
- int GetMonth () const

  Returns month represented by this date.
- int GetDay () const

  Returns day represented by this date.
- int GetHour () const

  Returns hour represented by this date.
- int GetMinute () const

  Returns minute represented by this date.
- int GetSecond () const

  Returns second represented by this date.
- std::string GetStringValue () const

  Returns a string with the date.

4.26.1 Detailed Description

Holds a timestamp.

Version

1.0

Since

1.0

Author

Ania Sikora, 2001

4.26.2 Member Function Documentation

4.26.2.1 string DateTime::GetStringValue ( ) const

Returns a string with the date.

The format of the returned string is "dd.mm.yyyy hh:MM:ss".

The documentation for this class was generated from the following files:

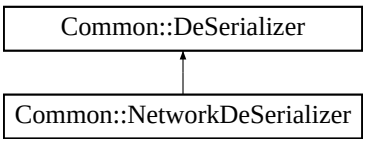
- Common/DateTime.h
- Common/DateTime.cpp

4.27 Common::DeSerializer Class Reference

Abstract class, recovers serialized data from a stream.

`#include <Serial.h>`

Inheritance diagram for Common::DeSerializer:



Public Member Functions

- virtual byte_t GetByte ()=0

  Reads byte value from the stream.
- virtual char_t GetChar ()=0

  Reads char value from the stream.
- virtual bool_t GetBool ()=0

  Reads bool value from the stream.
- virtual short_t GetShort ()=0

  Reads short value from the stream.
- virtual int_t GetInt ()=0

Reads int value from the stream.

- virtual long_t GetLong ()=0

  Reads long value from the stream.

- virtual double_t GetDouble ()=0

  Reads double value from the stream.

- virtual std::string GetString ()=0

  Reads string value from the stream.

- virtual void GetBuffer (char *buffer, int bufferSize)=0

  Reads data directly from the stream.

4.27.1 Detailed Description

Abstract class, recovers serialized data from a stream.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- Common/Serial.h

4.28 DiEx Class Reference

Implements the Dyninst's Exceptions.

`#include <di.h>`

Public Member Functions

- **_objName** (objName)
- string const & GetMessage () const

  Exception message getter.

- string const & GetObjectName () const

  Object Name getter.

Public Attributes

- __pad0__: _msg (msg)

  Constructor.

4.28.1 Detailed Description

Implements the Dyninst's Exceptions.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.28.2 Member Function Documentation

4.28.2.1 string const& DiEx::GetMessage ( ) const [inline]

Exception message getter.

Returns

_msg

4.28.2.2 string const& DiEx::GetObjectName ( ) const [inline]

Object Name getter.

Returns

_objName

4.28.3 Member Data Documentation

4.28.3.1 DiEx::__pad0__

Constructor.

Parameters

msg	Exception message to display
objName	Object Name

The documentation for this class was generated from the following file:

- Common/di.h

4.29 DiFunction Class Reference

Dyninst's function class. It represents a function in the application.

`#include <di.h>`

Public Member Functions

- **DiFunction** (BPatch_image &bpImage, string const &funcName)
- void GetLineNumber (unsigned int &start, unsigned int &end, char *fileName, unsigned int &max)

  Gets the current line number and the file name.
- unsigned long GetAddress ()

  Reads the address of _bpVar.
- char const * GetParams ()

  Gets the parameters of _bpFunc.
- PointVector * FindPoint (BPatch_procedureLocation loc=BPatch_subroutine)

  Finds the procedure point for the given location.
- void GetName (char *fileName, int len)

  Getter of the file name.
- **operator BPatch_function &** ()

Static Public Member Functions

- static void Dump (FuncVector &fv)

  Prints all functions from the FuncVector fv

4.29.1 Detailed Description

Dyninst's function class. It represents a function in the application.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.29.2 Member Function Documentation

4.29.2.1 void DiFunction::Dump (DiFunction::FuncVector & *fv*) [static]

Prints all functions from the FuncVector *fv*

Parameters

fv	FuncVector

4.29.2.2 DiFunction::PointVector * DiFunction::FindPoint (BPatch_procedureLocation *loc* = BPatch_subroutine)

Finds the procedure point for the given location.

Parameters

loc	Location of the point to look for

Returns

Point Vector of the specified location

4.29.2.3 unsigned long DiFunction::GetAddress ( )

Reads the address of _bpVar.

Returns

Address of _bpVar

4.29.2.4 void DiFunction::GetLineNumber (unsigned int & *start*, unsigned int & *end*, char * *fileName*, unsigned int & *max*)

Gets the current line number and the file name.

Parameters

start	Address
end	Line
fileName	String where the file name will be saved
max	Length

4.29.2.5 void DiFunction::GetName (char * *fileName*, int *len*)

Getter of the file name.

Parameters

fileName	Parameter where the file name will be stored
len	max length of the file name

4.29.2.6 char const * DiFunction::GetParams ( )

Gets the parameters of _bpFunc.

Returns

Parameters of the current function

The documentation for this class was generated from the following files:

- Common/di.h
- Common/di.cpp

4.30 DiImage Class Reference

Reads the program's image and gets an associated image object (the executable associated with a thread). It can also find a variable in the image and return it.

```
#include <di.h>
```

Public Member Functions

- **DiImage** (BPatch_process &bpProcess)
- BPatch_variableExpr * FindVariable (const char *name)

  Finds the variable from _bpImage via name and returns it.
- **operator BPatch_image &** ()

4.30.1 Detailed Description

Reads the program's image and gets an associated image object (the executable associated with a thread). It can also find a variable in the image and return it.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.30.2 Member Function Documentation

4.30.2.1 BPatch_variableExpr * DiImage::FindVariable (const char * *name*)

Finds the variable from _bpImage via name and returns it.

Parameters

name	Name of the variable

Returns

Global variable matching <name> in the image. NULL if not found.

The documentation for this class was generated from the following files:

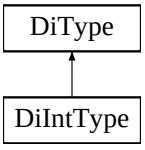
- Common/di.h
- Common/di.cpp

4.31 DiIntType Class Reference

Dyninst Int type class.

`#include <di.h>`

Inheritance diagram for DiIntType:



Public Member Functions

- **DiIntType** (BPatch_image &bpImage)

4.31.1 Detailed Description

Dyninst Int type class.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

The documentation for this class was generated from the following file:

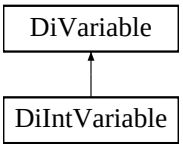
- Common/di.h

4.32 DiIntVariable Class Reference

Dyninst's int variable class.

`#include <di.h>`

Inheritance diagram for DiIntVariable:



Public Member Functions

- **DiIntVariable** (BPatch_process &bpProcess)

4.32.1 Detailed Description

Dyninst's int variable class.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

The documentation for this class was generated from the following file:

- Common/di.h

4.33 DiPoint Class Reference

Dyninst's point class. An object of this class represents a location in an application's code at which DynInst can insert instrumentation.

`#include <di.h>`

Public Member Functions

- DiPoint (BPatch_image &bpImage, string const &procName, BPatch_procedureLocation loc=BPatch_entry)

  Finds the given location in the function with a given name.
- void GetCalledFuncName (char *buf, int size)

  Getter of the current function name.
- unsigned long GetAddress ()

  Getter of the current address.
- **operator PointVector &** ()
- PointVector & getPoints ()

  _bpPoints getter

4.33.1 Detailed Description

Dyninst's point class. An object of this class represents a location in an application's code at which DynInst can insert instrumentation.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.33.2 Constructor & Destructor Documentation

4.33.2.1 DiPoint::DiPoint (BPatch_image & *bpImage,* string const & *procName,* BPatch_procedureLocation *loc =* `BPatch_entry`)

Finds the given location in the function with a given name.

Parameters

bpImage	BPatch Image of the program
procName	Process Name
loc	Location

4.33.3 Member Function Documentation

4.33.3.1 unsigned long DiPoint::GetAddress ( )

Getter of the current address.

Returns

address

4.33.3.2 void DiPoint::GetCalledFuncName (char * *buf*, int *size*)

Getter of the current function name.

Parameters

buf	Returned name
size	Max length of the buffer

4.33.3.3 PointVector& DiPoint::getPoints ( ) [inline]

_bpPoints getter

Returns

_bpPoints

The documentation for this class was generated from the following files:

- Common/di.h
- Common/di.cpp

4.34 DiProcess Class Reference

Operates on code in execution. This class can be used to manipulate the process.

`#include <di.h>`

Public Member Functions

- DiProcess ()
- DiProcess (char *mutateeName, int pid)

  Attaches the program to a running process.
- DiProcess (char *mutateeName, char *argv[], char *envp[]=0)

  Creates the program with the given arguments.
- DiProcess (char *mutateeName)

  Creates the program (mutatee)

- ~DiProcess ()

  Destructor.
- **operator BPatch_process &** ()
- int GetPid ()

  Getter of the Pid.
- bool IsStopped ()

  Asserts that _bpProcess has been created and returns a boolean.
- BPatchSnippetHandle * InsertSnippet (BPatch_snippet const &expr, BPatch_point &point)

  Inserts a given snippet into the given point.
- BPatchSnippetHandle * InsertSnippet (BPatch_snippet const &expr, BPatch_point &point, BPatch_callWhen when, BPatch_snippetOrder order)

  Inserts the snippet into the point in a given order.
- BPatchSnippetHandle * InsertSnippetBefore (BPatch_snippet const &expr, BPatch_point &point)

  Inserts a given snippet before the given point.
- BPatchSnippetHandle * InsertSnippetAfter (BPatch_snippet const &expr, BPatch_point &point)

  Inserts a given snippet after the given point.
- BPatchSnippetHandle * InsertSnippet (BPatch_snippet const &expr, PointVector &points)

  Inserts the given snippet in the given points.
- BPatchSnippetHandle * InsertSnippet (BPatch_snippet const &expr, PointVector &points, BPatch_callWhen when, BPatch_snippetOrder order)

  Inserts the given snippet in the series of points with the given order and moment.
- BPatchSnippetHandle * InsertSnippetBefore (BPatch_snippet const &expr, PointVector &points)

  Inserts the snippet before the given points.
- BPatchSnippetHandle * InsertSnippetAfter (BPatch_snippet const &expr, PointVector &points)

  Inserts the snippet after the given points.
- void DeleteSnippet (BPatchSnippetHandle *handle)

  Deletes the snippet in the given handle.
- void OneTimeCode (BPatch_snippet const &expr)

  Executes the given snippet once.
- void ReplaceFunction (BPatch_function &oldFunc, BPatch_function &newFunc)

  Replaces a function call with a call to another function.
- void ContinueExecution ()

  Resumes the execution of mutatee process.
- bool StopExecution ()

  Stops the execution of the mutatee process.
- void WaitFor ()

  Waits for the termination of the process.
- void Test ()

  Tests the current mutatee process by waiting for a status change.
- bool Terminate ()

  Terminates the mutatee process.
- bool IsTerminated ()

  Returns true if the mutatee process is terminated.
- void WaitForStop ()

  Waits for the mutatee process to stop.
- void loadLibrary (char *libName)

  Loads a shared library into the mutatee's address space. Returns true if successful.
- void GetLineNumber (unsigned long addr, unsigned short &line, char *fileName, int length)

  Gets information about the given line number from the mutatee process.
- BPatch_variableExpr * Malloc (BPatch_type &type)

  Allocates a new variable of the given type.

4.34.1 Detailed Description

Operates on code in execution. This class can be used to manipulate the process.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.34.2 Constructor & Destructor Documentation

4.34.2.1 DiProcess::DiProcess ( ) [inline]

Deprecated

4.34.2.2 DiProcess::DiProcess (char * *mutateeName,* int *pid*)

Attaches the program to a running process.

Parameters

mutateeName	Name of the mutatee
pid	PID to give to the process

4.34.2.3 DiProcess::DiProcess (char * *mutateeName,* char * *argv[],* char * *envp[]* = 0)

Creates the program with the given arguments.

Parameters

mutateeName	Name of the mutatee
argv	Arguments to give to the program. This cannot be null
envp	Environment list

4.34.2.4 DiProcess::DiProcess (char * *mutateeName*)

Creates the program (mutatee)

Parameters

mutateeName	Name of the mutatee

4.34.3 Member Function Documentation

4.34.3.1 void DiProcess::DeleteSnippet (BPatchSnippetHandle * *handle*)

Deletes the snippet in the given handle.

Parameters

handle	

4.34.3.2 void DiProcess::GetLineNumber (unsigned long *addr*, unsigned short & *line*, char * *fileName*, int *length*)

Gets information about the given line number from the mutatee process.

Parameters

addr	
line	
fileName	
length	

4.34.3.3 int DiProcess::GetPid ( ) [inline]

Getter of the Pid.

Returns

Pid

4.34.3.4 BPatchSnippetHandle * DiProcess::InsertSnippet (BPatch_snippet const & *expr*, BPatch_point & *point*)

Inserts a given snippet into the given point.

Parameters

expr	Snippet
point	Point where the Snippet will be inserted

Returns

Handle

4.34.3.5 BPatchSnippetHandle * DiProcess::InsertSnippet (BPatch_snippet const & *expr*, BPatch_point & *point*, BPatch_callWhen *when*, BPatch_snippetOrder *order*)

Inserts the snippet into the point in a given order.

Parameters

expr	Snippet
point	Point where the Snippet will be inserted
when	
order	

Returns

Handle

4.34.3.6 BPatchSnippetHandle * DiProcess::InsertSnippet (BPatch_snippet const & *expr*, PointVector & *points*)

Inserts the given snippet in the given points.

Parameters

expr	Snippet
points	Vector of points

Returns

Handle

4.34.3.7 BPatchSnippetHandle * DiProcess::InsertSnippet (BPatch_snippet const & *expr*, PointVector & *points*, BPatch_callWhen *when*, BPatch_snippetOrder *order*)

Inserts the given snippet in the series of points with the given order and moment.

Parameters

expr	Snippet
points	Vector of points
when	
order	

Returns

Handle

4.34.3.8 BPatchSnippetHandle * DiProcess::InsertSnippetAfter (BPatch_snippet const & *expr*, BPatch_point & *point*)

Inserts a given snippet after the given point.

Parameters

expr	Snippet
point	Point where the Snippet will be inserted

Returns

Handle

4.34.3.9 BPatchSnippetHandle * DiProcess::InsertSnippetAfter (BPatch_snippet const & *expr*, PointVector & *points*)

Inserts the snippet after the given points.

Parameters

expr	Snippet
points	Vector of points

Returns

Handle

4.34.3.10 BPatchSnippetHandle * DiProcess::InsertSnippetBefore (BPatch_snippet const & *expr*, BPatch_point & *point*)

Inserts a given snippet before the given point.

Parameters

expr	Snippet
point	Point where the Snippet will be inserted

Returns

Handle

4.34.3.11 BPatchSnippetHandle * DiProcess::InsertSnippetBefore (BPatch_snippet const & *expr,* PointVector & *points*)

Inserts the snippet before the given points.

Parameters

expr	Snippet
points	Vector of points

Returns

Handle

4.34.3.12 bool DiProcess::IsStopped ( ) [inline]

Asserts that _bpProcess has been created and returns a boolean.

Returns

bool

4.34.3.13 bool DiProcess::IsTerminated ( ) [inline]

Returns true if the mutatee process is terminated.

Returns

bool

4.34.3.14 void DiProcess::loadLibrary (char * *libName*)

Loads a shared library into the mutatee's address space. Returns true if successful.

Parameters

libName	Library name

4.34.3.15 BPatch_variableExpr* DiProcess::Malloc (BPatch_type & *type*) [inline]

Allocates a new variable of the given type.

Parameters

type	

4.34.3.16 void DiProcess::OneTimeCode (BPatch_snippet const & *expr*)

Executes the given snippet once.

Parameters

expr	Snippet

4.34.3.17 void DiProcess::ReplaceFunction (BPatch_function & *oldFunc,* BPatch_function & *newFunc*)

Replaces a function call with a call to another function.

Parameters

oldFunc	
newFunc	

4.34.3.18 bool DiProcess::StopExecution ( ) [inline]

Stops the execution of the mutatee process.

Returns

bool

4.34.3.19 bool DiProcess::Terminate ( ) [inline]

Terminates the mutatee process.

Returns

bool

The documentation for this class was generated from the following files:

- Common/di.h
- Common/di.cpp

4.35 DiSnippetHandle Class Reference

Dyninst's snippet handler class.

`#include <di.h>`

Public Member Functions

- DiSnippetHandle (BPatch_process &bpProcess, BPatch_snippet &snippet, BPatch_point &point, bool needDelete=false)
  Constructor.
- ~DiSnippetHandle ()
  Destructor.

4.35.1 Detailed Description

Dyninst's snippet handler class.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.35.2 Constructor & Destructor Documentation

4.35.2.1 DiSnippetHandle::DiSnippetHandle (BPatch_process & *bpProcess,* BPatch_snippet & *snippet,* BPatch_point & *point,* bool *needDelete* = `false`) [inline]

Constructor.

Parameters

bpProcess	BPatch process
snippet	Snipped of code
point	Point in the snippet
needDelete	Flag that states if the snipped has to be deleted. False by default

The documentation for this class was generated from the following file:

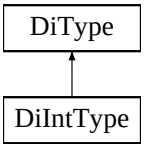
- Common/di.h

4.36 DiType Class Reference

Dyninst's Type class. It represents a variable or area of memory in a thread's address space.

`#include <di.h>`

Inheritance diagram for DiType:



Public Member Functions

- DiType (BPatch_image &bpImage, char const *typeName)

  Constructor.
- **operator BPatch_type &** ()

4.36.1 Detailed Description

Dyninst's Type class. It represents a variable or area of memory in a thread's address space.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.36.2 Constructor & Destructor Documentation

4.36.2.1 DiType::DiType (BPatch_image & *bpImage,* char const * *typeName*) [inline]

Constructor.

Parameters

bpImage	Image of the program
typeName	Name of the type

The documentation for this class was generated from the following file:

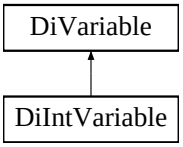
- Common/di.h

4.37 DiVariable Class Reference

Deals with Dyninst's Variable class. This can create, read and delete a variable of a given type or size in memory.

`#include <di.h>`

Inheritance diagram for DiVariable:



Public Member Functions

- DiVariable (BPatch_process &bpProcess, BPatch_type const &type)

  DiVariable constructor of a given type.
- DiVariable (BPatch_process &bpProcess, char const *typeName)

  DiVariable constructor of a given type name.
- DiVariable (BPatch_process &bpProcess, int size)

  DiVariable constructor of a given size.
- ~DiVariable ()

  Destructor.

- **operator BPatch_variableExpr &** ()
- **operator BPatch_variableExpr** ∗ ()
- void GetValue (void ∗dst) const

  Reads the value of a variable in a thread's address space. <dst> is assumed to be the same size as the variable.
- long int GetAddress () const

  Returns base address of this variable in the target's address space.

4.37.1 Detailed Description

Deals with Dyninst's Variable class. This can create, read and delete a variable of a given type or size in memory.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.37.2 Constructor & Destructor Documentation

4.37.2.1 DiVariable::DiVariable (BPatch_process & *bpProcess*, BPatch_type const & *type*) [inline]

DiVariable constructor of a given type.

Parameters

bpProcess	Process
type	Type of the variable

4.37.2.2 DiVariable::DiVariable (BPatch_process & *bpProcess*, char const ∗ *typeName*) [inline]

DiVariable constructor of a given type name.

Parameters

bpProcess	Process
typeName	Type name

4.37.2.3 DiVariable::DiVariable (BPatch_process & *bpProcess*, int *size*) [inline]

DiVariable constructor of a given size.

Parameters

bpProcess	Process
size	Size in bytes

4.37.3 Member Function Documentation

4.37.3.1 long int DiVariable::GetAddress ( ) const [inline]

Returns base address of this variable in the target's address space.

Returns

Base address of the variable

4.37.3.2 void DiVariable::GetValue (void * *dst*) const [inline]

Reads the value of a variable in a thread's address space. <dst> is assumed to be the same size as the variable.

Parameters

dst	Reads value from here

The documentation for this class was generated from the following file:

- Common/di.h

4.38 DTLibrary Class Reference

Dynamic Tuning Library that offers DT API. Encapsulates information about the application model and the event collector. Provides methods to create application models.

`#include <DTAPI.h>`

Public Member Functions

- Model::Application & CreateApplication (char const *appPath, int argc, char const **argv)
  Creates a new application model, a new event collector and associates them.
- Model::Application & GetApplication ()
  Application getter.

Friends

- class **DTLibraryFactory**

4.38.1 Detailed Description

Dynamic Tuning Library that offers DT API. Encapsulates information about the application model and the event collector. Provides methods to create application models.

4.38.2 Member Function Documentation

4.38.2.1 Model::Application & DTLibrary::CreateApplication (char const * *appPath*, int *argc*, char const ** *argv*)

Creates a new application model, a new event collector and associates them.

Parameters

appPath	path to the target application.
argc	number of arguments of the target application.
argv	list of arguments of the target application.

Returns

Reference to the application model object.

4.38.2.2 Model::Application & DTLibrary::GetApplication ( )

Application getter.

Returns

Reference to the application model.

The documentation for this class was generated from the following files:

- Analyzer/DTAPI.h
- Analyzer/DTAPI.cpp

4.39 DTLibraryFactory Class Reference

Handles the creation and destruction of DT Libraries.

`#include <DTAPI.h>`

Static Public Member Functions

- static DTLibrary * CreateLibrary (Config const &cfg)

  Creates and initializes the DT Library. Implements the singleton design pattern, so if the library is already created it returns a reference to it.
- static void DestroyLibrary (DTLibrary *lib)

  Destroys the library if there is only one last reference to the object.

4.39.1 Detailed Description

Handles the creation and destruction of DT Libraries.

4.39.2 Member Function Documentation

4.39.2.1 DTLibrary * DTLibraryFactory::CreateLibrary (Config const & *cfg*) [static]

Creates and initializes the DT Library. Implements the singleton design pattern, so if the library is already created it returns a reference to it.

Parameters

cfg	Reference to the configuration object.

Returns

Reference to the library.

4.39.2.2 void DTLibraryFactory::DestroyLibrary (DTLibrary * *lib*) [static]

Destroys the library if there is only one last reference to the object.

Parameters

lib	Reference to the library.

The documentation for this class was generated from the following files:

- Analyzer/DTAPI.h
- Analyzer/DTAPI.cpp

4.40 DynInst Class Reference

Assigns an instance of the class BPatch from Dyninst.

`#include <di.h>`

Static Public Member Functions

- static BPatch & **Instance** ()
- static BPatch & **Instance** (BPatch in_bp)

Static Protected Member Functions

- static void **OnError** (BPatchErrorLevel severity, int number, const char *const *params)

4.40.1 Detailed Description

Assigns an instance of the class BPatch from Dyninst.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

The documentation for this class was generated from the following files:

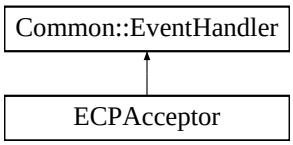
- Common/di.h
- Common/di.cpp

4.41 ECPAcceptor Class Reference

Event Acceptor class that collects incoming ECP events and prepares their correspondent handler.

`#include <EventCollector.h>`

Inheritance diagram for ECPAcceptor:



Public Member Functions

- ECPAcceptor (Reactor &reactor, int port)

  Constructor, starts listening to the socket and registers itself in the reactor.
- ~ECPAcceptor ()

  Destructor, unregister the object from the reactor.
- void HandleInput ()

  When a new connection is accepted, prepares a handler for it. It is registered in the reactor and added to the service.
- int GetHandle ()
- void SetEventCollector (EventCollector *collector)

  Setter for the event collector.

4.41.1 Detailed Description

Event Acceptor class that collects incoming ECP events and prepares their correspondent handler.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 ECPAcceptor::ECPAcceptor (Reactor & *reactor,* int *port*)

Constructor, starts listening to the socket and registers itself in the reactor.

Parameters

reactor	** Reactor of the application??? **
port	Socket port.

4.41.3 Member Function Documentation

4.41.3.1 int ECPAcceptor::GetHandle ( ) [inline],[virtual]

Returns

Reference to the handler object

Implements Common::EventHandler.

4.41.3.2 void ECPAcceptor::SetEventCollector (EventCollector * *collector*)

Setter for the event collector.

Parameters

collector	Event collector to be set.

The documentation for this class was generated from the following files:

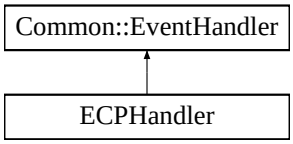
- Analyzer/EventCollector.h
- Analyzer/EventCollector.cpp

4.42 ECPHandler Class Reference

Encapsulates data structures and methods to handle incoming event collector inputs.

`#include <ECPHandler.h>`

Inheritance diagram for ECPHandler:



Public Member Functions

- ECPHandler (SocketPtr &socket, EventCollector *collector)

  Constructor.
- void Remove ()

  Not implemented (here for compatibility reasons)
- void HandleInput ()

  Reads an incoming message and handles it depending on its type. First reads a message from the socket, then creates the proper type of message and the calls the onEvent method of the listener of the events (if any).
- int GetHandle ()

  Handler getter.
- void SetService (Service *service)

  Service setter.

4.42.1 Detailed Description

Encapsulates data structures and methods to handle incoming event collector inputs.

4.42.2 Member Function Documentation

4.42.2.1 int ECPHandler::GetHandle ( ) [inline], [virtual]

Handler getter.

Returns

Reference to the handler object

Implements Common::EventHandler.

4.42.2.2 void ECPHandler::SetService (Service * *service*) [inline]

Service setter.

Parameters

service	Reference to the service.

The documentation for this class was generated from the following files:

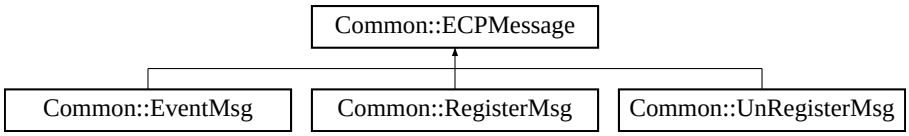
- Analyzer/ECPHandler.h
- Analyzer/ECPHandler.cpp

4.43 Common::ECPMessage Class Reference

Abstract class, EventCollectorProtocol, represents message interchanged between DMLib and analyzer.

`#include <ECPMsg.h>`

Inheritance diagram for Common::ECPMessage:



Public Member Functions

- virtual ECPMsgType GetType () const =0

  To be implemented by subclasses.
- virtual int GetDataSize () const

  Returns size of the data once serialized.
- virtual void Serialize (Serializer &out) const =0

  To be implemented by subclasses.
- virtual void DeSerialize (DeSerializer &in)=0

  To be implemented by subclasses.

Protected Member Functions

- ECPMessage ()

  Constructor.

4.43.1 Detailed Description

Abstract class, EventCollectorProtocol, represents message interchanged between DMLib and analyzer.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.43.2 Constructor & Destructor Documentation

4.43.2.1 Common::ECPMessage::ECPMessage ( ) [inline], [protected]

Constructor.

Protected so that this base class cannot be explicitly instantiated.

The documentation for this class was generated from the following files:

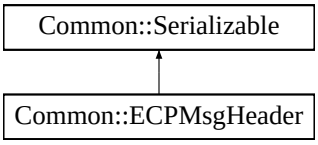
- Common/ECPMsg.h
- Common/ECPMsg.cpp

4.44 Common::ECPMsgHeader Class Reference

Represents header of an ECPMessage object.

`#include <ECPMsgHeader.h>`

Inheritance diagram for Common::ECPMsgHeader:



Public Member Functions

- ECPMsgHeader ()
  Constructor.
- void Serialize (Serializer &out) const
  Sends the message header.
- void DeSerialize (DeSerializer &in)
  Receives the message header.
- int GetMagic () const
  Returns magic attribute.
- int GetVersion () const
  Returns version attribute.
- ECPMsgType GetType () const
  Returns type of the message.
- int GetDataSize () const
  Returns data size.
- int GetHeaderSize () const
  Returns header size.
- void SetMagic (int magic)
  Sets magic attribute.
- void SetVersion (int version)
  Sets version attribute.
- void SetMsgType (ECPMsgType type)
  Sets type attribute.
- void SetDataSize (int size)
  Sets the data size attribute.
- void SetHeaderSize ()
  Updates header size.

4.44.1 Detailed Description

Represents header of an ECPMessage object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- Common/ECPMsgHeader.h
- Common/ECPMsgHeader.cpp

4.45 ECPProtocol Class Reference

Encapsulates methods to read and handle incoming network messages.

`#include <ECPProtocol.h>`

Static Public Member Functions

- static ECPMsgHeader ReadMessageHeader (Socket &sock)

  Reads a message header from the socket, deserializes it and creates a message header object.
- static ECPMessage * ReadMessageEx (Socket &sock)

  Reads a message from the socket, deserializes it and creates different kind of message objects depending on their type.

4.45.1 Detailed Description

Encapsulates methods to read and handle incoming network messages.

4.45.2 Member Function Documentation

4.45.2.1 ECPMessage * ECPProtocol::ReadMessageEx (Socket & *sock*) [static]

Reads a message from the socket, deserializes it and creates different kind of message objects depending on their type.

Parameters

sock	Reference to the socket.

Returns

Reference to the message object created.

4.45.2.2 ECPMsgHeader ECPProtocol::ReadMessageHeader (Socket & *sock*) [static]

Reads a message header from the socket, deserializes it and creates a message header object.

Parameters

sock	Reference to the socket.

Returns

Reference to the message header object created

The documentation for this class was generated from the following files:

- Analyzer/ECPProtocol.h
- Analyzer/ECPProtocol.cpp

4.46 Model::Event Class Reference

Encapsulates information about the events that the target application generates. For each event holds identification information (id, name), the place where it is produced, its attributes (for example the parameters of a function) and a reference to a handler. As this is a model class the methods provided are for accessing and setting the members of the data structure.

`#include <AppEvent.h>`

Public Member Functions

- Event (Event const &e)
  Copy constructor.
- Event (int id, std::string const &funcName, InstrPlace place)
  Constructor.
- ~Event ()
  Destructor.
- int GetId () const
  Globally unique event id getter.
- string GetFunctionName () const
  Name getter.
- InstrPlace GetInstrPlace () const
  Instruction place getter.
- int GetNumAttributes () const
  Number of attributes getter.
- Attribute * GetAttributes () const
  Attributes getter.
- void SetAttribute (int nAttrs, Attribute *attrs)
  Attributes setter.
- void SetNEvents (int nEvents)
  Attributes setter.
- void SetEventHandler (EventHandler &h)
  Installs a callback function that is called each time a record of this event is delivered.
- EventHandler * GetEventHandler ()
  Event handler getter.
- int GetNEvents () const
  Number of Papi metrics getter.
- int GetNumPapiMetrics () const
  Number of Papi metrics getter.
- std::string * **GetMetrics** () const
- void **SetMetric** (int nPapi, std::string *PapiMetrics)

4.46.1 Detailed Description

Encapsulates information about the events that the target application generates. For each event holds identification information (id, name), the place where it is produced, its attributes (for example the parameters of a function) and a reference to a handler. As this is a model class the methods provided are for accessing and setting the members of the data structure.

4.46.2 Constructor & Destructor Documentation

4.46.2.1 Event::Event (Event const & *e*)

Copy constructor.

Parameters

e	the event to copy

4.46.2.2 Event::Event (int *id,* std::string const & *funcName,* InstrPlace *place*)

Constructor.

Parameters

id	unique identification number for the event
funcName	name of the function which the event is associated to
place	place of the function

4.46.3 Member Function Documentation

4.46.3.1 Attribute* Model::Event::GetAttributes ( ) const [inline]

Attributes getter.

Returns

A collection of attributes to be recorded with this event

4.46.3.2 EventHandler* Model::Event::GetEventHandler ( ) [inline]

Event handler getter.

Returns

Event handler

4.46.3.3 string Model::Event::GetFunctionName ( ) const [inline]

Name getter.

Returns

Name of the function this event is associated to

4.46.3.4 int Model::Event::GetId ( ) const [inline]

Globally unique event id getter.

Returns

Event id

4.46.3.5 InstrPlace Model::Event::GetInstrPlace ( ) const [inline]

Instruction place getter.

Returns

Either the function entry or exit

4.46.3.6 int Model::Event::GetNEvents ( ) const [inline]

Number of Papi metrics getter.

Returns

Number of Papi metrics

4.46.3.7 int Model::Event::GetNumAttributes ( ) const [inline]

Number of attributes getter.

Returns

Number of event attributes

4.46.3.8 int Model::Event::GetNumPapiMetrics ( ) const [inline]

Number of Papi metrics getter.

Returns

Number of Papi metrics

4.46.3.9 void Event::SetAttribute (int *nAttrs*, Attribute * *attrs*)

Attributes setter.

Parameters

nAttrs	Number of attributes
attrs	Collection of attributes to be recorded with this event

4.46.3.10 void Event::SetEventHandler (EventHandler & *h*)

Installs a callback function that is called each time a record of this event is delivered.

Parameters

h	Event handler

4.46.3.11 void Event::SetNEvents (int *nEvents*)

Attributes setter.

Parameters

nAttrs	Number of attributes
attrs	Collection of attributes to be recorded with this event

The documentation for this class was generated from the following files:

- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

4.47 Common::Event Class Reference

Encapsulates information to record an event.

`#include <Event.h>`

Public Member Functions

- Event (long64_t timestamp, int eventId, EventPlace &place, int tid, int paramCount, std::string const &machine)

  Constructor.
- ~Event ()

  Destructor.
- long64_t GetTimestamp () const

  Returns timestamp.
- int GetPlace ()

  Returns place {EventEntry, EventExit}.
- int GetEventId () const

  Returns event id.
- int GetTid ()

  Returns tid attribute.
- int GetParamCount ()

  Returns count of the parameters.
- std::string const & GetMachine ()

  Returns name of the machine.

4.47.1 Detailed Description

Encapsulates information to record an event.

This information will be sent to the Analyzer, who will do the actual recording of the event attributes.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2004

4.47.2 Constructor & Destructor Documentation

4.47.2.1 Common::Event::Event (long64_t *timestamp,* int *eventId,* EventPlace & *place,* int *tid,* int *paramCount,* std::string const & *machine*) [inline]

Constructor.

Parameters

timestamp	Time stamp when the event was initialized.
eventId	Id of the event.
place	Part on the program where it'll take place. {EventEntry, EventExit}
tid	Task id.
paramCount	Number of parameters.
machine	String representing the machine where the event takes place.

The documentation for this class was generated from the following file:

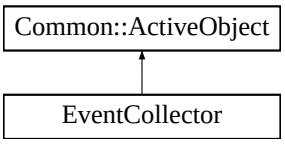
- Common/Event.h

4.48 EventCollector Class Reference

Processes the incoming event records from the DMLibs. It is based on an active object (thread) that collects incoming ECP events It stores a moving window of events incoming from different processes using a pool of buffers. The maximum size of this event window can be configured by the tunlets.

`#include <EventCollector.h>`

Inheritance diagram for EventCollector:



Public Types

- enum { **DefaultPort** }

Public Member Functions

- EventCollector (int port=DefaultPort)

  Constructor, starts an execution thread.
- ~EventCollector ()

  Destructor, stops the execution thread.
- void SetListener (EventListener *listener)

Setter for the listener.

- EventListener * GetListener ()

  Getter for the listener.

- bool IsAborted () const

  Determines if the collector is aborted.

Protected Member Functions

- void InitThread ()

  Not implemented (here for compatibility reasons).

- void Run ()

  Runner of the execution thread, handles events until it dies.

- void FlushThread ()

  Not implemented (here for compatibility reasons).

- void Fatal ()

  Called when an exception is caught in the execution thread.

Protected Attributes

- EventListener * **_listener**
- Reactor **_reactor**
- ECPAcceptor **_acceptor**
- bool **_aborted**

4.48.1 Detailed Description

Processes the incoming event records from the DMLibs. It is based on an active object (thread) that collects incoming ECP events It stores a moving window of events incoming from different processes using a pool of buffers. The maximum size of this event window can be configured by the tunlets.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 EventCollector::EventCollector (int *port* = `DefaultPort`)

Constructor, starts an execution thread.

Parameters

port	Acceptor port.

4.48.3 Member Function Documentation

4.48.3.1 EventListener* EventCollector::GetListener ( ) `[inline]`

Getter for the listener.

Returns

Listener of the event collector.

4.48.3.2 bool EventCollector::IsAborted ( ) const [inline]

Determines if the collector is aborted.

Returns

The status of the collector.

4.48.3.3 void EventCollector::SetListener (EventListener * *listener*)

Setter for the listener.

Parameters

listener	Listener to be set.

The documentation for this class was generated from the following files:

- Analyzer/EventCollector.h
- Analyzer/EventCollector.cpp

4.49 DMLib::EventCollectorProxy Class Reference

Connects to the analyzer host and sends requests.

`#include <ECPProxy.h>`

Public Member Functions

- EventCollectorProxy (std::string const &host, int port)

  Constructor.
- ~EventCollectorProxy ()

  Destructor.
- void RegisterLib (int pid, int mpiRank, std::string host, std::string taskName, int ACport)

  Sends a request to the Analyzer to register a new worker.
- void SendEvent (EventMsg const &event)

  Sends a message to the analyzer.
- void UnregisterLibrary (int pid)

  Sends a request to the Analyzer to unregister a worker.

4.49.1 Detailed Description

Connects to the analyzer host and sends requests.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- DMLib/ECPProxy.h
- DMLib/ECPProxy.cpp

4.50 Common::EventDemultiplexer Class Reference

Part of the reactor design pattern, takes requests coming from the reactor and passes them to different handlers.

`#include <Reactor.h>`

Public Member Functions

- EventDemultiplexer ()
  Constructor.
- void AddHandle (int handle)
  Adds a new handle.
- void RemoveHandle (int handle)
  Removes selected handle.
- int Select (TimeValue *timeout=0)
  Returns the number of socket handles ready or 0 if the time limit expired.
- bool IsHandleActivated (int handle) const
  Returns true if the given handle is activated, false otherwise.
- int GetMaxHandle () const
  Returns value of the max handle.

4.50.1 Detailed Description

Part of the reactor design pattern, takes requests coming from the reactor and passes them to different handlers.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.50.2 Member Function Documentation

4.50.2.1 int EventDemultiplexer::Select (TimeValue * *timeout* = 0)

Returns the number of socket handles ready or 0 if the time limit expired.

Parameters

timeout	If the parameter is a TimeValue object, it will wait the object value for events. In the event that the value is 0 it will check without blocking. If the parameter is a 0 (not a TimeValue object, default value) it will check and block in a forever loop.

The documentation for this class was generated from the following files:

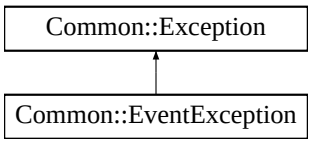
- Common/Reactor.h
- Common/Reactor.cpp

4.51 Common::EventException Class Reference

Event, EventMap and EventHandler exceptions.

`#include <EventException.h>`

Inheritance diagram for Common::EventException:



Public Member Functions

- EventException (std::string const &msg, std::string const &objName=std::string())

  Constructor.
- void Display (std::ostream &os) const

  Displays exception message on the given output stream.
- void Display () const

  Displays exception message on the standard error output.
- std::string GetReason () const

  Returns a string containing the error message.

Additional Inherited Members

4.51.1 Detailed Description

Event, EventMap and EventHandler exceptions.

Version

1.0b

Since

1.0b

Author

Noel De Martin, 2011

4.51.2 Constructor & Destructor Documentation

4.51.2.1 Common::EventException::EventException (std::string const & *msg,* std::string const & *objName* = `std::string ()`) `[inline]`

Constructor.

Parameters

msg	Exception message.
objName	Name of the object causing the exception, "" by default.

4.51.3 Member Function Documentation

4.51.3.1 void Common::EventException::Display (std::ostream & *os*) const [virtual]

Displays exception message on the given output stream.

Parameters

os	Output stream to display the message.

Reimplemented from Common::Exception.

4.51.3.2 string EventException::GetReason ( ) const

Returns a string containing the error message.

Returns

String with the error.

The documentation for this class was generated from the following files:

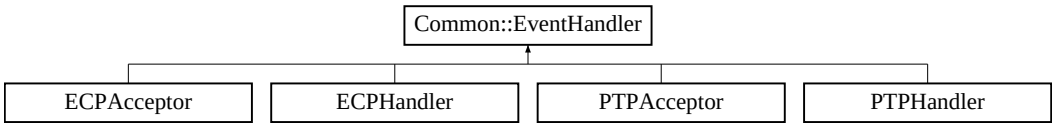
- Common/EventException.h
- Common/EventException.cpp

4.52 Common::EventHandler Class Reference

Abstract class, processes the requests sent to the reactor.

`#include <EventHandler.h>`

Inheritance diagram for Common::EventHandler:



Public Member Functions

- virtual void HandleInput ()=0

  Reads the data from the socket and treats it.
- virtual int GetHandle ()=0

  Returns socket descriptor.

4.52.1 Detailed Description

Abstract class, processes the requests sent to the reactor.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

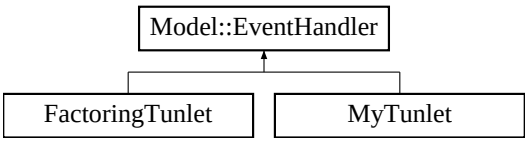
- Common/EventHandler.h

4.53 Model::EventHandler Class Reference

Abstract class that holds a method to manage event records.

`#include <AppEvent.h>`

Inheritance diagram for Model::EventHandler:



Public Member Functions

- virtual void HandleEvent (EventRecord const &r)=0

  Handles an event record (virtual).

4.53.1 Detailed Description

Abstract class that holds a method to manage event records.

4.53.2 Member Function Documentation

4.53.2.1 virtual void Model::EventHandler::HandleEvent (EventRecord const & *r*) [pure virtual]

Handles an event record (virtual).

Parameters

r	Event record to be handled

Implemented in FactoringTunlet, and MyTunlet.

The documentation for this class was generated from the following file:

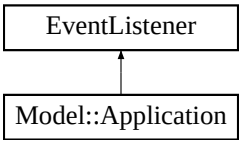
- Analyzer/AppEvent.h

4.54 EventListener Class Reference

Provides an interface for event listeners, which consist in methods to respond to events and errors.

`#include <EventCollector.h>`

Inheritance diagram for EventListener:



Public Member Functions

- virtual void OnEvent (ECPMessage ∗msg)=0

  Function which is triggered when an event happens.
- virtual void OnFatalError ()=0

  Function which is triggered when a fatal error happens.

4.54.1 Detailed Description

Provides an interface for event listeners, which consist in methods to respond to events and errors.

4.54.2 Member Function Documentation

4.54.2.1 virtual void EventListener::OnEvent (ECPMessage ∗ *msg*) [pure virtual]

Function which is triggered when an event happens.

Parameters

msg	Message that contains the event data.

Implemented in Model::Application.

The documentation for this class was generated from the following file:

- Analyzer/EventCollector.h

4.55 Common::EventMap Class Reference

Contains and manages a collection of Event objects.

`#include <EventMap.h>`

Public Member Functions

- EventMap ()

  Constructor.
- void Add (std::string const &name, int id)

  Adds a new event into the map.
- int GetId (std::string const &name) const

Returns id of the given event.

- int GetSize () const

  Returns map size.

4.55.1 Detailed Description

Contains and manages a collection of Event objects.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

4.55.2 Member Function Documentation

4.55.2.1 void EventMap::Add (std::string const & *name*, int *id*)

Adds a new event into the map.

Exceptions

EventException	

4.55.2.2 int EventMap::GetId (std::string const & *name*) const

Returns id of the given event.

Exceptions

EventException	

The documentation for this class was generated from the following files:

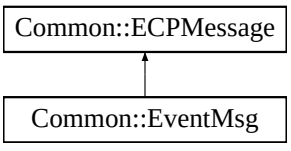
- Common/EventMap.h
- Common/EventMap.cpp

4.56 Common::EventMsg Class Reference

Encapsulates a message generated by DMLib to trace events.

`#include <ECPMsg.h>`

Inheritance diagram for Common::EventMsg:



Public Member Functions

- EventMsg ()

  Constructor.
- ~EventMsg ()

  Destructor.
- void Reset (long_t timestamp, int eventId, InstrPlace place, int paramCount)

  Sets the message to the indicated state.
- void SetTid (int tid)

  Sets the task id.
- ECPMsgType GetType () const

  Returns the type of event.
- void SetParams (char const *buffer, int size)

  Sets the buffer to be used and indicates its size.
- void SetBuffer (char *buffer)

  Sets the parameters buffer.
- int GetParamBufSize () const

  Returns buffer size.
- const char * GetParamBuffer () const

  Returns a pointer to the content of the buffer.
- long_t GetTimestamp () const

  Returns timestamp.
- int GetPlace () const

  Returns place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.
- int GetEventId () const

  Returns event ID.
- int GetParamCount () const

  Returns parameters count.
- int GetDataSize () const

  Returns size of the data serialized.
- void Serialize (Serializer &out) const

  Serializes the message with the given Serializer.
- void DeSerialize (DeSerializer &in)

  Deserializes the message with the given DeSerializer.
- int GetTid () const

  Returns the task id.

Additional Inherited Members

4.56.1 Detailed Description

Encapsulates a message generated by DMLib to trace events.

The message indicates what information should be gathered of certain event, this messages are created with an EventMsgWriter object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.56.2 Member Function Documentation

4.56.2.1 void EventMsg::Reset (long_t *timestamp*, int *eventId*, InstrPlace *place*, int *paramCount*)

Sets the message to the indicated state.

Parameters

timestamp	Timestamp when the event occurs.
eventId	Id of the event.
place	Place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.
paramCount	Number of parameters.

The documentation for this class was generated from the following files:

- Common/ECPMsg.h
- Common/ECPMsg.cpp

4.57 EventMsgReader Class Reference

Provides methods for getting data from event messages. The data structure that supports the class consist in the message to be processed, a buffer to hold the data and a deserializer object to reconstruct the information.

`#include <EventMsgReader.h>`

Public Member Functions

- EventMsgReader (EventMsg const &msg)

  Constructor.
- int GetParamCount () const

  Getter of ParamCount.
- AttrValueType GetAttrType ()

  getter of AttrType.
- int GetIntValue ()

  Gets an integer from the stream.
- float GetFloatValue ()

  Gets a float from the stream.
- double GetDoubleValue ()

  Gets a double from the stream.
- char GetCharValue ()

  Gets a character from the stream.
- short GetShortValue ()

  Gets a short from the stream.
- std::string GetStringValue ()

  Get a string from the stream.
- void DumpValues ()

  Gets the value of each ECP event parameter. For each parameter checks the type and use the proper getter.

4.57.1 Detailed Description

Provides methods for getting data from event messages. The data structure that supports the class consist in the message to be processed, a buffer to hold the data and a deserializer object to reconstruct the information.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 EventMsgReader::EventMsgReader (EventMsg const & *msg*) [inline]

Constructor.

Parameters

msg	input message.

4.57.3 Member Function Documentation

4.57.3.1 AttrValueType EventMsgReader::GetAttrType ( ) [inline]

getter of AttrType.

Returns

Type of the attribute.

4.57.3.2 char EventMsgReader::GetCharValue ( ) [inline]

Gets a character from the stream.

Returns

Character value.

4.57.3.3 double EventMsgReader::GetDoubleValue ( ) [inline]

Gets a double from the stream.

Returns

Double value.

4.57.3.4 float EventMsgReader::GetFloatValue ( ) [inline]

Gets a float from the stream.

Returns

Float value.

4.57.3.5 int EventMsgReader::GetIntValue ( ) [inline]

Gets an integer from the stream.

Returns

Integer value.

4.57.3.6 int EventMsgReader::GetParamCount ( ) const [inline]

Getter of ParamCount.

Returns

Number of parameters.

4.57.3.7 short EventMsgReader::GetShortValue ( ) [inline]

Gets a short from the stream.

Returns

Short value.

4.57.3.8 std::string EventMsgReader::GetStringValue ( ) [inline]

Get a string from the stream.

Returns

String value.

The documentation for this class was generated from the following files:

- Analyzer/EventMsgReader.h
- Analyzer/EventMsgReader.cpp

4.58 DMLib::EventMsgWriter Class Reference

Creates EventMsg objects.

`#include <EventMsgWriter.h>`

Public Member Functions

- EventMsgWriter ()
  Constructor.
- ~EventMsgWriter ()
  Destructor.
- void OpenEvent (long_t timestamp, int eventId, InstrPlace place, int paramCount)
  Open the event and sets its specifications.
- void AddIntParam (int value)
  Adds an integer parameter to the event.
- void AddFloatParam (float value)
  Adds a float parameter to the event.
- void AddDoubleParam (double value)
  Adds a double parameter to the event.
- void AddCharParam (char c)
  Adds a char parameter to the event.
- void AddStringParam (std::string const &s)
  Adds a string parameter to the event.
- EventMsg const & CloseEvent ()
  Closes the event and returns the object.

4.58.1 Detailed Description

Creates EventMsg objects.

Loads the specifications of an EventMsg object and prepares it. Once it's been prepared it returns the object using the CloseEvent method.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

4.58.2 Member Function Documentation

4.58.2.1 void EventMsgWriter::OpenEvent (long_t *timestamp*, int *eventId*, InstrPlace *place*, int *paramCount*)

Open the event and sets its specifications.

Parameters

timestamp	Timestamp when the event occurs.
eventId	Id of the event.
place	Place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.
paramCount	Number of parameters.

The documentation for this class was generated from the following files:

- DMLib/EventMsgWriter.h
- DMLib/EventMsgWriter.cpp

4.59 Model::EventRecord Class Reference

Particular instance of the event abstraction. Holds information about the kind of event, the task that produced, the message sent and the values it contained. On the one hand it provides methods to get/set the information above, on the other hand, it provides methods to parse messages and get the information that they contain.

`#include <AppEvent.h>`

Public Member Functions

- int GetEventId () const
  Id getter.
- Event const & GetEvent () const
  Associated event getter.
- long_t GetTimestamp () const
  Time stamp getter.
- Task & GetTask () const
  Task getter.
- AttributeValue * GetAttributeValues () const

Values getter.

- AttributeValue const & GetAttributeValue (int index) const

  Gets the i-th attribute from the list of values.

Protected Member Functions

- EventRecord (Event const &e, Task &t, EventMsg const &msg)

  Constructor.
- void ParseAttrs (EventMsg const &msg)

  Reads from the message and sets the value of the attributes depending on their type.

Friends

- class **Task**

4.59.1 Detailed Description

Particular instance of the event abstraction. Holds information about the kind of event, the task that produced, the message sent and the values it contained. On the one hand it provides methods to get/set the information above, on the other hand, it provides methods to parse messages and get the information that they contain.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 EventRecord::EventRecord (Event const & *e*, Task & *t*, EventMsg const & *msg*) [protected]

Constructor.

Parameters

e	Event object this record is associated to
t	Task object which produces the event
msg	Message produced by the event

4.59.3 Member Function Documentation

4.59.3.1 AttributeValue const& Model::EventRecord::GetAttributeValue (int *index*) const [inline]

Gets the i-th attribute from the list of values.

Parameters

index	Position of the attribute from which we want the value

Returns

The recorded value for the i-th attribute

4.59.3.2 AttributeValue* Model::EventRecord::GetAttributeValues ( ) const [inline]

Values getter.

Returns

A collection of recorded attribute values

4.59.3.3 Event const& Model::EventRecord::GetEvent ( ) const [inline]

Associated event getter.

Returns

Event object this record is associated to

4.59.3.4 int Model::EventRecord::GetEventId ( ) const [inline]

Id getter.

Returns

Globally unique event id

4.59.3.5 Task& Model::EventRecord::GetTask ( ) const [inline]

Task getter.

Returns

The task that generated this event

4.59.3.6 long_t Model::EventRecord::GetTimestamp ( ) const [inline]

Time stamp getter.

Returns

Time stamp that indicates when the event happened

4.59.3.7 void EventRecord::ParseAttrs (Common::EventMsg const & *msg*) [protected]

Reads from the message and sets the value of the attributes depending on their type.

Parameters

msg	Reference to the msg to be read.

The documentation for this class was generated from the following files:

- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

4.60 Model::Events Class Reference

Encapsulates information to create and manage events lists. Uses a data structure based on a vector to keep data and a map to retrieve it. Provides methods to add, remove and find elements in the list.

`#include <AppEvent.h>`

Public Member Functions

- Events ()

  Constructor.
- void Add (Event const &e)

  Maps and adds an event to the events list.
- bool Remove (int eventId, InstrPlace place)

  Removes an event from the events list.
- Event * Find (int eventId, InstrPlace place)

  Searches for an event in the event list.
- int Size () const

  Size getter.

4.60.1 Detailed Description

Encapsulates information to create and manage events lists. Uses a data structure based on a vector to keep data and a map to retrieve it. Provides methods to add, remove and find elements in the list.

4.60.2 Member Function Documentation

4.60.2.1 void Events::Add (Event const & *e*)

Maps and adds an event to the events list.

Parameters

e	The event to be added

4.60.2.2 Event * Events::Find (int *eventId*, InstrPlace *place*)

Searches for an event in the event list.

Parameters

eventId	Unique Id of the event.
place	Instruction where the event is placed

Returns

A reference to the found event or NULL if not found

4.60.2.3 bool Events::Remove (int *eventId*, InstrPlace *place*)

Removes an event from the events list.

Parameters

eventId	Unique Id of the event
place	Instruction where the event is placed

Returns

True if found &removed, false otherwise

4.60.2.4 int Events::Size ( ) const

Size getter.

Returns

Number of events

The documentation for this class was generated from the following files:

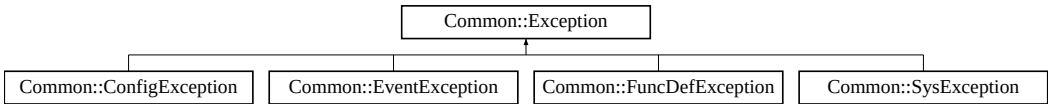
- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

4.61 Common::Exception Class Reference

Abstract class, stores information of errors on determined situations.

`#include <Exception.h>`

Inheritance diagram for Common::Exception:



Public Member Functions

- Exception (std::string const &msg, std::string const &objName=std::string(), long err=0)
  Constructor.
- Exception ()
  Constructor.
- virtual ~Exception ()
  Destructor.
- long GetError () const
  Returns error code.
- std::string const & GetErrorMessage () const
  Returns error message.
- std::string const & GetObjectName () const
  Returns the name of the object.
- virtual void Display () const
  Displays exception message on the standard error output.
- virtual void Display (std::ostream &os) const
  Displays exception message on the given output stream.

Protected Attributes

- long **_err**
- std::string **_msg**
- std::string **_objName**

4.61.1 Detailed Description

Abstract class, stores information of errors on determined situations.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.61.2 Constructor & Destructor Documentation

4.61.2.1 Common::Exception::Exception (std::string const & *msg*, std::string const & *objName* = `std::string ()`, long *err* = `0`) `[inline]`

Constructor.

Parameters

msg	Exception message.
objName	Name of the object causing the exception, "" by default.

4.61.3 Member Function Documentation

4.61.3.1 virtual void Common::Exception::Display (std::ostream & *os*) const `[virtual]`

Displays exception message on the given output stream.

Parameters

os	Output stream to display the message.

Reimplemented in Common::SysException, Common::EventException, Common::ConfigException, and Common::FuncDefException.

The documentation for this class was generated from the following files:

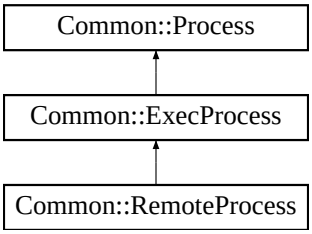
- Common/Exception.h
- Common/Exception.cpp

4.62 Common::ExecProcess Class Reference

Executes a program as a child of the current process.

`#include <Process.h>`

Inheritance diagram for Common::ExecProcess:



Public Types

- enum **Status** {
  stOutReady, **stOutEof**, **stErrReady**, **stErrEof**,
  stTimeout }

Public Member Functions

- ExecProcess (std::string const &programPath, char ∗const argv[])

  Constructor.
- void Start ()

  Executes the program.
- Status WaitForEvent (char ∗buffer, int bufSize, int &bytesRead, TimeValue ∗timeout=0)

  Waits until an event is placed on any of the outputs of the process or the time limit is reached.

Protected Member Functions

- ExecProcess ()

  Constructor.
- int Run ()

  Executes de process.

Additional Inherited Members

4.62.1 Detailed Description

Executes a program as a child of the current process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

4.62.2 Constructor & Destructor Documentation

4.62.2.1 Common::ExecProcess::ExecProcess (std::string const & *programPath,* char *const *argv[]*) [inline]

Constructor.

Example usage:

```
char * argv [] = { "/usr/bin/vi", "param1", "param2", 0 };
ExecProcess p ("/usr/bin/vi", argv);
```

Notes:

- first element of argv must be program path
- last element of argv table must be 0

Parameters

programPath	Path of the program to execute.
argv	Arguments to pass to the execution of the program.

4.62.3 Member Function Documentation

4.62.3.1 void ExecProcess::Start ( ) [virtual]

Executes the program.

The standard outputs and inputs of the program will be redirected to internal pipes, to be handled on the WaitForEvent() method.

Reimplemented from Common::Process.

4.62.3.2 ExecProcess::Status ExecProcess::WaitForEvent (char * *buffer,* int *bufSize,* int & *bytesRead,* TimeValue * *timeout* = 0)

Waits until an event is placed on any of the outputs of the process or the time limit is reached.

If any event was sent by the process, it is placed on the buffer.

Returns

Information about how the function ended and what was placed on the buffer.

The documentation for this class was generated from the following files:

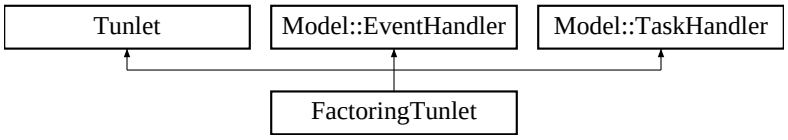
- Common/Process.h
- Common/Process.cpp

4.63 FactoringTunlet Class Reference

Factoring optimization tunlet for m/w apps.

`#include <FactoringTunlet_nw.h>`

Inheritance diagram for FactoringTunlet:



Public Member Functions

- FactoringTunlet ()

  Constructor.
- ~FactoringTunlet ()

  Destructor.
- void Initialize (Model::Application &app)
- void BeforeAppStart ()

  Asserts that _app != 0 and sets the task handler of the app to the current one.
- void Destroy ()

  Sets _app = 0.
- void HandleEvent (Model::EventRecord const &r)

  Handles all incoming events.
- void TaskStarted (Model::Task &t)

  Increments the number of workers by 1 and inserts an event to the worker.
- void TaskTerminated (Model::Task &t)

  Decrements the number of workers by 1.
- void CreateEvent ()

  Creates events by using the configuration specified in the file tunlet.ini.
- void CreateEvent (Model::Task &t)

  Creates events by using the configuration specified in the file tunlet.ini and a given task.

4.63.1 Detailed Description

Factoring optimization tunlet for m/w apps.

4.63.2 Member Function Documentation

4.63.2.1 void FactoringTunlet::CreateEvent (Model::Task & *t*)

Creates events by using the configuration specified in the file tunlet.ini and a given task.

Parameters

t	Task

4.63.2.2 void FactoringTunlet::HandleEvent (Model::EventRecord const & *r*) [virtual]

Handles all incoming events.

Parameters

r	EventRecord where its ID can be found

Implements Model::EventHandler.

4.63.2.3 void FactoringTunlet::Initialize (Model::Application & *app*) [virtual]

Parameters

app	

Implements Tunlet.

4.63.2.4 void FactoringTunlet::TaskStarted (Model::Task & *t*) [virtual]

Increments the number of workers by 1 and inserts an event to the worker.

Parameters

t	Task

Implements Model::TaskHandler.

4.63.2.5 void FactoringTunlet::TaskTerminated (Model::Task & *t*) [virtual]

Decrements the number of workers by 1.

Parameters

t	Task

Implements Model::TaskHandler.

The documentation for this class was generated from the following files:

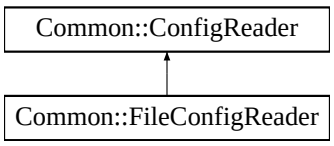
- Analyzer/FactoringTunlet_nw.h
- Analyzer/FactoringTunlet_nw-XFire.cpp
- Analyzer/FactoringTunlet_nw.cpp

4.64 Common::FileConfigReader Class Reference

Parses the content of a file into a Config object.

`#include <ConfigReader.h>`

Inheritance diagram for Common::FileConfigReader:



Public Member Functions

- FileConfigReader (std::string const &fileName)

  Constructor.
- Config Read ()

Parses the configuration of the file into a Config object.

Additional Inherited Members

4.64.1 Detailed Description

Parses the content of a file into a Config object.

Extends ConfigReader

Version

1.0b

Since

1.0b

Author

Noel De Martin, 2011

4.64.2 Constructor & Destructor Documentation

4.64.2.1 Common::FileConfigReader::FileConfigReader (std::string const & *fileName*) [inline]

Constructor.

Parameters

fileName	Path of the file to read.

Exceptions

ConfigException	

4.64.3 Member Function Documentation

4.64.3.1 Config FileConfigReader::Read ( ) [virtual]

Parses the configuration of the file into a Config object.

Exceptions

ConfigException	

Implements Common::ConfigReader.

The documentation for this class was generated from the following files:

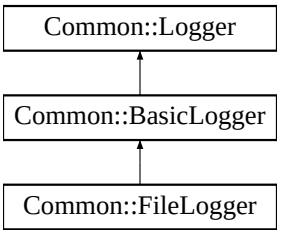
- Common/ConfigReader.h
- Common/ConfigReader.cpp

4.65 Common::FileLogger Class Reference

Stores information of interest into a file.

```
#include <Syslog.h>
```

Inheritance diagram for Common::FileLogger:



Public Member Functions

- FileLogger (std::string const &filepath, bool append=false)

  Constructor.
- ~FileLogger ()

  Destructor.
- void Log (LogEntry const &entry)

  Inserts a new entry to the log.

Additional Inherited Members

4.65.1 Detailed Description

Stores information of interest into a file.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.65.2 Constructor & Destructor Documentation

4.65.2.1 FileLogger::FileLogger (std::string const & *filepath*, bool *append* = `false`)

Constructor.

Parameters

filepath	Path of the file where the log will be stored. append Flag that determines if the file will be overwritten or the logs will be appended, default false.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/Syslog.h
- Common/Syslog.cpp

4.66 Common::FuncDef Class Reference

Represents definition of the function to be traced.

`#include <FuncDefs.h>`

Public Member Functions

- FuncDef (std::string const &name, std::string const &paramFormat, int paramCount, int funcId)

  Constructor.
- std::string const & GetName () const

  Returns name of the function.
- std::string const & GetParamFormat () const

  Returns format of the parameters.
- int GetParamCount () const

  Returns number of parameters used by the function.
- int GetFuncId () const

  Returns Id of the function.

4.66.1 Detailed Description

Represents definition of the function to be traced.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.66.2 Constructor & Destructor Documentation

4.66.2.1 Common::FuncDef::FuncDef (std::string const & *name*, std::string const & *paramFormat*, int *paramCount*, int *funcId*) [inline]

Constructor.

Parameters

name	Name of the function
paramFormat	String denoting types of the parameters. S: String, I: Integer, P: Pointer.
paramCount	Number of parameters.
funcId	Function Id.

The documentation for this class was generated from the following file:

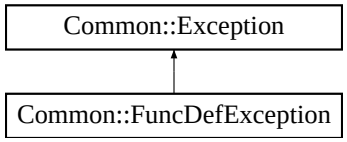
- Common/FuncDefs.h

4.67 Common::FuncDefException Class Reference

FuncDef exceptions.

`#include <FuncDefException.h>`

Inheritance diagram for Common::FuncDefException:



Public Member Functions

- FuncDefException (std::string const &msg, std::string const &objName=std::string())

  Constructor.
- void Display (std::ostream &os) const

  Displays exception message on the given output stream.
- void Display () const

  Displays exception message on the standard error output.
- std::string GetReason () const

  Returns a string containing the error message.

Additional Inherited Members

4.67.1 Detailed Description

FuncDef exceptions.

Version

1.0b

Since

1.0b

Author

Noel De Martin, 2011

4.67.2 Constructor & Destructor Documentation

4.67.2.1 Common::FuncDefException::FuncDefException (std::string const & *msg,* std::string const & *objName* = `std::string ()`) `[inline]`

Constructor.

Parameters

msg	Exception message.
objName	Name of the object causing the exception, "" by default.

4.67.3 Member Function Documentation

4.67.3.1 void FuncDefException::Display (std::ostream & *os*) const [virtual]

Displays exception message on the given output stream.

Parameters

os	Output stream to display the message.

Reimplemented from Common::Exception.

4.67.3.2 string FuncDefException::GetReason ( ) const

Returns a string containing the error message.

Returns

String with the error.

The documentation for this class was generated from the following files:

- Common/FuncDefException.h
- Common/FuncDefException.cpp

4.68 Common::FuncDefs Class Reference

Creates and stores objects of the FuncDef class.

`#include <FuncDefs.h>`

Public Member Functions

- FuncDefs ()
  Constructor.
- void Add (std::string const &funcName, std::string const &paramFormat, int paramCount, int funcId)
  Adds a FuncDef object.
- FuncDef const & Find (std::string const &name)
  Returns a FuncDef object with the given name.
- int GetSize () const
  Returns number of FuncDef objects stored.

4.68.1 Detailed Description

Creates and stores objects of the FuncDef class.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.68.2 Constructor & Destructor Documentation

4.68.2.1 FuncDefs::FuncDefs ( )

Constructor.

Exceptions

FuncDefException	

4.68.3 Member Function Documentation

4.68.3.1 void FuncDefs::Add (std::string const & *funcName,* std::string const & *paramFormat,* int *paramCount,* int *funcId*)

Adds a FuncDef object.

Uses the default constructor of FuncDef with the given parameters.

Exceptions

FuncDefException	

4.68.3.2 FuncDef const & FuncDefs::Find (std::string const & *name*)

Returns a FuncDef object with the given name.

Exceptions

FuncDefException	

The documentation for this class was generated from the following files:

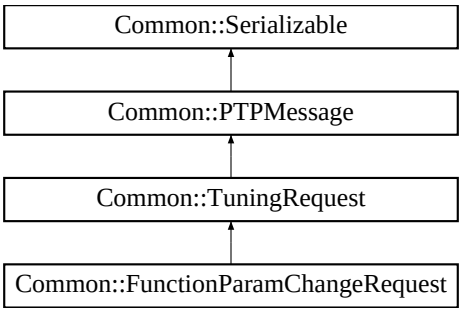
- Common/FuncDefs.h
- Common/FuncDefs.cpp

4.69 Common::FunctionParamChangeRequest Class Reference

Encapsulates a tuning request to set the value of an input parameter of a given function in a given application process.

`#include <PTPMsg.h>`

Inheritance diagram for Common::FunctionParamChangeRequest:



Public Member Functions

- FunctionParamChangeRequest (int pid=0, std::string const &funcName=std::string(), int paramIdx=0, int newValue=0, int *requiredOldValue=0, Breakpoint *brkpt=0)

  Constructor.
- PTPMsgType GetType () const

  Returns type of message (PTPFuncParamChange).
- std::string const & GetFuncName () const

  Returns name of the function.
- int GetParamIdx () const

  Returns index of the parameter to change on the attributes array.
- int GetNewValue () const

  Returns the new value to replace on the function call.
- int const * GetReqOldValue () const

  Returns the value the parameter should have for the tuning to be performed.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.69.1 Detailed Description

Encapsulates a tuning request to set the value of an input parameter of a given function in a given application process.

This parameter value is modified before the function body is invoked. It's also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to a new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.69.2 Constructor & Destructor Documentation

4.69.2.1 Common::FunctionParamChangeRequest::FunctionParamChangeRequest (int *pid* = `0`, std::string const & *funcName* = `std::string()`, int *paramIdx* = `0`, int *newValue* = `0`, int * *requiredOldValue* = `0`, Breakpoint * *brkpt* = `0`) [inline]

Constructor.

Parameters

pid	Id of the process where the parameter will be changed, default 0.
funcName	Name of the function call to modify, default "".
paramIdx	Parameter index inside the attributes array, default 0.
newValue	New Value to set, default 0.
requiredOld-Value	Current value the parameter should have to perform the tuning. If the value doesn't match this one, the tuning won't be performed. If this value is 0, the tuning will be performed without checking the old value, default 0.
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.70 Common::HandlerMap Class Reference

Contains and manages a collection of EventHandler objects.

`#include <Reactor.h>`

Public Member Functions

- HandlerMap ()

  Constructor.
- void Add (int handle, EventHandler ∗handler)

  Adds the handler to the map.
- EventHandler ∗ Get (int handle)

  Returns the EventHandler object stored with the given handle.
- int GetSize () const

  Returns map size.

4.70.1 Detailed Description

Contains and manages a collection of EventHandler objects.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- Common/Reactor.h
- Common/Reactor.cpp

4.71 Model::Host Class Reference

Encapsulates host information. Basically consists in a string with the name of the host and a method to access it.

`#include <Host.h>`

Public Member Functions

- string GetName () const

Protected Member Functions

- Host (string const &name)
  Constructor.

Friends

- class **Application**

4.71.1 Detailed Description

Encapsulates host information. Basically consists in a string with the name of the host and a method to access it.

4.71.2 Member Function Documentation

4.71.2.1 string Model::Host::GetName ( ) const [inline]

Returns

Name of the host

The documentation for this class was generated from the following file:

- Analyzer/Host.h

4.72 Model::HostHandler Class Reference

Provides mechanisms to handle the addition and removing of hosts.

`#include <Host.h>`

Public Member Functions

- virtual void HostAdded (Host &h)=0
  Called when a new host is added to the virtual machine.
- virtual void HostRemoved (Host &h)=0
  Called when a host is removed from the virtual machine.

4.72.1 Detailed Description

Provides mechanisms to handle the addition and removing of hosts.

4.72.2 Member Function Documentation

4.72.2.1 virtual void Model::HostHandler::HostAdded (Host & *h*) [pure virtual]

Called when a new host is added to the virtual machine.

Parameters

h	Added host.

4.72.2.2 virtual void Model::HostHandler::HostRemoved (Host & *h*) [pure virtual]

Called when a host is removed from the virtual machine.

Parameters

h	Removed host.

The documentation for this class was generated from the following file:

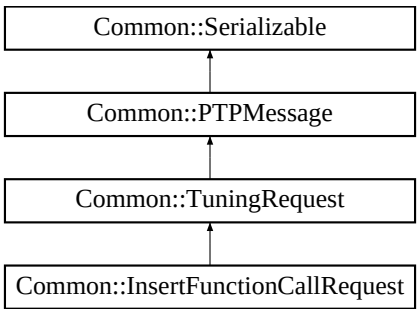
- Analyzer/Host.h

4.73 Common::InsertFunctionCallRequest Class Reference

Encapsulates a tuning request to insert a new function invocation code with a specified attributes at a given location in an application process.

`#include <PTPMsg.h>`

Inheritance diagram for Common::InsertFunctionCallRequest:



Public Member Functions

- InsertFunctionCallRequest (int pid=0, std::string const &funcName=std::string(), int nAttrs=0, Attribute *attrs=0, std::string const &destFunc=std::string(), InstrPlace place=ipFuncEntry, Breakpoint *brkpt=0)

  Constructor.
- ~InsertFunctionCallRequest ()

  Destructor.
- PTPMsgType GetType () const

  Returns type of message (PTPInsertFuncCall).
- std::string const & GetFuncName () const

  Returns name of the function to add.
- int GetAttrCount () const

  Returns number of attributes the function has.

- Attribute * GetAttributes () const

  Returns array of attributes.
- std::string const & GetDestFunc () const

  Returns name of the function where the call will be added.
- InstrPlace GetInstrPlace () const

  Returns the place where the call will be added.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.73.1 Detailed Description

Encapsulates a tuning request to insert a new function invocation code with a specified attributes at a given location in an application process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.73.2 Constructor & Destructor Documentation

4.73.2.1 Common::InsertFunctionCallRequest::InsertFunctionCallRequest (int *pid* = `0`, std::string const & *funcName* = `std::string()`, int *nAttrs* = `0`, Attribute * *attrs* = `0`, std::string const & *destFunc* = `std::string()`, InstrPlace *place* = `ipFuncEntry`, Breakpoint * *brkpt* = `0`) `[inline]`

Constructor.

Parameters

pid	Id of the process where the call will be inserted, default 0.
funcName	Name of the function to call, default "".
nAttrs	Number of attributes the function has, default 0.
attrs	Attribute array, default 0.
destFunc	Function where the call will be inserted, default "".
place	Place where the call will be added, default ipFuncEntry.
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.74 InstrGroup Class Reference

Contains a group of snippets to be inserted in a function.

`#include <InstrSet.h>`

Public Types

- typedef vector< SnippetHandler * >
  ::iterator **Iterator**

Public Member Functions

- InstrGroup (int eventId, std::string const &funcName)
  Constructor.
- ~InstrGroup ()
  Destructor.
- int GetEventId () const
  Getter for the variable _eventId.
- int GetSize () const
  Getter of the size of _vector.
- bool IsEmpty () const
  Checks if _vector is empty.
- std::string const & GetFuncName () const
  Getter of the name of the function.
- void AddHandler (InstrPlace place, BPatchSnippetHandle *handle)
  Add the handler passed as a parameter to the _vector.
- void RemoveHandler (InstrPlace place)
  Eliminates the handlers from the vector to be inserted in the place passed as a parameter.
- Iterator begin ()
  Getter for an iterator pointing to the first element in the InstrGroup.
- Iterator end ()
  Getter for an iterator pointing to the last instruction (handler) on the group.

4.74.1 Detailed Description

Contains a group of snippets to be inserted in a function.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.74.2 Member Function Documentation

4.74.2.1 void InstrGroup::AddHandler (InstrPlace *place*, BPatchSnippetHandle * *handle*)

Add the handler passed as a parameter to the _vector.

Parameters

place	Object that represents the place in the program in which the snippet will be inserted.
handle	Object of the class BPatchSnippetHandle that handles a Dyninst snippet.

4.74.2.2 Iterator InstrGroup::begin ( ) [inline]

Getter for an iterator pointing to the first element in the InstrGroup.

Returns

Iterator for the variable _vector that points to its beginning.

4.74.2.3 Iterator InstrGroup::end ( ) [inline]

Getter for an iterator pointing to the last instruction (handler) on the group.

Returns

Iterator for the variable _vector that points to its final element.

4.74.2.4 int InstrGroup::GetEventId ( ) const [inline]

Getter for the variable _eventId.

Returns

Id of the event.

4.74.2.5 std::string const& InstrGroup::GetFuncName ( ) const [inline]

Getter of the name of the function.

Returns

String that contains the name of the function.

4.74.2.6 int InstrGroup::GetSize ( ) const [inline]

Getter of the size of _vector.

Returns

Size of the vector _vector.

4.74.2.7 bool InstrGroup::IsEmpty ( ) const [inline]

Checks if _vector is empty.

Returns

0 if not empty, 1 if empty.

4.74.2.8 void InstrGroup::RemoveHandler (InstrPlace *place*)

Eliminates the handlers from the vector to be inserted in the place passed as a parameter.

Parameters

place	Object that represents the place in the program in which the snippet will be inserted.

The documentation for this class was generated from the following files:

- AC/InstrSet.h
- AC/InstrSet.cpp

4.75 Common::ConfigMap::Iterator Class Reference

Iterates over a ConfigMap object.

`#include <ConfigMap.h>`

Public Member Functions

- Iterator (ConfigMap const &map)

  Constructor.
- bool AtEnd () const

  Indicates whether the iterator is pointing to the end of the map or not.
- void Next ()

  The pointer increases a position on the map.
- std::string GetSection () const

  Returns section of the current position.
- std::string GetKey () const

  Returns key of the current position.
- std::string const & GetValue () const

  Returns value of the current position.

4.75.1 Detailed Description

Iterates over a ConfigMap object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2000

The documentation for this class was generated from the following files:

- Common/ConfigMap.h
- Common/ConfigMap.cpp

4.76 IterData Class Reference

Statistics for a single iteration.

`#include <FactoringStats_nw.h>`

Public Member Functions

- IterData (int iterIdx)

  Constructor.
- ~IterData ()

  Destructor.
- void OnIterStart (long_t time, int numTuples, int sizeBytes, int nw)

  Sets the flag of the iteration started to 1, the iteration starting time to time, the number of tasks as numTuples, the number of workers to nw and the size in Bytes to sizeBytes.
- void OnIterEnd (long_t time)

  Sets the flag of the finishing iteration to 1 and makes sure that the final time stated in time is greater than the starting one. Finally, it computes the iteration's elapsed time.
- void OnNewBatch ()

  Increments the number of batches by 1.
- BatchData & GetBatchData (int IdxBatch)

  Gets the data in a batch by specifying the batch ID. If it's the last node from the iterator, adds a new BatchData object and returns it.
- bool IsComplete () const

  Checks if the current iteration has a start and an end and whether all batches are complete.
- bool AreBatchsComplete () const

  Checks if all batches are complete.
- int GetTupleSizeInBytes () const

  Getter of the tuple size in bytes.
- BatchData ** AllocBatchsArray ()

  Allocates a new batch data for each element in the array _mapB.
- int GetNumWorkers ()

  Getter of the number of workers.
- int GetTotalTasks () const

  Getter of the number of tasks.
- int GetNumBatchs () const

  Getter of the number of batches.

4.76.1 Detailed Description

Statistics for a single iteration.

4.76.2 Constructor & Destructor Documentation

4.76.2.1 IterData::IterData (int *iterIdx*)

Constructor.

Parameters

iterIdx	ID of the iterator

4.76.3 Member Function Documentation

4.76.3.1 BatchData ** IterData::AllocBatchsArray ( )

Allocates a new batch data for each element in the array _mapB.

Returns

Array with all batches allocated

4.76.3.2 bool IterData::AreBatchsComplete ( ) const

Checks if all batches are complete.

Returns

True if all batches are complete and False if not.

4.76.3.3 BatchData & IterData::GetBatchData (int *IdxBatch*)

Gets the data in a batch by specifying the batch ID. If it's the last node from the iterator, adds a new BatchData object and returns it.

Parameters

IdxBatch	

Returns

Data of the batch in a BatchData object

4.76.3.4 int IterData::GetNumBatchs ( ) const [inline]

Getter of the number of batches.

Returns

_nbatchs

4.76.3.5 int IterData::GetNumWorkers ( ) [inline]

Getter of the number of workers.

Returns

_numWorkers

4.76.3.6 int IterData::GetTotalTasks ( ) const [inline]

Getter of the number of tasks.

Returns

_numTasks

4.76.3.7 int IterData::GetTupleSizeInBytes ( ) const [inline]

Getter of the tuple size in bytes.

Returns

_sizeBytes

4.76.3.8 bool IterData::IsComplete ( ) const [inline]

Checks if the current iteration has a start and an end and whether all batches are complete.

Returns

True if the current iteration is complete or False if not

4.76.3.9 void IterData::OnIterEnd (long_t *time*)

Sets the flag of the finishing iteration to 1 and makes sure that the final time stated in *time* is greater than the starting one. Finally, it computes the iteration's elapsed time.

Parameters

time	Ending time of the iteration.

4.76.3.10 void IterData::OnIterStart (long_t *time*, int *numTuples*, int *sizeBytes*, int *nw*)

Sets the flag of the iteration started to 1, the iteration starting time to *time*, the number of tasks as *numTuples*, the number of workers to *nw* and the size in Bytes to *sizeBytes*.

Parameters

time	Starting time of the iteration in milliseconds
numTuples	Number of tasks
sizeBytes	Task size in Bytes
nw	Number of workers

The documentation for this class was generated from the following files:

- Analyzer/FactoringStats_nw.h
- Analyzer/FactoringStats_nw.cpp

4.77 Common::Config::KeyIterator Class Reference

Iterates over the keys of a Config object.

`#include <Config.h>`

Public Member Functions

- KeyIterator (Config const &config, std::string const &section)

  Constructor.
- bool AtEnd () const

  Indicates whether the iterator is pointing to the end of the map or not.
- void Next ()

  The pointer increases a position on the config.
- std::string GetKey () const

  Returns key of the current position.
- std::string const & GetValue () const

  Returns value of the current position.
- int GetIntValue () const

  Returns integer value of the current position.

4.77.1 Detailed Description

Iterates over the keys of a Config object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2000

The documentation for this class was generated from the following files:

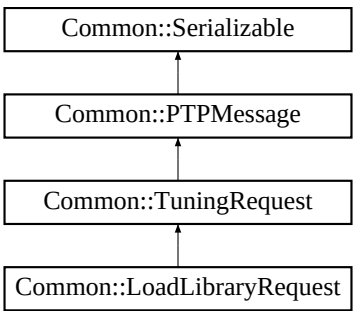
- Common/Config.h
- Common/Config.cpp

4.78 Common::LoadLibraryRequest Class Reference

Encapsulates a tuning request to load the specified shared library to a given application process.

`#include <PTPMsg.h>`

Inheritance diagram for Common::LoadLibraryRequest:



Public Member Functions

- **_libPath** (libPath)
- PTPMsgType GetType () const

  Returns type of message (PTPLoadLibrary).
- std::string const & GetLibraryPath () const

  Returns the path of the library to be loaded.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Public Attributes

- __pad0__: TuningRequest (pid

  Constructor.

Additional Inherited Members

4.78.1 Detailed Description

Encapsulates a tuning request to load the specified shared library to a given application process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.78.2 Member Data Documentation

4.78.2.1 Common::LoadLibraryRequest::__pad0__

Constructor.

Parameters

pid	Id of the process where the library will be included, default 0.
libPath	Path of the library, default "".

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.79 Common::LogEntry Class Reference

Entry on a log.

`#include <Syslog.h>`

Public Member Functions

- LogEntry (LogSeverity s, std::string const &message)

  Constructor.
- DateTime const & GetTimestamp () const

  Returns the date when the entry was performed.
- LogSeverity GetSeverity () const

  Returns log severity.
- std::string const & GetMessage () const

  Returns a string containing the log message.

4.79.1 Detailed Description

Entry on a log.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.79.2 Constructor & Destructor Documentation

4.79.2.1 Common::LogEntry::LogEntry (LogSeverity *s*, std::string const & *message*) [inline]

Constructor.

Parameters

s	Log severity, can be DEBUG, INFO, WARNING, ERROR or FATAL.
message	Message.

4.79.3 Member Function Documentation

4.79.3.1 LogSeverity Common::LogEntry::GetSeverity ( ) const [inline]

Returns log severity.

DEBUG A log generated during debugging of the software. INFO An informational message. WARNING A warning message that the system administrator might want to know about ERROR One of the software components caused an error or exception. FATAL One of the software components is no longer functional.

The documentation for this class was generated from the following file:

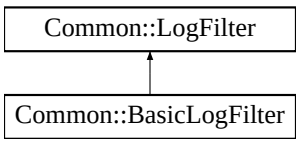
- Common/Syslog.h

4.80 Common::LogFilter Class Reference

Abstract class, validates logs.

`#include <Syslog.h>`

Inheritance diagram for Common::LogFilter:



Public Member Functions

- virtual ~LogFilter ()

  Constructor.
- virtual bool Accept (LogEntry const &entry) const =0

  Filters log entry.

4.80.1 Detailed Description

Abstract class, validates logs.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.80.2 Member Function Documentation

4.80.2.1 virtual bool Common::LogFilter::Accept (LogEntry const & *entry*) const [pure virtual]

Filters log entry.

Returns

True if entry is accepted, false otherwise.

Implemented in Common::BasicLogFilter.

The documentation for this class was generated from the following file:

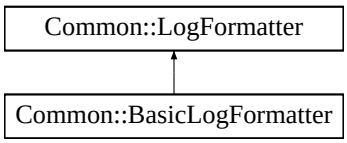
- Common/Syslog.h

4.81 Common::LogFormatter Class Reference

Abstract class, Gives logs the correct format.

`#include <Syslog.h>`

Inheritance diagram for Common::LogFormatter:



Public Member Functions

- virtual ~LogFormatter ()

  Destructor.
- virtual std::string GetLogHeader () const =0

  Returns a string containing the log header.
- virtual std::string GetLogFooter () const =0

  Returns a string containing the log footer.
- virtual std::string Format (LogEntry const &entry) const =0

  Returns a string containing the LogEntry object formatted.

4.81.1 Detailed Description

Abstract class, Gives logs the correct format.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

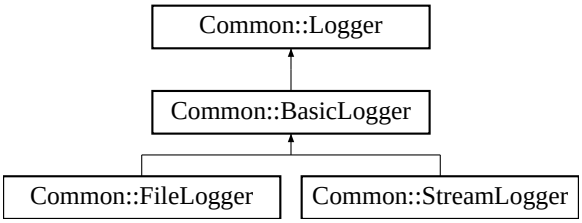
- Common/Syslog.h

4.82 Common::Logger Class Reference

Abstract class, tracks and stores information about events of interest happening in a system.

`#include <Syslog.h>`

Inheritance diagram for Common::Logger:



Public Member Functions

- virtual ~Logger ()

  Destructor.
- virtual void Log (LogEntry const &entry)=0

  Constructor.
- void SetName (std::string const &name)

  Sets logger name.
- std::string const & GetName () const

  Returns a string containing the logger name.
- virtual void SetFilter (LogFilterPtr &filter)=0

  Sets the LogFilter to be used by the logger.
- virtual LogFilter const * GetFilter () const =0

  Returns the LogFilter the logger uses.
- virtual void SetFormatter (LogFormatterPtr &formatter)=0

  Sets the LogFormatter to be used by the logger.
- virtual LogFormatter const * GetFormatter () const =0

  Returns the LogFormatter the logger uses.

4.82.1 Detailed Description

Abstract class, tracks and stores information about events of interest happening in a system.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.82.2 Member Function Documentation

4.82.2.1 virtual void Common::Logger::Log (LogEntry const & *entry*) [pure virtual]

Constructor.

Parameters

entry	

Implemented in Common::FileLogger, and Common::StreamLogger.

The documentation for this class was generated from the following file:

- Common/Syslog.h

4.83 ModelParam Struct Reference

Stores the total volume of data, the total amount of data sent by workers and the total computed time.

`#include <FactoringStats_nw.h>`

Public Attributes

- int **TotalDataVolume**
- int **TotalDataSendW**
- double **TotalCompTime**

4.83.1 Detailed Description

Stores the total volume of data, the total amount of data sent by workers and the total computed time.

The documentation for this struct was generated from the following file:

- Analyzer/FactoringStats_nw.h

4.84 ModuleList Class Reference

Class that stores and handles a vector of BPatch_modules.

`#include <di.h>`

Public Member Functions

- ModuleList (BPatch_image &bpImage)

  Constructor.
- int GetSize () const

  Getter of the vector size.
- BPatch_module & **operator[]** (int i) const

4.84.1 Detailed Description

Class that stores and handles a vector of BPatch_modules.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.84.2 Constructor & Destructor Documentation

4.84.2.1 ModuleList::ModuleList (BPatch_image & *bpImage*) [inline]

Constructor.

Parameters

bpImage	Program image

4.84.3 Member Function Documentation

4.84.3.1 int ModuleList::GetSize ( ) const [inline]

Getter of the vector size.

Returns

_vector->size

The documentation for this class was generated from the following file:

- Common/di.h

4.85 Monitor Class Reference

Adds request to add or remove instrumentation in/from the tasks that it is monitoring.

`#include <Monitor.h>`

Public Member Functions

- Monitor (TaskCollection &tasks)
  Constructor.
- void AddInstr (AddInstrRequest &instrReq)
  Adds the instructions requested to the task they belong to.
- void RemoveInstr (RemoveInstrRequest &instrReq)
  Removes the instructions requested from the selected task.

4.85.1 Detailed Description

Adds request to add or remove instrumentation in/from the tasks that it is monitoring.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.85.2 Constructor & Destructor Documentation

4.85.2.1 Monitor::Monitor (TaskCollection & *tasks*) [inline]

Constructor.

Parameters

tasks	Collection of tasks susceptible to be modified.

4.85.3 Member Function Documentation

4.85.3.1 void Monitor::AddInstr (AddInstrRequest & *instrReq*)

Adds the instructions requested to the task they belong to.

Parameters

instReq	Object that represents the request for instrumentation to be added to a task.

4.85.3.2 void Monitor::RemoveInstr (RemoveInstrRequest & *instrReq*)

Removes the instructions requested from the selected task.

Parameters

instReq	Object that represents the request for instrumentation to be removed from a task.

The documentation for this class was generated from the following files:

- AC/Monitor.h
- AC/Monitor.cpp

4.86 Common::Mutex Class Reference

Guarantees non concurrent access to a resource.

`#include <sync.h>`

Public Member Functions

- Mutex ()

  Constructor.
- ~Mutex ()

  Destructor.
- **operator pthread_mutex_t** * ()

Protected Member Functions

- void Enter ()

  Denotes that someone starts using the resource.
- bool CanEnter ()

  Returns true if the resource is not being used, false otherwise.
- void Leave ()

  Denotes that someone stops using the resource.

Friends

- class **MutexLock**

4.86.1 Detailed Description

Guarantees non concurrent access to a resource.

Mutual-Exclusion Object

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.86.2 Constructor & Destructor Documentation

4.86.2.1 Common::Mutex::Mutex ( ) [inline]

Constructor.

The mutex is always initialized as a recursive entity.

4.86.3 Member Function Documentation

4.86.3.1 void Mutex::Enter ( ) [protected]

Denotes that someone starts using the resource.

Exceptions

SysException	

4.86.3.2 void Mutex::Leave ( ) [protected]

Denotes that someone stops using the resource.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/sync.h
- Common/sync.cpp

4.87 Common::MutexLock Class Reference

System to manage access to a resource with a mutex.

`#include <sync.h>`

Public Member Functions

- MutexLock (Mutex &mutex)

  Constructor.
- ~MutexLock ()

  Destructor.

4.87.1 Detailed Description

System to manage access to a resource with a mutex.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- Common/sync.h

4.88 myauto_ptr< X > Class Template Reference

Public Types

- typedef X **element_type**

Public Member Functions

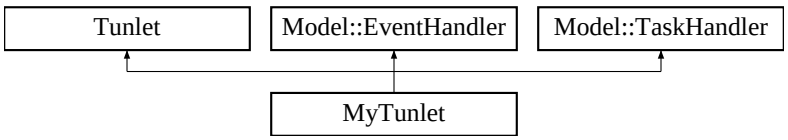
- **myauto_ptr** (X *p=0)
- **myauto_ptr** (const myauto_ptr &a)
- myauto_ptr & **operator=** (const myauto_ptr &a)
- X & **operator*** () const
- X * **operator->** () const
- X * **get** () const
- X * **release** () const

The documentation for this class was generated from the following file:

- Common/auto_ptr.h

4.89 MyTunlet Class Reference

Inheritance diagram for MyTunlet:



Public Member Functions

- void Initialize (Model::Application &app)

  Initializes the tunlet.
- void BeforeAppStart ()

  Asserts that _app != 0 and sets the task handler of the app to the current one.
- void Destroy ()

  Sets _app = 0.
- void HandleEvent (Model::EventRecord const &r)

  Handles an event record (virtual).
- void **CreateEvent** ()
- void **CreateEvent** (Model::Task &t)
- void TaskStarted (Model::Task &t)

  Called when a new task is started.
- void TaskTerminated (Model::Task &t)

  Called when a task is terminated.

4.89.1 Member Function Documentation

4.89.1.1 void MyTunlet::HandleEvent (Model::EventRecord const & *r*) [virtual]

Handles an event record (virtual).

Parameters

r	Event record to be handled

Implements Model::EventHandler.

4.89.1.2 void MyTunlet::Initialize (Model::Application & *app*) [virtual]

Initializes the tunlet.

Parameters

app	App associated to the tunlet

Implements Tunlet.

4.89.1.3 void MyTunlet::TaskStarted (Model::Task & *t*) [virtual]

Called when a new task is started.

Parameters

t	Started task object.

Implements Model::TaskHandler.

4.89.1.4 void MyTunlet::TaskTerminated (Model::Task & *t*) [virtual]

Called when a task is terminated.

Parameters

t	Terminated task object.

Implements Model::TaskHandler.

The documentation for this class was generated from the following files:

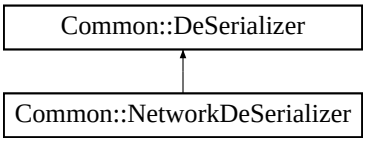
- Analyzer/MyTunlet.h
- Analyzer/MyTunlet.cpp

4.90 Common::NetworkDeSerializer Class Reference

Extracts serialized data from an istream object.

`#include <NetSer.h>`

Inheritance diagram for Common::NetworkDeSerializer:



Public Member Functions

- NetworkDeSerializer (std::istream &stream)

  Constructor.

- std::istream & GetStream ()
  Returns istream object where the data is serialized.
- long_t GetLong ()
  Reads long value from the stream.
- double_t GetDouble ()
  Reads double value from the stream.
- bool_t GetBool ()
  Reads bool value from the stream.
- short_t GetShort ()
  Reads short value from the stream.
- byte_t GetByte ()
  Reads byte value from the stream.
- char_t GetChar ()
  Reads char value from the stream.
- std::string GetString ()
  Reads string value from the stream.
- int_t GetInt ()
  Reads int value from the stream.
- void GetBuffer (char ∗buffer, int bufferSize)
  Reads data directly from the stream.

4.90.1 Detailed Description

Extracts serialized data from an istream object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

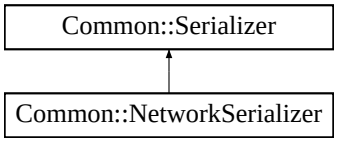
- Common/NetSer.h
- Common/NetSer.cpp

4.91 Common::NetworkSerializer Class Reference

Puts serialized data into an OutputStream object.

`#include <NetSer.h>`

Inheritance diagram for Common::NetworkSerializer:



Public Member Functions

- NetworkSerializer (OutputStream &stream)

  Constructor.
- void PutLong (long_t l)

  Puts a long into the stream.
- void PutDouble (double_t d)

  Puts a double into the stream.
- void PutBool (bool_t b)

  Puts a boolean into the stream.
- void PutShort (short_t s)

  Puts a short into the stream.
- void PutByte (byte_t b)

  Puts a byte into the stream.
- void PutChar (char_t c)

  Puts a char into the stream.
- void PutString (std::string const &str)

  Puts a string into the stream.
- void PutInt (int_t i)

  Puts an integer long into the stream.
- void PutBuffer (char const *buffer, int bufferSize)

  Puts a buffer into the stream.

4.91.1 Detailed Description

Puts serialized data into an OutputStream object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.91.2 Constructor & Destructor Documentation

4.91.2.1 Common::NetworkSerializer::NetworkSerializer (OutputStream & *stream*) [inline]

Constructor.

Parameters

stream	Stream where the serialized data will be written.

The documentation for this class was generated from the following files:

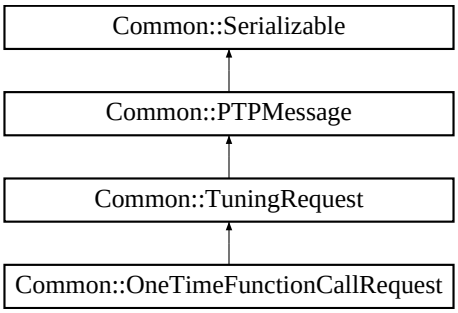
- Common/NetSer.h
- Common/NetSer.cpp

4.92 Common::OneTimeFunctionCallRequest Class Reference

Encapsulates a tuning request to invoke one time a given function in a given application process.

`#include <PTPMsg.h>`

Inheritance diagram for Common::OneTimeFunctionCallRequest:



Public Member Functions

- OneTimeFunctionCallRequest (int pid=0, std::string const &funcName=std::string(), int nAttrs=0, Attribute const *attrs=0, Breakpoint const *brkpt=0)

  Constructor.
- ~OneTimeFunctionCallRequest ()

  Destructor.
- PTPMsgType GetType () const

  Returns type of message (PTPOneTimeFuncCall).
- std::string const & GetFuncName () const

  Returns name of the function to be added.
- int GetAttrCount () const

  Returns number of attributes the function has.
- Attribute * GetAttributes () const

  Returns array of attributes.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Gets the message.

Additional Inherited Members

4.92.1 Detailed Description

Encapsulates a tuning request to invoke one time a given function in a given application process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.92.2 Constructor & Destructor Documentation

4.92.2.1 Common::OneTimeFunctionCallRequest::OneTimeFunctionCallRequest (int *pid* = 0, std::string const & *funcName* = std::string(), int *nAttrs* = 0, Attribute const * *attrs* = 0, Breakpoint const * *brkpt* = 0) [inline]

Constructor.

Parameters

pid	Id of the process where the call will be inserted, default 0.
funcName	Name of the function to call, default "".
nAttrs	Number of attributes the function has, default 0.
attrs	Attribute array, default 0.
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

The documentation for this class was generated from the following files:

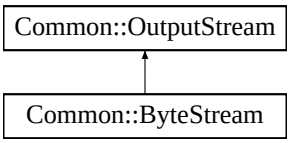
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.93 Common::OutputStream Class Reference

Abstract class, represents an output stream of bytes.

`#include <OutputStream.h>`

Inheritance diagram for Common::OutputStream:



Public Member Functions

- virtual void **Write** (char const *buf, size_t size)=0

4.93.1 Detailed Description

Abstract class, represents an output stream of bytes.

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of OutputStream must always provide at least a method that writes one byte of output.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

The documentation for this class was generated from the following file:

- Common/OutputStream.h

4.94 Common::Pipe Class Reference

Element used to join output and input from two processes.

`#include <Pipe.h>`

Public Member Functions

- Pipe ()
  Constructor.
- ~Pipe ()
  Destructor.
- bool IsReadOpen () const
  Returns whether the read end is open or not.
- bool IsWriteOpen () const
  Returns whether the write end is open or not.
- int GetRead () const
  Returns read file descriptor.
- int GetWrite () const
  Returns write file descriptor.
- void CloseRead ()
  Closes the read end.
- void CloseWrite ()
  Closes the write end.
- int Read (char *buf, int bufSize)
  Reads from the read end and stores the content on the buffer.
- int Write (char const *buf, int bufSize)
  Writes the content of the buffer on the write end.

4.94.1 Detailed Description

Element used to join output and input from two processes.

A pair of channels that implements a unidirectional pipe.

A pipe consists of a pair of channels: A writable sink channel and a readable source channel. Once some bytes are written to the sink channel these can be read from source channel in the exact order in which they were written.

Whether or not a thread writing bytes to a pipe will block until another thread reads those bytes, or some previously-written bytes, from the pipe is system-dependent and therefore unspecified. Many pipe implementations will buffer up to a certain number of bytes between the sink and source channels, but such buffering should not be assumed.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

4.94.2 Constructor & Destructor Documentation

4.94.2.1 Pipe::Pipe ( )

Constructor.

Exceptions

SysException	

4.94.3 Member Function Documentation

4.94.3.1 int Pipe::Read (char * *buf*, int *bufSize*)

Reads from the read end and stores the content on the buffer.

Exceptions

SysException	

4.94.3.2 int Pipe::Write (char const * *buf*, int *bufSize*)

Writes the content of the buffer on the write end.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/Pipe.h
- Common/Pipe.cpp

4.95 PointList Class Reference

Class that stores a vector of BPatch_points and handles it. Can also get the address and function names of a given point.

`#include <di.h>`

Public Member Functions

- PointList (DiFunction &func)

  Constructor.
- int GetSize () const

  Getter of the vector size.
- void GetCalledFuncName (BPatch_point &point, char *name, int length)

  Gets the name of the function called at the given point.

- unsigned long GetAddress (BPatch_point &point)

  Gets the address of the given point.
- BPatch_point & **operator[]** (int i) const

4.95.1 Detailed Description

Class that stores a vector of BPatch_points and handles it. Can also get the address and function names of a given point.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.95.2 Constructor & Destructor Documentation

4.95.2.1 PointList::PointList (DiFunction & *func*) [inline]

Constructor.

Parameters

func	

4.95.3 Member Function Documentation

4.95.3.1 unsigned long PointList::GetAddress (BPatch_point & *point*) [inline]

Gets the address of the given point.

Parameters

point	Given point to look get address from

Returns

Address of the given point

4.95.3.2 void PointList::GetCalledFuncName (BPatch_point & *point*, char * *name*, int *length*) [inline]

Gets the name of the function called at the given point.

Parameters

point	Point where the function is

name	Name of the function
length	Length of the name

4.95.3.3 int PointList::GetSize ( ) const [inline]

Getter of the vector size.

Returns

Vector's size

The documentation for this class was generated from the following file:

- Common/di.h

4.96 ProcedureList Class Reference

Implements and handles a vector of BPatch_functions.

`#include <di.h>`

Public Member Functions

- ProcedureList (BPatch_image &bpImage)
  Constructor.
- int GetSize () const
  Getter of the vector's size.
- BPatch_function & **operator[]** (int i) const

4.96.1 Detailed Description

Implements and handles a vector of BPatch_functions.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.96.2 Constructor & Destructor Documentation

4.96.2.1 ProcedureList::ProcedureList (BPatch_image & *bpImage*) [inline]

Constructor.

Parameters

bpImage	Image of the program

4.96.3 Member Function Documentation

4.96.3.1 int ProcedureList::GetSize ( ) const [inline]

Getter of the vector's size.

Returns

Vector's size

The documentation for this class was generated from the following file:

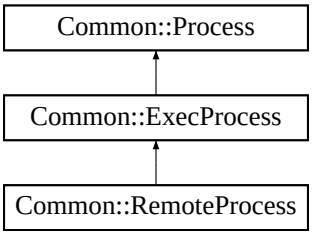
- Common/di.h

4.97 Common::Process Class Reference

Abstract class, creates a new process to perform different operations on the overrided method Run().

`#include <Process.h>`

Inheritance diagram for Common::Process:



Public Member Functions

- virtual void Start ()

  Executes the process.
- int GetPid () const

  Returns process id.

Protected Member Functions

- Process ()

  Constructor.
- virtual int **Run** ()=0

Protected Attributes

- int **_pid**

4.97.1 Detailed Description

Abstract class, creates a new process to perform different operations on the overrided method Run().

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

The documentation for this class was generated from the following files:

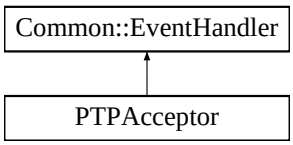
- Common/Process.h
- Common/Process.cpp

4.98 PTPAcceptor Class Reference

Manages socket connection and handles data input through them.

`#include <PTPAcceptor.h>`

Inheritance diagram for PTPAcceptor:



Public Member Functions

- PTPAcceptor (Reactor &reactor, TaskManager &tm, int port)

  Constructor.
- ~PTPAcceptor ()

  Destructor.
- void HandleInput ()

  Gets the socket for the client and binds it with the task manager.
- int GetHandle ()

  Getter of a handler for the variable _socket.

4.98.1 Detailed Description

Manages socket connection and handles data input through them.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.98.2 Constructor & Destructor Documentation

4.98.2.1 PTPAcceptor::PTPAcceptor (Reactor & *reactor,* TaskManager & *tm,* int *port*)

Constructor.

Parameters

reactor	Object of class reactor that manages event handlers.
tm	Task manager.

4.98.3 Member Function Documentation

4.98.3.1 int PTPAcceptor::GetHandle ( ) [inline], [virtual]

Getter of a handler for the variable _socket.

Returns

Handle of the server socket.

Implements Common::EventHandler.

The documentation for this class was generated from the following files:

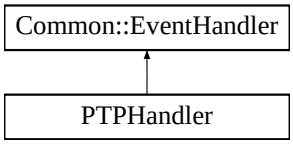
- AC/PTPAcceptor.h
- AC/PTPAcceptor.cpp

4.99 PTPHandler Class Reference

Manages the requests from the PTPAcceptor.

`#include <PTPHandler.h>`

Inheritance diagram for PTPHandler:



Public Member Functions

- PTPHandler (SocketPtr &socket, TaskManager &tm)
  Constructor.
- void HandleInput ()
  Reads message from socket and handles the different kinds of requests that are received.
- int GetHandle ()
  Getter of a handler for the variable _socket.

4.99.1 Detailed Description

Manages the requests from the PTPAcceptor.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.99.2 Constructor & Destructor Documentation

4.99.2.1 PTPHandler::PTPHandler (SocketPtr & *socket,* TaskManager & *tm*) [inline]

Constructor.

Parameters

socket	Pointer to the socket used to get the input (request).
tm	Task manager that handles the task to which the request affects.

4.99.3 Member Function Documentation

4.99.3.1 int PTPHandler::GetHandle ( ) [inline],[virtual]

Getter of a handler for the variable _socket.

Returns

Handle of the socket.

Implements Common::EventHandler.

The documentation for this class was generated from the following files:

- AC/PTPHandler.h
- AC/PTPHandler.cpp

4.100 Common::PTPMessage Class Reference

Performance tuning protocol, represents a message interchanged between analyzer and tuner/tracer.

`#include <PTPMsg.h>`

Inheritance diagram for Common::PTPMessage:

Public Member Functions

- PTPMessage ()

  Constructor.
- virtual PTPMsgType GetType () const

  To be implemented by subclasses.
- int GetDataSize () const

  Returns size of the data once serialized.
- virtual ~PTPMessage ()

  Destructor.

4.100.1 Detailed Description

Performance tuning protocol, represents a message interchanged between analyzer and tuner/tracer.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

The documentation for this class was generated from the following files:

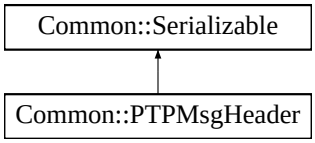
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.101 Common::PTPMsgHeader Class Reference

Represents header of a PTPMessage object.

`#include <PTPMsgHeader.h>`

Inheritance diagram for Common::PTPMsgHeader:



Public Member Functions

- PTPMsgHeader ()

  Constructor.
- void Serialize (Serializer &out) const

  Sends the message header.
- void DeSerialize (DeSerializer &in)

Receives the message header.

- int GetMagic () const

  Returns magic attribute.

- int GetVersion () const

  Returns version attribute.

- PTPMsgType GetType () const

  Returns the type of the message.

- int GetDataSize () const

  Returns data size.

- int GetHeaderSize () const

  Returns header size.

- void SetMagic (int magic)

  Sets the magic attribute.

- void SetVersion (int version)

  Sets the version attribute.

- void SetMsgType (PTPMsgType type)

  Sets the type of the message.

- void SetDataSize (int size)

  Sets data size.

- void SetHeaderSize ()

  Updates header size.

4.101.1 Detailed Description

Represents header of a PTPMessage object.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- Common/PTPMsgHeader.h
- Common/PTPMsgHeader.cpp

4.102 Common::PTPProtocol Class Reference

Communicates analyzer and tuner.

`#include <PTPProtocol.h>`

Static Public Member Functions

- static void WriteMessage (PTPMessage &msg, OutputStream &stream)

  Sends a message through a stream to the tuner.
- static PTPMessage * ReadMessage (std::istream &stream)

  Receives a message through a stream from the analyzer.
- static void WriteMessageEx (PTPMessage &msg, Socket &sock)

  Sends a message through a socket to the tuner.
- static PTPMessage * ReadMessageEx (Socket &sock)

  Receives a message through a socket from the analyzer.

4.102.1 Detailed Description

Communicates analyzer and tuner.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- Common/PTPProtocol.h
- Common/PTPProtocol.cpp

4.103 Common::Queue< T > Class Template Reference

Data structure that stores objects of any class.

`#include <Queue.h>`

Public Member Functions

- Queue (int maxSize)

  Constructor.
- ~Queue ()

  Destructor.
- bool IsEmpty () const

  Returns true if the queue is empty, false otherwise.
- bool IsFull () const

  Returns true if the queue is full, false otherwise.
- int GetMaxSize () const

  Returns the maximum quantity of objects the queue can store.
- int GetCount () const

  Returns current size of the queue.

- bool Get (T &item)

  Returns first object of the queue and removes it.
- void GetB (T &item)

  Returns first object of the queue and removes it.
- void Put (T &item)

  Puts an object at the end of the queue.

4.103.1 Detailed Description

template<class T>class Common::Queue< T >

Data structure that stores objects of any class.

This data structure manages the objects using a FIFO priority.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.103.2 Member Function Documentation

4.103.2.1 template<class T> bool Queue::Get (T & *item*)

Returns first object of the queue and removes it.

Returns false if the queue is empty.

4.103.2.2 template<class T> void Queue::GetB (T & *item*)

Returns first object of the queue and removes it.

If the queue is empty, waits until it has any object.

4.103.2.3 template<class T> void Queue::Put (T & *item*)

Puts an object at the end of the queue.

If the queue is full, it waits until there's space.

The documentation for this class was generated from the following file:

- Common/Queue.h

4.104 Common::Reactor Class Reference

Registers, removes and dispatches EventHandler objects.

`#include <Reactor.h>`

Public Member Functions

- Reactor ()

  Constructor.
- void Register (EventHandler &handler)

  Registers a new EventHandler.
- void UnRegister (EventHandler &handler)

  Removes given EventHandler.
- void HandleEvents (TimeValue *timeout=0)

  Runs event loop.
- EventHandler & GetHandler (int handle)

  Returns selected handler.

4.104.1 Detailed Description

Registers, removes and dispatches EventHandler objects.

Uses reactor design pattern.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

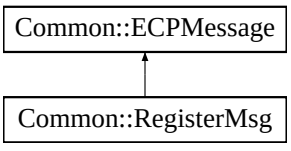
- Common/Reactor.h
- Common/Reactor.cpp

4.105 Common::RegisterMsg Class Reference

Represents message that is sent when DMLib is registered with analyzer to send event messages.

`#include <ECPMsg.h>`

Inheritance diagram for Common::RegisterMsg:



Public Member Functions

- RegisterMsg (int pid=0, int mpiRank=0, std::string host=std::string(), std::string taskName=std::string(), int ACport=0)

  Constructor.
- ECPMsgType GetType () const

  Returns the type of event.
- int GetPid () const

  Returns Id of the process where the library will be loaded.
- int GetMpiRank () const

  Returns mpi rank.
- string const & GetHost () const

  Returns host name where the process is located.
- string const & GetTaskName () const

  Returns task name.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.105.1 Detailed Description

Represents message that is sent when DMLib is registered with analyzer to send event messages.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.105.2 Constructor & Destructor Documentation

4.105.2.1 Common::RegisterMsg::RegisterMsg (int *pid* = `0`, int *mpiRank* = `0`, std::string *host* = `std::string()`, std::string *taskName* = `std::string()`, int *ACport* = `0`) [inline]

Constructor.

Parameters

pid	Id of the process where the library will be registered.
mpiRank	Mpi rank.
host	Host where the process is located.

taskName	Name of the task.

The documentation for this class was generated from the following files:

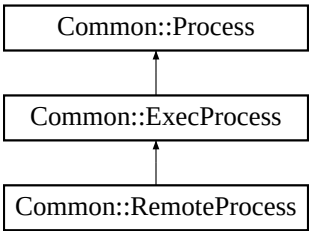
- Common/ECPMsg.h
- Common/ECPMsg.cpp

4.106 Common::RemoteProcess Class Reference

Remotely executes a command in another machine.

`#include <Process.h>`

Inheritance diagram for Common::RemoteProcess:



Public Member Functions

- RemoteProcess (std::string hostName, std::string command, std::string rshPath=std::string("/usr/bin/rsh"))

  Constructor.
- RemoteProcess (std::string hostName, std::string userName, std::string command, std::string rshPath=std::string("/usr/bin/rsh"))

  Constructor.
- void SetOutputToNull ()

  Enables output to null property.

Protected Member Functions

- int Run ()

  Executes de process.
- std::string **GetHostName** () const

Additional Inherited Members

4.106.1 Detailed Description

Remotely executes a command in another machine.

Uses the rsh program to perform the execution of the process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2001

4.106.2 Constructor & Destructor Documentation

4.106.2.1 Common::RemoteProcess::RemoteProcess (std::string *hostName,* std::string *command,* std::string *rshPath =* `std::string("/usr/bin/rsh")`) [inline]

Constructor.

Parameters

hostName	Host where the command will be executed remotely.
command	Command to execute.
rshPath	Local path of the rsh program, default = "/usr/bin/rsh".

4.106.2.2 Common::RemoteProcess::RemoteProcess (std::string *hostName,* std::string *userName,* std::string *command,* std::string *rshPath =* `std::string("/usr/bin/rsh")`) [inline]

Constructor.

Parameters

hostName	Host where the command will be executed remotely.
command	Command to execute.
userName	Name of the user on the host to execute the command.
rshPath	Local path of the rsh program, default = "/usr/bin/rsh".

The documentation for this class was generated from the following files:

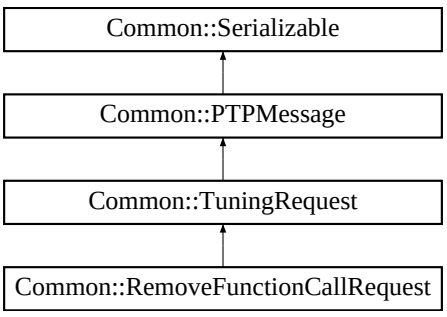
- Common/Process.h
- Common/Process.cpp

4.107 Common::RemoveFunctionCallRequest Class Reference

Encapsulates a tuning request to remove all calls to a given function from the given caller function.

`#include <PTPMsg.h>`

Inheritance diagram for Common::RemoveFunctionCallRequest:



Public Member Functions

- RemoveFunctionCallRequest (int pid=0, std::string const &funcName=std::string(), std::string const &callerFunc=string(), Breakpoint const *brkpt=0)

  Constructor.
- PTPMsgType GetType () const

  Returns type of message (PTPRemoveFuncCall).
- std::string const & GetFuncName () const

  Returns name of the function to be added.
- std::string const & GetCallerFunc () const

  Returns the function caller name.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.107.1 Detailed Description

Encapsulates a tuning request to remove all calls to a given function from the given caller function.

Version

1.0

Since

1.0

Author

Ania Sikora, 2003

4.107.2 Constructor & Destructor Documentation

4.107.2.1 Common::RemoveFunctionCallRequest::RemoveFunctionCallRequest (int *pid* = `0`, std::string const & *funcName* = `std::string()`, std::string const & *callerFunc* = `string()`, Breakpoint const * *brkpt* = `0`) `[inline]`

Constructor.

Parameters

pid	Id of the process where the call will be removed, default 0.
funcName	Name of the function to remove, default "".
callerFunc	Name of the caller function, default "".
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

The documentation for this class was generated from the following files:

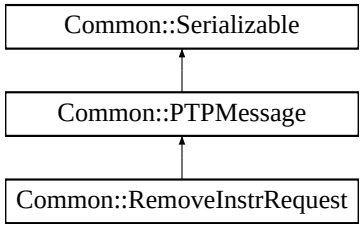
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.108 Common::RemoveInstrRequest Class Reference

Represents message sent when analyzer requests to remove instrumentation.

`#include <PTPMsg.h>`

Inheritance diagram for Common::RemoveInstrRequest:



Public Member Functions

- RemoveInstrRequest (int pid=0, int eventId=0, InstrPlace place=ipFuncEntry)
  Constructor.
- PTPMsgType GetType () const
  Returns type of message (PTPRemoveInstr).
- int GetPid () const
  Returns the process id.
- int GetEventId () const
  Returns the event id.
- InstrPlace GetInstrPlace () const
  Returns the place where the instruction should be removed.
- void Serialize (Serializer &out) const
  Sends the message.
- void DeSerialize (DeSerializer &in)
  Receives the message.

4.108.1 Detailed Description

Represents message sent when analyzer requests to remove instrumentation.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.108.2 Constructor & Destructor Documentation

4.108.2.1 Common::RemoveInstrRequest::RemoveInstrRequest (int *pid* = `0`, int *eventId* = `0`, InstrPlace *place* = `ipFuncEntry`) [inline]

Constructor.

Parameters

pid	Id of the process where the instrumentation will be removed, default 0.
eventId	Event id, default 0.
place	Place where the instrumentation will be removed, default ipFuncEntry.

The documentation for this class was generated from the following files:

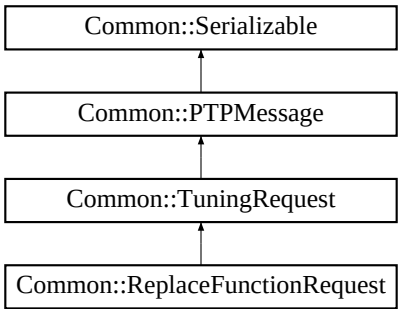
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.109 Common::ReplaceFunctionRequest Class Reference

Encapsulates a tuning request to replace all calls to a function inside a process with calls to another function.

`#include <PTPMsg.h>`

Inheritance diagram for Common::ReplaceFunctionRequest:



Public Member Functions

- ReplaceFunctionRequest (int pid=0, std::string const &oldFunc=std::string(), std::string const &newFunc=std::string(), Breakpoint *brkpt=0)

  Constructor.
- PTPMsgType GetType () const

  Returns type of message (PTPReplaceFunction).
- std::string const & GetOldFunction () const

  Returns a string containing the name of the function to replace.
- std::string const & GetNewFunction () const

  Returns a string containing the name of the function added.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.109.1 Detailed Description

Encapsulates a tuning request to replace all calls to a function inside a process with calls to another function.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.109.2 Constructor & Destructor Documentation

4.109.2.1 Common::ReplaceFunctionRequest::ReplaceFunctionRequest (int *pid* = `0`, std::string const & *oldFunc* = `std::string()`, std::string const & *newFunc* = `std::string()`, Breakpoint * *brkpt* = `0`) [inline]

Constructor.

Parameters

pid	Id of the process where the function will be replaced, default 0.
oldFunc	Name of the function to replace, default "".
newFunc	Name of the function to add, default "".
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.110 Common::Semaphore Class Reference

Synchronizes access to a resource.

`#include <sync.h>`

Public Member Functions

- Semaphore (int initialValue=0, bool crossProcess=false)

  Constructor.
- ~Semaphore ()

  Destructor.
- void Wait ()

  Stops the current thread until the access to the resource is open.
- bool TryWait ()

  Returns true if the access to the resource is open, false otherwise.
- void Post ()

  Gives a signal to the semaphore indicating that a client has stopped using the resource.

Protected Member Functions

- **Semaphore** (sem_t s)

Protected Attributes

- sem_t **_semaphore**

4.110.1 Detailed Description

Synchronizes access to a resource.

A semaphore is generally used as a synchronization object between multiple threads or to protect a limited and finite resource such as a memory or thread pool. The semaphore has a counter which only permits access by one or more threads when the value of the semaphore is non-zero. Each access reduces the current value of the semaphore by 1. One or more threads can wait on a semaphore until it is no longer 0, and hence the semaphore can be used as a simple thread synchronization object to enable one thread to pause others until the thread is ready or has provided data for them.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.110.2 Constructor & Destructor Documentation

4.110.2.1 Common::Semaphore::Semaphore (int *initialValue* = `0`, bool *crossProcess* = `false`) [inline]

Constructor.

Parameters

initialValue	Initial value of the semaphore, default 0.
crossProcess	Flag indicating whether or not the semaphore should be shared with forked processes, default false.

4.110.3 Member Function Documentation

4.110.3.1 void Semaphore::Post ( )

Gives a signal to the semaphore indicating that a client has stopped using the resource.

Posting to a semaphore increments its current value and releases the first thread waiting for the semaphore if it is currently at 0.

4.110.3.2 bool Semaphore::TryWait ( )

Returns true if the access to the resource is open, false otherwise.

TryWait is a non-blocking variant of Wait. If the semaphore counter is greater than 0, then the thread is accepted and the semaphore counter is decreased. If the semaphore counter is 0 TryWait returns immediately with false.

4.110.3.3 void Semaphore::Wait ( )

Stops the current thread until the access to the resource is open.

Wait is used to keep a thread held until the semaphore counter is greater than 0. If the current thread is held, then another thread must increment the semaphore. Once the thread is accepted, the semaphore is automatically decremented, and the thread continues execution.

The documentation for this class was generated from the following files:

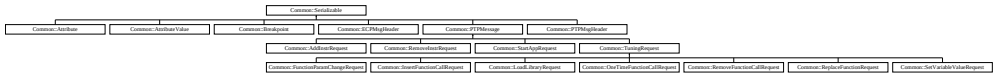
- Common/sync.h
- Common/sync.cpp

4.111 Common::Serializable Class Reference

Abstract class, makes an object able to be passed through a stream using Serializer and DeSerializer objects.

`#include <Serial.h>`

Inheritance diagram for Common::Serializable:



Public Member Functions

- virtual ~Serializable ()

  Destructor.
- virtual void Serialize (Serializer &out) const =0

  Sends the message header.
- virtual void DeSerialize (DeSerializer &in)=0

  Receives the message header.

4.111.1 Detailed Description

Abstract class, makes an object able to be passed through a stream using Serializer and DeSerializer objects.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

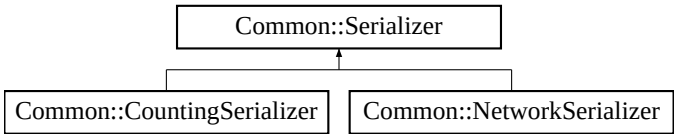
- Common/Serial.h

4.112 Common::Serializer Class Reference

Abstract class, prepares objects to be passed on a stream.

`#include <Serial.h>`

Inheritance diagram for Common::Serializer:



Public Member Functions

- virtual void PutByte (byte_t b)=0
  Adds the size of a serialized byte.
- virtual void PutChar (char_t c)=0
  Adds the size of a serialized char.
- virtual void PutBool (bool_t b)=0
  Adds the size of a serialized bool.
- virtual void PutShort (short_t s)=0
  Adds the size of a serialized short.
- virtual void PutInt (int_t i)=0
  Adds the size of a serialized int.
- virtual void PutLong (long_t l)=0
  Adds the size of a serialized long.
- virtual void PutDouble (double_t d)=0
  Adds the size of a serialized double.
- virtual void PutString (std::string const &str)=0
  Adds the size of a serialized string.
- virtual void PutBuffer (char const *buffer, int bufferSize)=0
  Adds the size of a serialized buffer.

4.112.1 Detailed Description

Abstract class, prepares objects to be passed on a stream.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- Common/Serial.h

4.113 Common::ServerSocket Class Reference

Holds a SocketBase object and represents a TCP/IP server socket.

`#include <Socket.h>`

Public Member Functions

- ServerSocket (int port, int backLog=5)

  Constructor.
- void Listen ()

  Sets the socket to a listening state.
- SocketPtr Accept ()

  Accepts a connection and creates a socket.
- SocketPtr Accept (int timeoutMs)

  Accepts a connection and creates a socket, waits the given timeout.
- Address const & GetAddress () const

  Returns local server address.
- int GetLocalPort () const

  Returns the port the socket is listening to.
- int GetHandle () const

  Returns socket handle.

4.113.1 Detailed Description

Holds a SocketBase object and represents a TCP/IP server socket.

All the functions on this class are present on the SocketBase class and have the same functionality.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.113.2 Constructor & Destructor Documentation

4.113.2.1 ServerSocket::ServerSocket (int *port*, int *backLog* = 5)

Constructor.

Parameters

backLog	Maximum length of the queue for pending connections.

Exceptions

SysException.	

The documentation for this class was generated from the following files:

- Common/Socket.h
- Common/Socket.cpp

4.114 Service Class Reference

Provides methods to work with EventCollectorHandlers lists. Holds a list of EventCollHandler and a reference to the reactor. Provides methods to add and remove handlers from the list.

`#include <Service.h>`

Public Member Functions

- Service (Reactor &reactor)

  Constructor.
- ~Service ()

  Destructor, deletes the handlers and the references to them.
- void Add (ECPHandler *handler)

  Adds a handler to the list and sets its service to this.
- void Remove (ECPHandler *handler)

  Unregisters the handler from the reactor and removes it from the list.

4.114.1 Detailed Description

Provides methods to work with EventCollectorHandlers lists. Holds a list of EventCollHandler and a reference to the reactor. Provides methods to add and remove handlers from the list.

4.114.2 Constructor & Destructor Documentation

4.114.2.1 Service::Service (Reactor & *reactor*) [inline]

Constructor.

Parameters

reactor	Reactor of the application

4.114.3 Member Function Documentation

4.114.3.1 void Service::Add (ECPHandler * *handler*)

Adds a handler to the list and sets its service to this.

Parameters

handler	ECP Handler.

4.114.3.2 void Service::Remove (ECPHandler * *handler*)

Unregisters the handler from the reactor and removes it from the list.

Parameters

handler	ECP Handler

Exceptions

Exception	when the handler does not exist in the list.

The documentation for this class was generated from the following files:

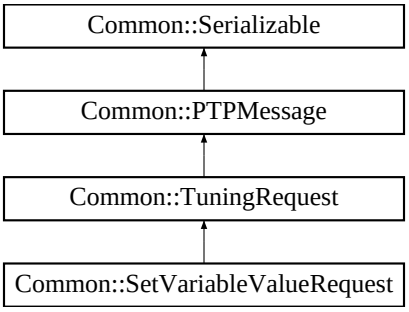
- Analyzer/Service.h
- Analyzer/Service.cpp

4.115 Common::SetVariableValueRequest Class Reference

Encapsulates a tuning request to modify a value of a specified variable in a given application process.

`#include <PTPMsg.h>`

Inheritance diagram for Common::SetVariableValueRequest:



Public Member Functions

- SetVariableValueRequest (int pid=0, std::string const &varName=std::string(), AttributeValue const &varValue=AttributeValue(), Breakpoint *brkpt=0)

  Constructor.
- void * GetValueBuffer ()

  Returns the value to set in a buffer format.
- int GetValueSize () const

  Returns size of the variable new value.
- std::string GetValueString () const

  Returns a string containing the value of the variable.
- PTPMsgType GetType () const

  Returns type of message (PTPSetVariableValue).
- std::string const & GetVariableName () const

  Returns a string containing the variable name.
- void Serialize (Serializer &out) const

Sends the message.

- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.115.1 Detailed Description

Encapsulates a tuning request to modify a value of a specified variable in a given application process.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.115.2 Constructor & Destructor Documentation

4.115.2.1 Common::SetVariableValueRequest::SetVariableValueRequest (int *pid* = `0`, std::string const & *varName* = `std::string()`, AttributeValue const & *varValue* = AttributeValue `()`, Breakpoint * *brkpt* = `0`) `[inline]`

Constructor.

Parameters

pid	Id of the process where the variable will be modified, default 0.
varName	Name of the variable, default "".
varValue	New value to set, default empty AttributeValue object.
brkpt	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

4.115.3 Member Function Documentation

4.115.3.1 void* Common::SetVariableValueRequest::GetValueBuffer ( ) `[inline]`

Returns the value to set in a buffer format.

If the type is known, it can be used using a cast, for example:

```
int foo = (int) VarRequest.GetValueBuffer();
```

The documentation for this class was generated from the following files:

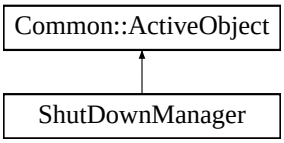
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.116 ShutDownManager Class Reference

Handles the shut down of MATE (Analyzer and AC's) The data structure consists basically in a reference to the application model (to know the hosts where the AC's are running in real time) and a boolean to determine if MATE is finished (to let the main process know, and make it stop). Provides a method to set the application model from outside (when it is ready, the main process of the Analyzer will set it). On the other hand, this class inherits from ActiveObject, so its objects are execution threads, this is done to wait for the user to stop MATE without stopping its own execution.

`#include <ShutDownManager.h>`

Inheritance diagram for ShutDownManager:



Public Member Functions

- ShutDownManager ()

  Constructor. Sets the finished member to false and starts the thread.
- virtual ~ShutDownManager ()

  Destructor.
- void Run ()

  Function that is executed by the thread. Waits for the user to stop MATE, when receives the commandment sends a stop signal to AC's and sets the variable finished to true in order to stop the Analyzer itself.
- void InitThread ()

  Not implemented (Here for compatibility reasons).
- void FlushThread ()

  Not implemented (Here for compatibility reasons).
- bool isFinished ()

  Getter of finished boolean.
- void setApp (Model::Application &app)

  Application model reference setter.

Additional Inherited Members

4.116.1 Detailed Description

Handles the shut down of MATE (Analyzer and AC's) The data structure consists basically in a reference to the application model (to know the hosts where the AC's are running in real time) and a boolean to determine if MATE is finished (to let the main process know, and make it stop). Provides a method to set the application model from outside (when it is ready, the main process of the Analyzer will set it). On the other hand, this class inherits from ActiveObject, so its objects are execution threads, this is done to wait for the user to stop MATE without stopping its own execution.

4.116.2 Member Function Documentation

4.116.2.1 bool ShutDownManager::isFinished ( ) [inline]

Getter of finished boolean.

Returns

True if the user stopped MATE, false otherwise.

4.116.2.2 void ShutDownManager::setApp (Model::Application & *app*) [inline]

Application model reference setter.

Parameters

app	Reference to the application model

The documentation for this class was generated from the following files:

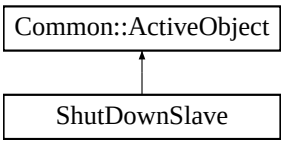
- Analyzer/ShutDownManager.h
- Analyzer/ShutDownManager.cpp

4.117 ShutDownSlave Class Reference

Receives terminating message from Analyzer.

`#include <ShutDownSlave.h>`

Inheritance diagram for ShutDownSlave:



Public Member Functions

- ShutDownSlave (string analyzerHost, int analyzerPort, Controller ctrl)

  Constructor.
- virtual ~ShutDownSlave ()

  Destructor.
- void Run ()

  Contains the main function to be run by the thread.

Protected Member Functions

- void InitThread ()

  Not implemented.
- void FlushThread ()

  Not implemented.

Additional Inherited Members

4.117.1 Detailed Description

Receives terminating message from Analyzer.

Runs a thread that waits blocked for a message from the analyzer. When it receives this message the main loop in the controller is stopped, and subsequently the AC is terminated.

Note: the tasks should be deleted at this point.

Version

1.1

Since

1.1

4.117.2 Constructor & Destructor Documentation

4.117.2.1 ShutDownSlave::ShutDownSlave (string *analyzerHost,* int *analyzerPort,* Controller *ctrl*)

Constructor.

Parameters

analyzerHost	Host in which the Analyzer is running.
analyzerPort	Port through which the connection will happen.
ctrl	Controller object.

The documentation for this class was generated from the following files:

- AC/ShutDownSlave.h
- AC/ShutDownSlave.cpp

4.118 SnippetHandler Class Reference

Contains he necessary fields to manage snippets.

`#include <InstrSet.h>`

Public Member Functions

- SnippetHandler (int eventId, std::string const &funcName, InstrPlace place, BPatchSnippetHandle ∗handle)

  Constructor.
- int GetEventId () const

  Getter for the variable _eventId.
- std::string const & GetFuncName () const

  Getter for the variable _funcName.
- InstrPlace GetInstrPlace () const

  Getter for the variable _place.
- BPatchSnippetHandle ∗ GetHandle () const

  Getter for the variable _handle.

4.118.1 Detailed Description

Contains he necessary fields to manage snippets.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.118.2 Constructor & Destructor Documentation

4.118.2.1 SnippetHandler::SnippetHandler (int *eventId,* std::string const & *funcName,* InstrPlace *place,* BPatchSnippetHandle * *handle*) [inline]

Constructor.

Parameters

eventId	Unique identifier for the event.
funcName	Name of the function in which the snippet will be inserted.
place	Position in which the instrumentation will be added.
handle	Handle for the Dyninst snippet.

4.118.3 Member Function Documentation

4.118.3.1 int SnippetHandler::GetEventId ( ) const [inline]

Getter for the variable _eventId.

Returns

Id of the event.

4.118.3.2 std::string const& SnippetHandler::GetFuncName ( ) const [inline]

Getter for the variable _funcName.

Returns

Name of the function in which the snippet will be inserted.

4.118.3.3 BPatchSnippetHandle* SnippetHandler::GetHandle ( ) const [inline]

Getter for the variable _handle.

Returns

Handle of the snippet to be inserted.

4.118.3.4 InstrPlace SnippetHandler::GetInstrPlace ( ) const [inline]

Getter for the variable _place.

Returns

Position of the function in which the snippet will be inserted.

The documentation for this class was generated from the following file:

- AC/InstrSet.h

4.119 SnippetMaker Class Reference

Prepares the snippets to be inserted into the processes.

`#include <SnippetMaker.h>`

Public Member Functions

- SnippetMaker (DiProcess &process, DiImage &image)
  Constructor.
- BPatchSnippetHandle * MakeEventSnippet (int eventId, std::string const &funcName, InstrPlace instrPlace, int nAttrs, Attribute *attrs, int nPapi, std::string *PapiMetrics)
  Creates and inserts a snippet into the running process.

4.119.1 Detailed Description

Prepares the snippets to be inserted into the processes.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.119.2 Constructor & Destructor Documentation

4.119.2.1 SnippetMaker::SnippetMaker (DiProcess & *process,* DiImage & *image*) [inline]

Constructor.

Parameters

process	Dyninst process to be modified.
image	Dyninst image of the process to be modified.

4.119.3 Member Function Documentation

4.119.3.1 BPatchSnippetHandle * SnippetMaker::MakeEventSnippet (int *eventId*, std::string const & *funcName*, InstrPlace *instrPlace*, int *nAttrs*, Attribute * *attrs*, int *nPapi*, std::string * *PapiMetrics*)

Creates and inserts a snippet into the running process.

Parameters

eventId	Identifier of the event.
funcName	Name of the function to be modified.
instrPlace	Place in the function where the snippet will be inserted.
nAttrs	Number of attributes.
attrs	Array of attributes.

Returns

Handle for the prepared snippet.

The documentation for this class was generated from the following files:

- AC/SnippetMaker.h
- AC/SnippetMaker.cpp

4.120 Common::Socket Class Reference

Holds a SocketBase object and represents a client socket.

`#include <Socket.h>`

Public Member Functions

- Socket (Address &address)

  Constructor.
- Socket (std::string const &host, int port)

  Constructor.
- Address const & GetRemoteAddress () const

  Returns the address to which the socket is connected.
- Address GetLocalAddress () const

  Returns the local address the socket is listening to.
- int GetLocalPort () const

  Returns the local port the socket is listening.
- void Send (char const *buf, int bufSize, int flags=0)

  Sends the data through the socket converted to network byte order.
- void Send (std::string const &str, int flags=0)

  Sends the data through the socket converted to network byte order.
- void Send (ByteStream &stream, int flags=0)

  Sends the data through the socket converted to network byte order.
- int Receive (char *buf, int bufSize, int flags=0)

  Gets data from the socket.

- int ReceiveN (char *buf, int bufSize, int flags=0)

  Gets data from the socket, performs multiple Receive() calls until the buffer is full.
- operator int ()

  Cast to int returns the socket handle.
- int GetHandle () const

  Returns socket handle.
- void SetTCPNoDelay (bool value)

  Sets the TCPNoDelay property.
- void SetReuseAddress (bool value)

  Sets the ReuseAddress property.
- void SetKeepAlive (bool value)

  Sets the KeepAlive property.
- void SetReceiveTimeout (int timeoutMs)

  Sets the ReceiveTimeout property.
- void SetSendTimeout (int timeoutMs)

  Sets the SendTimeout property.
- int GetReceiveTimeout ()

  Gets the ReceiveTimeout property.
- int GetSendTimeout ()

  Gets the SendTimeout property.

Protected Member Functions

- Socket (int hSocket, Address &addr)

  Constructor.

Friends

- class **SocketBase**

4.120.1 Detailed Description

Holds a SocketBase object and represents a client socket.

All the functions in this class are present on the SocketBase class and have the same functionality.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.120.2 Constructor & Destructor Documentation

4.120.2.1 Socket::Socket (Address & *address*) [inline]

Constructor.

Exceptions

SysException	

4.120.2.2 Socket::Socket (std::string const & *host,* int *port*) [inline]

Constructor.

Exceptions

SysException	

4.120.3 Member Function Documentation

4.120.3.1 int Socket::GetReceiveTimeout ( ) [inline]

Gets the ReceiveTimeout property.

Exceptions

SysException	

4.120.3.2 int Socket::GetSendTimeout ( ) [inline]

Gets the SendTimeout property.

Exceptions

SysException	

4.120.3.3 int Socket::Receive (char * *buf,* int *bufSize,* int *flags* = 0) [inline]

Gets data from the socket.

If no incoming data is available at the socket, the call blocks and waits for data to arrive.

Returns

Number of bytes received.

Exceptions

SysException	

4.120.3.4 int Socket::ReceiveN (char * *buf,* int *bufSize,* int *flags* = 0) [inline]

Gets data from the socket, performs multiple Receive() calls until the buffer is full.

Returns

Number of bytes received.

Exceptions

SysException	

4.120.3.5 void Socket::Send (char const * *buf*, int *bufSize*, int *flags* = 0) [inline]

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.120.3.6 void Socket::Send (std::string const & *str*, int *flags* = 0) [inline]

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.120.3.7 void Socket::Send (ByteStream & *stream*, int *flags* = 0)

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.120.3.8 void Socket::SetKeepAlive (bool *value*) [inline]

Sets the KeepAlive property.

Exceptions

SysException	

4.120.3.9 void Socket::SetReceiveTimeout (int *timeoutMs*) [inline]

Sets the ReceiveTimeout property.

Exceptions

SysException	

4.120.3.10 void Socket::SetReuseAddress (bool *value*) [inline]

Sets the ReuseAddress property.

Exceptions

SysException	

4.120.3.11 void Socket::SetSendTimeout (int *timeoutMs*) [inline]

Sets the SendTimeout property.

Exceptions

SysException	

4.120.3.12 void Socket::SetTCPNoDelay (bool *value*) [inline]

Sets the TCPNoDelay property.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/Socket.h
- Common/Socket.cpp

4.121 Common::SocketBase Class Reference

Represents an endpoint for communication between two machines.

`#include <Socket.h>`

Public Member Functions

- SocketBase (int family=AF_INET, int type=SOCK_STREAM, int protocol=0)
  Constructor.
- SocketBase (int hSocket, Address &addr)
  Constructor.
- virtual ~SocketBase ()
  Destructor.
- SocketPtr Accept ()
  Accepts a connection and creates a socket.
- void Bind (int port)
  Associates the socket with a local endpoint.
- void Listen (int backLog)
  Sets the socket to a listening state.
- void Connect (Address &address)
  Connects a socket on the given address.
- void Connect (std::string const &host, int port)
  Connects a socket on the given host and port.
- void Send (char const *buf, int bufSize, int flags=0)
  Sends the data through the socket converted to network byte order.
- void Send (std::string const &str, int flags=0)
  Sends the data through the socket converted to network byte order.
- void Send (ByteStream &stream, int flags=0)
  Sends the data through the socket converted to network byte order.
- int Receive (char *buf, int bufSize, int flags=0)
  Gets data from the socket.
- int ReceiveN (char *buf, int bufSize, int flags=0)
  Gets data from the socket, performs multiple Receive() calls until the buffer is full.
- operator int ()

Cast to int returns the socket handle.

- int GetHandle () const

  Returns socket handle.
- void SetTCPNoDelay (bool value)

  Sets the TCPNoDelay property.
- void SetKeepAlive (bool value)

  Sets the KeepAlive property.
- void SetReuseAddress (bool value)

  Sets the ReuseAddress property.
- void SetReceiveTimeout (int timeoutMs)

  Sets the ReceiveTimeout property.
- void SetSendTimeout (int timeoutMs)

  Sets the SendTimeout property.
- int GetReceiveTimeout ()

  Gets the ReceiveTimeout property.
- int GetSendTimeout ()

  Gets the SendTimeout property.
- Address const & GetAddress () const

  Returns the address where the socket is connected.
- int GetLocalPort () const

  Returns the port the socket is listening.

Protected Member Functions

- void SetOption (int level, int option, char const *value, int valueSize)

  Sets a socket option.
- int GetOption (int level, int option, char *value, int &valueSize)

  Gets the value of a socket option.
- int DoSend (char const *buf, int bufSize, int flags=0)

  Sends the data through the socket.

Protected Attributes

- int **_hSocket**
- Address **_addr**
- int **_localPort**

4.121.1 Detailed Description

Represents an endpoint for communication between two machines.

This class works as an adapter for the socket functions included on the sys/socket library.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.121.2 Constructor & Destructor Documentation

4.121.2.1 SocketBase::SocketBase (int *family* = AF_INET, int *type* = SOCK_STREAM, int *protocol* = 0)

Constructor.

Parameters

family	Socket family, default AF_INET.
type	Socket type, default SOCK_STREAM.
protocol	Socket protocol, default 0.

Exceptions

SysException	

4.121.2.2 SocketBase::SocketBase (int *hSocket*, Address & *addr*)

Constructor.

Sets the socket to the given handle with the given address.

Parameters

hSocket	Socket handle.
addr	Socket address.

4.121.3 Member Function Documentation

4.121.3.1 void SocketBase::Bind (int *port*)

Associates the socket with a local endpoint.

Parameters

port	Port where the socket will perform the connection.

4.121.3.2 int SocketBase::DoSend (char const * *buf*, int *bufSize*, int *flags* = 0) [protected]

Sends the data through the socket.

Exceptions

SysException	

4.121.3.3 int SocketBase::GetOption (int *level*, int *option*, char * *value*, int & *valueSize*) [protected]

Gets the value of a socket option.

Parameters

level	Protocol level at which the option resides.
option	Option name.
value	Buffer to save the value.
valueSize	Integer to save the size of the value.

Exceptions

SysException	

4.121.3.4 int SocketBase::GetReceiveTimeout ( )

Gets the ReceiveTimeout property.

Exceptions

SysException	

4.121.3.5 int SocketBase::GetSendTimeout ( )

Gets the SendTimeout property.

Exceptions

SysException	

4.121.3.6 void SocketBase::Listen (int *backLog*)

Sets the socket to a listening state.

Parameters

backLog	Maximum length of the queue for pending connections.

4.121.3.7 int SocketBase::Receive (char * *buf*, int *bufSize*, int *flags* = 0)

Gets data from the socket.

If no incoming data is available at the socket, the call blocks and waits for data to arrive.

Returns

Number of bytes received.

Exceptions

SysException	

4.121.3.8 int SocketBase::ReceiveN (char * *buf*, int *bufSize*, int *flags* = 0)

Gets data from the socket, performs multiple Receive() calls until the buffer is full.

Returns

Number of bytes received.

Exceptions

SysException	

4.121.3.9 void SocketBase::Send (char const * *buf*, int *bufSize*, int *flags* = 0)

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.121.3.10 void SocketBase::Send (std::string const & *str,* int *flags* = 0)

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.121.3.11 void SocketBase::Send (ByteStream & *stream,* int *flags* = 0)

Sends the data through the socket converted to network byte order.

Exceptions

SysException	

4.121.3.12 void SocketBase::SetKeepAlive (bool *value*)

Sets the KeepAlive property.

Exceptions

SysException	

4.121.3.13 void SocketBase::SetOption (int *level,* int *option,* char const * *value,* int *valueSize*) [protected]

Sets a socket option.

Parameters

level	Protocol level at which the option resides.
option	Option name.
value	Option value to set.
valueSize	Size of the value.

Exceptions

SysException	

4.121.3.14 void SocketBase::SetReceiveTimeout (int *timeoutMs*)

Sets the ReceiveTimeout property.

Exceptions

SysException	

4.121.3.15 void SocketBase::SetReuseAddress (bool *value*)

Sets the ReuseAddress property.

Exceptions

SysException	

4.121.3.16 void SocketBase::SetSendTimeout (int *timeoutMs*)

Sets the SendTimeout property.

Exceptions

SysException	

4.121.3.17 void SocketBase::SetTCPNoDelay (bool *value*)

Sets the TCPNoDelay property.

Exceptions

SysException	

The documentation for this class was generated from the following files:

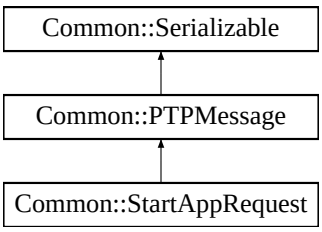
- Common/Socket.h
- Common/Socket.cpp

4.122 Common::StartAppRequest Class Reference

Represents a request to start the application.

`#include <PTPMsg.h>`

Inheritance diagram for Common::StartAppRequest:



Public Member Functions

- StartAppRequest (std::string const &appPath=std::string(), int argc=0, char const **argv=0, std::string const &analyzerHost=std::string())

  Constructor.
- ~StartAppRequest ()

  Destructor.
- PTPMsgType GetType () const

  Returns type of message (PTPStartApp).
- std::string const & GetAppPath () const

  Returns the application path.
- std::string const & GetAnalyzerHost () const

  Returns the analyzer host name.

- char ** GetArgs () const
  Returns a pointer to the array of arguments.
- int GetArgCount () const
  Returns number of arguments given.
- void Serialize (Serializer &out) const
  Sends the message.
- void DeSerialize (DeSerializer &in)
  Receives the message.

4.122.1 Detailed Description

Represents a request to start the application.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

4.122.2 Constructor & Destructor Documentation

4.122.2.1 StartAppRequest::StartAppRequest (std::string const & *appPath* = `std::string()`, int *argc* = `0`, char const ** *argv* = `0`, std::string const & *analyzerHost* = `std::string()`)

Constructor.

Parameters

appPath	Application path, default 0.
argc	Argument count, default 0.
argv	Arguments array, default 0.
analyzerHost	Host of the analyzer, default 0.

The documentation for this class was generated from the following files:

- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.123 Stats Struct Reference

Struct that stores the mean and standard deviation values.

`#include <FactoringTunlet_nw.h>`

Public Attributes

- double desv
  Standard deviation attribute.
- double mean
  Mean attribute.

4.123.1 Detailed Description

Struct that stores the mean and standard deviation values.

The documentation for this struct was generated from the following file:

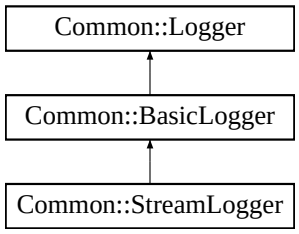
- Analyzer/FactoringTunlet_nw.h

4.124 Common::StreamLogger Class Reference

Stores the logged information into a stream.

`#include <Syslog.h>`

Inheritance diagram for Common::StreamLogger:



Public Member Functions

- StreamLogger (std::ostream &stream)
  Constructor.
- ~StreamLogger ()
  Destructor.
- void Log (LogEntry const &entry)
  Inserts an entry to the log.

Additional Inherited Members

4.124.1 Detailed Description

Stores the logged information into a stream.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following files:

- Common/Syslog.h
- Common/Syslog.cpp

4.125 Common::StringArray Class Reference

Container of strings.

`#include <StringArray.h>`

Public Member Functions

- StringArray (int size=0)
  Constructor.
- ~StringArray ()
  Destructor.
- void AddString (char const *s)
  Adds a string to the array.
- void Grow (int newSize)
  Increments max size of the array.
- int GetCount () const
  Returns number of strings currently stored.
- int GetSize () const
  Returns max size of the array.
- char const * GetString (int idx) const
  Returns string stored on the given position.
- char ** GetAccess () const
  Returns a pointer to the actual array.
- void Dump () const
  Writes current state of the array on the standard output.

4.125.1 Detailed Description

Container of strings.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.125.2 Constructor & Destructor Documentation

4.125.2.1 StringArray::StringArray (int *size* = 0)

Constructor.

Parameters

size	Size of the array, default 0.

The documentation for this class was generated from the following files:

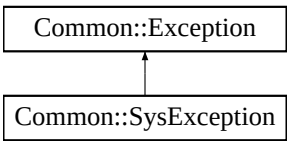
- Common/StringArray.h
- Common/StringArray.cpp

4.126 Common::SysException Class Reference

System exception.

`#include <SysException.h>`

Inheritance diagram for Common::SysException:



Public Member Functions

- SysException (std::string const &msg, std::string const &objName=std::string())

  Constructor.
- SysException (std::string const &msg, long errorCode)

  Constructor.
- void Display (std::ostream &os) const

  Displays exception message on the given output stream.
- void Display () const

  Displays exception message on the standard error output.
- std::string GetReason () const

  Returns a string containing the error message.

Additional Inherited Members

4.126.1 Detailed Description

System exception.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.126.2 Constructor & Destructor Documentation

4.126.2.1 Common::SysException::SysException (std::string const & *msg*, std::string const & *objName* = std::string ()) [inline]

Constructor.

Parameters

msg	Exception message.
objName	Name of the object causing the exception, "" by default.

4.126.2.2 Common::SysException::SysException (std::string const & *msg*, long *errorCode*) [inline]

Constructor.

Parameters

msg	Exception message.
errorCode	Exception error code.

4.126.3 Member Function Documentation

4.126.3.1 void Common::SysException::Display (std::ostream & *os*) const [virtual]

Displays exception message on the given output stream.

Parameters

os	Output stream to display the message.

Reimplemented from Common::Exception.

The documentation for this class was generated from the following files:

- Common/SysException.h
- Common/SysException.cpp

4.127 Common::Syslog Class Reference

Holds and manages a loggers on the system.

`#include <Syslog.h>`

Public Types

- typedef auto_vector< Logger > **LoggerVector**
- typedef auto_iterator< Logger > **LoggerIterator**

Static Public Member Functions

- static void Configure ()

  Configures the system with a default configuration.
- static void Configure (Config &cfg, string loggerName="")

  Configures the logger with a given Config.
- static void LogEvent (LogSeverity s, std::string const &message)

  Adds an entry with the given event to all the loggers.
- static void Debug (std::string const &message)

  Logs an event with DEBUG level of severity.
- static void Info (std::string const &message)

  Logs an event with INFO level of severity.

- static void Warn (std::string const &message)

  Logs an event with WARNING level of severity.
- static void Error (std::string const &message)

  Logs an event with ERROR level of severity.
- static void Fatal (std::string const &message)

  Logs an event with FATAL level of severity.
- static void Debug (char *formatStr,...)

  Logs an event with DEBUG level of severity.
- static void Info (char *formatStr,...)

  Logs an event with INFO level of severity.
- static void Warn (char *formatStr,...)

  Logs an event with WARNING level of severity.
- static void Error (char *formatStr,...)

  Logs an event with ERROR level of severity.
- static void Fatal (char *formatStr,...)

  Logs an event with FATAL level of severity.
- static bool CanWrite ()

  Returns true if the system log is enabled, false otherwise.
- static Logger const * GetLogger (std::string const &name)

  Returns logger with given name.
- static void AddLogger (LoggerPtr &logger)

  Adds a new logger to the system log.
- static void RemoveLogger (std::string const &name)

  Not implemented.
- static LoggerIterator GetLoggers ()

  Returns an iterator pointing to the first logger.

4.127.1 Detailed Description

Holds and manages a loggers on the system.

This is a static class that can't be instantiated.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.127.2 Member Function Documentation

4.127.2.1 void Syslog::Configure ( ) [static]

Configures the system with a default configuration.

This configuration uses an only Logger that outputs it's logs on the standard error output.

Exceptions

SysException	

4.127.2.2 void Syslog::Debug (std::string const & *message*) [inline], [static]

Logs an event with DEBUG level of severity.

Uses a string as a parameter.

4.127.2.3 void Syslog::Debug (char * *formatStr, ...*) [static]

Logs an event with DEBUG level of severity.

Uses a char * as a parameter.

4.127.2.4 void Syslog::Error (std::string const & *message*) [inline], [static]

Logs an event with ERROR level of severity.

Uses a string as a parameter.

4.127.2.5 void Syslog::Error (char * *formatStr, ...*) [static]

Logs an event with ERROR level of severity.

Uses a char * as a parameter.

4.127.2.6 void Syslog::Fatal (std::string const & *message*) [inline], [static]

Logs an event with FATAL level of severity.

Uses a string as a parameter.

4.127.2.7 void Syslog::Fatal (char * *formatStr, ...*) [static]

Logs an event with FATAL level of severity.

Uses a char * as a parameter.

4.127.2.8 void Syslog::Info (std::string const & *message*) [inline], [static]

Logs an event with INFO level of severity.

Uses a string as a parameter.

4.127.2.9 void Syslog::Info (char * *formatStr, ...*) [static]

Logs an event with INFO level of severity.

Uses a char * as a parameter.

4.127.2.10 void Syslog::Warn (std::string const & *message*) [inline], [static]

Logs an event with WARNING level of severity.

Uses a string as a parameter.

4.127.2.11 void Syslog::Warn (char * *formatStr, ...*) [static]

Logs an event with WARNING level of severity.

Uses a char * as a parameter.

The documentation for this class was generated from the following files:

- Common/Syslog.h
- Common/Syslog.cpp

4.128 Model::Task Class Reference

Encapsulates information to define the tasks that form the application. The data structure of a task consists of identification data (pid, mpiRank, name), status data, where it is running (host), which events are being collected from it and if it is either a master task or not. Provides methods to:

`#include <AppTask.h>`

Public Member Functions

- int GetPid () const

  Pid getter.
- int GetMpiRank () const

  MPI Rank getter.
- string GetName () const

  Name getter.
- Host & GetHost () const

  Host getter.
- bool IsRunning () const

  Indicates if the task is still running.
- bool IsMaster () const

  Indicates if the task is the master task.
- Status GetStatus () const

  Status getter.
- void AddEvent (Event const &e)

  Adds a definition of new event to be traced in this task.
- bool RemoveEvent (int eventId, InstrPlace place)

  Removes previously added event from this task.
- void LoadLibrary (string const &libPath)

  Loads a shared library to this task. This enables the Analyzer to load any additional code required for the tuning.
- void SetVariableValue (string const &varName, AttributeValue const &varValue, Breakpoint *brkpt)

  Modifies a value of a specified variable in the running task application process.
- void ReplaceFunction (string const &oldFunc, string const &newFunc, Breakpoint *brkpt)

  Replaces all calls to a function with calls to another one in this task.
- void InsertFunctionCall (string const &funcName, int nAttrs, Attribute *attrs, string const &destFunc, InstrPlace destPlace, Breakpoint *brkpt)

Inserts a new function invocation code at a given location in this task.

- void OneTimeFuncCall (string const &funcName, int nAttrs, Attribute *attrs, Breakpoint *brkpt)

  Inserts a new function invocation code in this task and invokes it once.
- void RemoveFuncCall (string const &funcName, string const &callerFunc, Breakpoint *brkpt)

  Removes all calls to a given function from the given caller function in this task. For example this method can be used to remove all flush() function calls from a debug() function.
- void FuncParamChange (string const &funcName, int paramIdx, int newValue, int *requiredOldValue, Breakpoint *brkpt)

  Sets the value of an input parameter of a given function in this task. This parameter value is modified before the function body is invoked. It is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.
- void SetTaskExitHandler (TaskHandler &h)

  Installs a callback function that is called when this task terminates.

Protected Member Functions

- Task (int pid, int mpiRank, string const &name, Host &h)

  Constructor.
- void SetMaster (bool value)

  Sets if this task is Master or not.
- ACProxy * GetACProxy ()
- void DispatchEvent (EventMsg const &msg)

  Finds the Event of the given message, gets its handler and Record and passes it to HandleEvent()

Friends

- class **Application**

4.128.1 Detailed Description

Encapsulates information to define the tasks that form the application. The data structure of a task consists of identification data (pid, mpiRank, name), status data, where it is running (host), which events are being collected from it and if it is either a master task or not. Provides methods to:

- Retrieve application information
- Monitoring: add/remove events to trace.
- Tuning: loading libraries, changing variables & parameter values, adding/removing function calls and calling them explicitly.

4.128.2 Constructor & Destructor Documentation

4.128.2.1 Task::Task (int *pid,* int *mpiRank,* string const & *name,* Host & *h*) [protected]

Constructor.

Parameters

pid	Globally unique task id.
mpiRank	Id associated to MPI
name	Process name.
h	Reference to the host object this task is running on.

4.128.3 Member Function Documentation

4.128.3.1 void Task::AddEvent (Event const & *e*)

Adds a definition of new event to be traced in this task.

Parameters

e	Event to be traced.

Returns

Number of tasks where the event tracing was added.

4.128.3.2 void Task::DispatchEvent (EventMsg const & *msg*) [protected]

Finds the Event of the given message, gets its handler and Record and passes it to HandleEvent()

Parameters

msg	

4.128.3.3 void Task::FuncParamChange (string const & *funcName*, int *paramIdx*, int *newValue*, int * *requiredOldValue*, Breakpoint * *brkpt*)

Sets the value of an input parameter of a given function in this task. This parameter value is modified before the function body is invoked. It is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

Parameters

funcName	Name of the function
paramIdx	Id of the parameter to change
newValue	New value for the parameter
requiredOld-Value	Required old value of the parameter to change
brkpt	—

Returns

Number of tasks where the parameter was changed.

4.128.3.4 ACProxy * Task::GetACProxy ( ) [protected]

Returns

ACProxy object of this task.

4.128.3.5 Host& Model::Task::GetHost ( ) const [inline]

Host getter.

Returns

Reference to the host object this task is running on.

4.128.3.6 int Model::Task::GetMpiRank ( ) const [inline]

MPI Rank getter.

Returns

MPI Rank of the task.

4.128.3.7 string Model::Task::GetName ( ) const [inline]

Name getter.

Returns

Process name

4.128.3.8 int Model::Task::GetPid ( ) const [inline]

Pid getter.

Returns

Globally unique process id

4.128.3.9 Status Model::Task::GetStatus ( ) const [inline]

Status getter.

Returns

Task Status information

4.128.3.10 void Task::InsertFunctionCall (string const & *funcName*, int *nAttrs*, Attribute * *attrs*, string const & *destFunc*, InstrPlace *destPlace*, Breakpoint * *brkpt*)

Inserts a new function invocation code at a given location in this task.

Parameters

funcName	Name of the function to call.
nAttrs	Number of parameters of the function.
attrs	Values for each parameter.
destFunc	Function where the calls will be placed.
destPlace	Place of the function where the calls will be placed.
brkpt	—

Returns

Number of tasks where the function calls were added.

4.128.3.11 bool Model::Task::IsMaster ( ) const [inline]

Indicates if the task is the master task.

Returns

True if master false otherwise.

4.128.3.12 bool Model::Task::IsRunning ( ) const [inline]

Indicates if the task is still running.

Returns

True if still running false otherwise.

4.128.3.13 void Task::LoadLibrary (string const & *libPath*)

Loads a shared library to this task. This enables the Analyzer to load any additional code required for the tuning.

Parameters

libPath	Path to the library.

Returns

Number of tasks where the library is loaded.

4.128.3.14 void Task::OneTimeFuncCall (string const & *funcName*, int *nAttrs*, Attribute * *attrs*, Breakpoint * *brkpt*)

Inserts a new function invocation code in this task and invokes it once.

Parameters

funcName	Name of the function to call
nAttrs	Number of arguments of the function
attrs	Values for each argument of the function
brkpt	—

Returns

Number of tasks where the function was called.

4.128.3.15 bool Task::RemoveEvent (int *eventId*, InstrPlace *place*)

Removes previously added event from this task.

Parameters

eventId	Id of the event
place	Place of the function where the event is recorded

Returns

Number of tasks where the event was removed.

4.128.3.16 void Task::RemoveFuncCall (string const & *funcName,* string const & *callerFunc,* Breakpoint * *brkpt*)

Removes all calls to a given function from the given caller function in this task. For example this method can be used to remove all flush() function calls from a debug() function.

Parameters

funcName	Name of the function
callerFunc	Function that calls the function that will be removed
brkpt	—

Returns

Number of tasks where the function call is removed.

4.128.3.17 void Task::ReplaceFunction (string const & *oldFunc,* string const & *newFunc,* Breakpoint * *brkpt*)

Replaces all calls to a function with calls to another one in this task.

Parameters

oldFunc	Name of the function to replace.
newFunc	Name of the new function.
brkpt	—

Returns

Number of tasks where the function calls were changed.

4.128.3.18 void Model::Task::SetMaster (bool *value*) [inline],[protected]

Sets if this task is Master or not.

Parameters

value	Determines if its Master or not.

4.128.3.19 void Task::SetVariableValue (string const & *varName,* AttributeValue const & *varValue,* Breakpoint * *brkpt*)

Modifies a value of a specified variable in the running task application process.

Parameters

varName	Name of the variable.
varValue	New value for the variable.
brkpt	—

Returns

Number of tasks where the values were changed.

The documentation for this class was generated from the following files:

- Analyzer/AppTask.h
- Analyzer/AppTask.cpp

4.129 Task Class Reference

Represents each of the processes that we can modify using Dyninst.

`#include <Task.h>`

Public Member Functions

- Task (const std::string &path, char *args[], TimeValue const &clockDiff, string const &analyzerHost, int analyzerPort, int debugLevel, int debugStdErr, string const &DMLibName)

  Constructor.
- ~Task ()

  Destructor.
- int GetPid ()

  Getter of the identifier of the process being modified.
- int GetMpiRank ()

  Getter of the MPIRank attribute.
- void Continue ()

  Restarts the execution of the process after breakpoint.
- void WaitFor ()

  Waits for the process to be terminated.
- DiProcess & GetProcess ()

  Getter of the variable _process.
- DiImage & GetImage ()

  Getter of the variable _image.
- TaskInstr & GetInstr ()

  Getter of the variable _instr.
- bool IsStopped ()

  Checks if the process is running.
- bool Terminate ()

  Terminates a running process and invokes the callback function if exists.
- bool IsTerminated ()

  Checks if the process is terminated.
- void AddDelayedTuning (Common::TuningRequest *req)

  Adds a tuning request to the pending list.
- bool IsStoppedOnBreakpoint ()

  Checks if the process is stopped in a breakpoint.
- void ProcessBreakpoint (Tuner &t)

  It is called when the process hits a breakpoint and needs to be handled.
- void UnloadLibrary ()

  Unload the DMLib from a process which has terminated.

4.129.1 Detailed Description

Represents each of the processes that we can modify using Dyninst.

Provides the necessary function to manage the process to be modified and to control its execution during the modifications.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.129.2 Constructor & Destructor Documentation

4.129.2.1 Task::Task (const std::string & *path,* char * *args[],* TimeValue const & *clockDiff,* string const & *analyzerHost,* int *analyzerPort,* int *debugLevel,* int *debugStdErr,* string const & *DMLibName*)

Constructor.

Parameters

path	Path to the executable of the application.
args[]	Array that contains the arguments with which the application should be executed.
clockDiff	Correction of the clock difference.
anayzerHost	Address of the host in which the analyzer is running.
analyzerPort	Port through which the analyzer communicates.
debugLevel	Selected level of debugging.
debugStdErr	Debug messages output.
DMLibName	Name of the dynamic library to be loaded into the process.

4.129.3 Member Function Documentation

4.129.3.1 void Task::AddDelayedTuning (Common::TuningRequest * *req*)

Adds a tuning request to the pending list.

Parameters

req	Request for tuning procedure.

4.129.3.2 DiImage& Task::GetImage ( ) [inline]

Getter of the variable _image.

Returns

Image of the process that is being modified.

4.129.3.3 TaskInstr& Task::GetInstr ( ) [inline]

Getter of the variable _instr.

Returns

Instrumentation to be inserted.

4.129.3.4 int Task::GetPid ( ) [inline]

Getter of the identifier of the process being modified.

Returns

Pid of the process being modified.

4.129.3.5 DiProcess& Task::GetProcess ( ) [inline]

Getter of the variable _process.

Returns

Process that is being modified.

4.129.3.6 bool Task::IsStopped ( ) [inline]

Checks if the process is running.

Returns

False if its running, true if stopped.

4.129.3.7 bool Task::IsStoppedOnBreakpoint ( ) [inline]

Checks if the process is stopped in a breakpoint.

Returns

True if process is currently stopped and situated in a breakpoint .

4.129.3.8 bool Task::IsTerminated ( ) [inline]

Checks if the process is terminated.

Returns

True if the process has exited.

4.129.3.9 void Task::ProcessBreakpoint (Tuner & *t*)

It is called when the process hits a breakpoint and needs to be handled.

Parameters

t	Tuner that will apply the changes specified in the request for each task.

4.129.3.10 bool Task::Terminate ( ) [inline]

Terminates a running process and invokes the callback function if exists.

Returns

True for success, false for failure.

The documentation for this class was generated from the following files:

- AC/Task.h
- AC/Task.cpp

4.130 TaskCollection Class Reference

Groups task in a single, easy to handle, collection.

`#include <Tasks.h>`

Public Types

- enum { **NotFound** = -1 }

Public Member Functions

- TaskCollection ()

  Constructor.
- void Add (auto_ptr< Task > &task)

  Adds a new task to the collection.
- void Delete (int index)

  Removes a task from the collection.
- void Clear ()

  Erases all the elements of the array.
- Task const * operator[] (int index) const

  Enables the use of the [] to select an element from the collection.
- Task * operator[] (int index)

  Enables the use of the [] to select an element from the collection.
- int FindByPid (int pid)

  Finds task by its PID and returns its index.
- int GetCount () const

  Getter of the number of tasks contained.
- Task & GetByPid (int pid)

  Looks for a task among the ones stored and returns its reference if found.

4.130.1 Detailed Description

Groups task in a single, easy to handle, collection.

Provides collection methods to a group of tasks to facilitate the handling of many tasks at once.

4.130.2 Member Function Documentation

4.130.2.1 void TaskCollection::Add (auto_ptr< Task > & *task*) [inline]

Adds a new task to the collection.

Parameters

task	Pointer to the task to be added.

4.130.2.2 void TaskCollection::Delete (int *index*) [inline]

Removes a task from the collection.

Parameters

index	Position in the array of the task to be removed.

4.130.2.3 int TaskCollection::FindByPid (int *pid*) [inline]

Finds task by its PID and returns its index.

Checks if a task with the given PID is being executed by the AC.

Parameters

pid	ID of the process which is executing the task.

Returns

NotFound if a task with given PID is not stored in the collection

4.130.2.4 Task& TaskCollection::GetByPid (int *pid*) [inline]

Looks for a task among the ones stored and returns its reference if found.

Parameters

pid	ID of the process executing the desired task.

Returns

The task if it's found.

Exceptions

Task not found	exception if there is no task with the required PID.

4.130.2.5 int TaskCollection::GetCount ( ) const [inline]

Getter of the number of tasks contained.

Returns

number of stored tasks

4.130.2.6 Task const* TaskCollection::operator[] (int *index*) const [inline]

Enables the use of the [] to select an element from the collection.

Parameters

index	Position in the array.

Returns

Constant value of a pointer to the selected task.

4.130.2.7 Task* TaskCollection::operator[] (int *index*) [inline]

Enables the use of the [] to select an element from the collection.

Parameters

index	Position in the array.

Returns

Pointer to the selected task.

The documentation for this class was generated from the following file:

- AC/Tasks.h

4.131 TaskExitHandler Class Reference

Contains a virtual function to handle the exit of a task.

`#include <TaskManager.h>`

Public Member Functions

- virtual void HandleTaskExit (Task const &task, int exitCode)=0

  Installs a callback function that is called when the task terminates.

4.131.1 Detailed Description

Contains a virtual function to handle the exit of a task.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

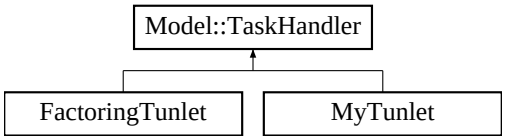
- AC/TaskManager.h

4.132 Model::TaskHandler Class Reference

Abstract class that provides methods to determine if a task is started or terminated.

`#include <AppTask.h>`

Inheritance diagram for Model::TaskHandler:



Public Member Functions

- virtual void TaskStarted (Task &t)=0

  Called when a new task is started.
- virtual void TaskTerminated (Task &t)=0

  Called when a task is terminated.

4.132.1 Detailed Description

Abstract class that provides methods to determine if a task is started or terminated.

4.132.2 Member Function Documentation

4.132.2.1 virtual void Model::TaskHandler::TaskStarted (Task & *t*) [pure virtual]

Called when a new task is started.

Parameters

t	Started task object.

Implemented in FactoringTunlet, and MyTunlet.

4.132.2.2 virtual void Model::TaskHandler::TaskTerminated (Task & *t*) [pure virtual]

Called when a task is terminated.

Parameters

t	Terminated task object.

Implemented in FactoringTunlet, and MyTunlet.

The documentation for this class was generated from the following file:

- Analyzer/AppTask.h

4.133 TaskInstr Class Reference

Adds and remove instrumentation from the process in execution.

`#include <TaskInstr.h>`

Public Member Functions

- TaskInstr ()

  Constructor.
- ~TaskInstr ()

  Destructor.
- int GetSize () const

  Getter of the size of the variable _map.
- void Add (int eventId, string const &functionName, InstrPlace instrPlace, BPatchSnippetHandle *handler)

  Adds a new snippet (handler) to the group in the map under the eventId key.
- void Remove (int eventId, InstrPlace instrPlace)

  Eliminates all the snippets to be inserted on the same event and place.
- InstrGroup * FindGroup (int eventId)

  Finds an instrumentation group for a given eventId.
- void SetBreakpoint (BPatchSnippetHandle *h)

  Setter of the variable _brkptHandle which represents a place in the process where it should be stopped.
- BPatchSnippetHandle * GetBreakpoint ()

  Getter of the variable _brkptHandle which represents a place in the process where it should be stopped.

4.133.1 Detailed Description

Adds and remove instrumentation from the process in execution.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.133.2 Member Function Documentation

4.133.2.1 void TaskInstr::Add (int *eventId,* string const & *functionName,* InstrPlace *instrPlace,* BPatchSnippetHandle * *handler*)

Adds a new snippet (handler) to the group in the map under the eventId key.

If a group doesn't exist with that key one is created and added to the map.

Parameters

eventId	Identifier of the event used as key to store the instrumentation groups in the map.
functionName	Name of the function to be modified.
instrPlace	Place of the function where the snippet will be inserted.
handler	Handle of the snippet to be inserted.

4.133.2.2 InstrGroup * TaskInstr::FindGroup (int *eventId*)

Finds an instrumentation group for a given eventId.

Parameters

eventId	Key to find the instrumentation group in the map.

Returns

IntrGroup object that contains all the snippets to be added on a given event.

4.133.2.3 BPatchSnippetHandle* TaskInstr::GetBreakpoint ( ) [inline]

Getter of the variable _brkptHandle which represents a place in the process where it should be stopped.

Returns

Handle of the breakpoint.

4.133.2.4 int TaskInstr::GetSize ( ) const [inline]

Getter of the size of the variable _map.

Returns

size of the map _map.

4.133.2.5 void TaskInstr::Remove (int *eventId,* InstrPlace *instrPlace*)

Eliminates all the snippets to be inserted on the same event and place.

Parameters

eventId	Identifier of the event used as key to find and remove the instrumentation groups in the map.
instrPlace	Place of the function where the snippet would have been.

Exceptions

Remove	cannot find instrumentation group.

The documentation for this class was generated from the following files:

- AC/TaskInstr.h
- AC/TaskInstr.cpp

4.134 TaskManager Class Reference

Single class that starts and handles all the tasks.

`#include <TaskManager.h>`

4.134.1 Detailed Description

Single class that starts and handles all the tasks.

Originally this class started the applications in each node using MPI, now this is done externally.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- AC/TaskManager.h

4.135 Model::Tasks Class Reference

Tasks encapsulate methods to work with lists of Task objects. The data structure to hold the information is an auto_vector. This class provides methods to add, remove, access Task objects in an array. It also provides methods to find Tasks and for measuring the array.

`#include <AppTask.h>`

Public Types

- enum { **NotFound** = -1 }

Public Member Functions

- Tasks ()

  Constructor.
- void Add (auto_ptr< Task > &task)

  Adds a task to the list.
- void Delete (int index)

  Deletes a task from the list.
- Task const * operator[] (int index) const

  Accessor to the array.
- Task * operator[] (int index)

  Accessor to the array.
- int FindById (int id)

  Finds a task by ID and returns its index.
- int Size () const

  Size getter.
- Task & GetById (int id)

  Finds a task by ID and returns a reference to it.
- auto_ptr< Task > Remove (int id)

  Removes a task by ID from the list.

4.135.1 Detailed Description

Tasks encapsulate methods to work with lists of Task objects. The data structure to hold the information is an auto_vector. This class provides methods to add, remove, access Task objects in an array. It also provides methods to find Tasks and for measuring the array.

4.135.2 Member Function Documentation

4.135.2.1 void Tasks::Add (auto_ptr< Task > & *task*)

Adds a task to the list.

Parameters

task	The task to add.

4.135.2.2 void Tasks::Delete (int *index*)

Deletes a task from the list.

Parameters

index	Position in the list of the task to delete.

4.135.2.3 int Tasks::FindById (int *id*)

Finds a task by ID and returns its index.

Parameters

id	Identifier of the task.

Returns

Index of the task or NotFound if a task with given ID is not stored in the collection.

4.135.2.4 Task & Tasks::GetById (int *id*)

Finds a task by ID and returns a reference to it.

Parameters

id	Identifier of the task (pid).

Returns

A reference to the found task.

Exceptions

Exception	if not found.

4.135.2.5 Task const * Tasks::operator[] (int *index*) const

Accessor to the array.

Parameters

index	Position of the array to access.

Returns

Pointer to the specified position

4.135.2.6 Task * Tasks::operator[] (int *index*)

Accessor to the array.

Parameters

index	Position of the array to access.

Returns

Pointer to the specified position

4.135.2.7 auto_ptr< Task > Tasks::Remove (int *id*)

Removes a task by ID from the list.

Parameters

id	Identifier of the task.

Returns

A reference to the removed task or a null pointer if it was not present.

4.135.2.8 int Tasks::Size ( ) const

Size getter.

Returns

Number of stored tasks

The documentation for this class was generated from the following files:

- Analyzer/AppTask.h
- Analyzer/AppTask.cpp

4.136 TaskStats Class Reference

Class that deals with the statistics of a certain task e.g. the communication costs, optimal fragment size or the total number of changes.

Public Member Functions

- TaskStats (int tid)

  Constructor.
- void ChangeFragSize (int size)

  Changes the fragment size to the one provided by the parameter and increments _nChanges by 1.
- void Update (int count, int size)

  Updates the number of messages by count and the sum size by size
- double GetCommCost () const

  Returns the communication costs by multiplying the number of messages and the sum size by two factors and then adding their results together.
- int GetOptimalFragSize () const

  Finds the optimal fragment size based on the current number of messages and the sum size.
- int GetCurrentFragSize () const

  Getter of the current fragment size.
- int GetNumChanges () const

  Getter of the current number of changes.
- int GetTid () const

  Getter of the Tid.

4.136.1 Detailed Description

Class that deals with the statistics of a certain task e.g. the communication costs, optimal fragment size or the total number of changes.

Version

1.0b

Author

Ania Sikora, 2002

Since

1.0b

4.136.2 Constructor & Destructor Documentation

4.136.2.1 TaskStats::TaskStats (int *tid*) [inline]

Constructor.

Parameters

tid	Id to assign to the task

4.136.3 Member Function Documentation

4.136.3.1 void TaskStats::ChangeFragSize (int *size*) [inline]

Changes the fragment size to the one provided by the parameter and increments _nChanges by 1.

Parameters

size	Value of the new frag size

4.136.3.2 double TaskStats::GetCommCost ( ) const [inline]

Returns the communication costs by multiplying the number of messages and the sum size by two factors and then adding their results together.

Returns

Communication costs in *ms*

4.136.3.3 int TaskStats::GetCurrentFragSize ( ) const [inline]

Getter of the current fragment size.

Returns

_fragSize

4.136.3.4 int TaskStats::GetNumChanges ( ) const [inline]

Getter of the current number of changes.

Returns

_nChanges

4.136.3.5 int TaskStats::GetOptimalFragSize ( ) const [inline]

Finds the optimal fragment size based on the current number of messages and the sum size.

Returns

Optimal fragment size

4.136.3.6 int TaskStats::GetTid ( ) const [inline]

Getter of the Tid.

Returns

_tid

4.136.3.7 void TaskStats::Update (int *count*, int *size*) [inline]

Updates the number of messages by *count* and the sum size by *size*

Parameters

count	Increments the number of messages by its value
size	Increments the sum size by its value

The documentation for this class was generated from the following file:

- Analyzer/Analysis.cpp

4.137 Common::Thread Class Reference

Posix thread.

`#include <Thread.h>`

Public Member Functions

- Thread (void ∗(∗pFun)(void ∗arg), void ∗pArg)
  Constructor.
- void WaitForDeath ()
  Waits for termination of the thread.
- void Exit ()
  Stops the thread execution.

4.137.1 Detailed Description

Posix thread.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.137.2 Constructor & Destructor Documentation

4.137.2.1 Thread::Thread (void *(*)(void *arg) *pFun*, void * *pArg*)

Constructor.

Parameters

arg	Pointer to the function to run.
pArg	Pointer to the function arguments.

Exceptions

SysException	

4.137.3 Member Function Documentation

4.137.3.1 void Thread::WaitForDeath ( )

Waits for termination of the thread.

Exceptions

SysException	

The documentation for this class was generated from the following files:

- Common/Thread.h
- Common/Thread.cpp

4.138 Common::TimeValue Class Reference

Stores a time value up to microseconds.

`#include <TimeValue.h>`

Public Member Functions

- TimeValue ()
  Constructor.
- TimeValue (long secs, long usecs=0)
  Constructor.
- TimeValue (TimeValue const &t)
  Constructor.
- TimeValue (timeval const &t)
  Constructor.
- TimeValue (struct timeb const &tb)
  Constructor.

- TimeValue & **operator=** (TimeValue const &t)
- TimeValue & **operator=** (timeval const &t)
- void **operator+=** (TimeValue const &t)
- void **operator-=** (TimeValue const &t)
- long GetSeconds () const

  Returns seconds of the object.
- long GetMicroseconds () const

  Returns microseconds of the object without counting on the seconds.
- long_t GetMilliseconds () const

  Returns milliseconds of the object.
- long_t GetTotalMicroseconds () const

  Returns microseconds of the object.
- void Zero ()

  Sets seconds and microseconds to 0.
- void SetCurrentTime ()

  Sets values to current time.
- **operator struct timeval** * ()

Static Public Member Functions

- static TimeValue Now ()

  Returns current time.

Friends

- TimeValue **operator+** (TimeValue const &t1, TimeValue const &t2)
- TimeValue **operator-** (TimeValue const &t1, TimeValue const &t2)
- TimeValue **operator**/ (TimeValue const &t1, int value)
- bool **operator**< (TimeValue const &t1, TimeValue const &t2)
- bool **operator**> (TimeValue const &t1, TimeValue const &t2)
- bool **operator==** (TimeValue const &t1, TimeValue const &t2)
- bool **operator!=** (TimeValue const &t1, TimeValue const &t2)

4.138.1 Detailed Description

Stores a time value up to microseconds.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

4.138.2 Constructor & Destructor Documentation

4.138.2.1 Common::TimeValue::TimeValue ( ) [inline]

Constructor.

By default the value of the time value is 0 seconds and 0 microseconds.

4.138.2.2 Common::TimeValue::TimeValue (long *secs*, long *usecs* = 0) [inline]

Constructor.

Sets seconds and microseconds (microseconds are 0 by default).

Parameters

secs	Seconds.
usecs	Microseconds.

4.138.2.3 Common::TimeValue::TimeValue (TimeValue const & *t*) [inline]

Constructor.

Assigns the values of the TimeValue object.

4.138.2.4 Common::TimeValue::TimeValue (timeval const & *t*) [inline]

Constructor.

Assigns values of the given structure.

4.138.2.5 Common::TimeValue::TimeValue (struct timeb const & *tb*) [inline]

Constructor.

Assigns values of the given structure converted to the used format.

The documentation for this class was generated from the following files:

- Common/TimeValue.h
- Common/TimeValue.cpp

4.139 Tuner Class Reference

Contains the tools necessary to handle the requests from the Analyzer. Performs the different tuning jobs and handles breakpoints by delaying the tuning until the target point is reached.

`#include <Tuner.h>`

Public Member Functions

- Tuner (TaskCollection &tasks)

  Constructor.
- void Process (Common::TuningRequest *req)

  Handles the tuning requests received from the Analyzer.
- void RemoveLastBreakpoint (Task &task)

Removes the breakpoint snippet inserted in the process.

4.139.1 Detailed Description

Contains the tools necessary to handle the requests from the Analyzer. Performs the different tuning jobs and handles breakpoints by delaying the tuning until the target point is reached.

Version

1.0

Since

1.0

Author

Ania Sikora, 2002

4.139.2 Constructor & Destructor Documentation

4.139.2.1 Tuner::Tuner (TaskCollection & *tasks*) [inline]

Constructor.

Parameters

tasks	

4.139.3 Member Function Documentation

4.139.3.1 void Tuner::Process (Common::TuningRequest * *req*)

Handles the tuning requests received from the Analyzer.

Applies the required changes in the target process. If there's a breakpoint for the request instead of processing the request it is pushed back to the pending requests list and a breakpoint is inserted and its handler is passed to the task.

When the breakpoint in the process is reached its removed and the tuning is performed.

After applying changes the request is deleted.

Parameters

req	Tuning request to be applied on the target process.

4.139.3.2 void Tuner::RemoveLastBreakpoint (Task & *task*)

Removes the breakpoint snippet inserted in the process.

To be called when a breakpoint is processed and therefore has not further purpose.

Parameters

task	Task to which executes the process.

The documentation for this class was generated from the following files:

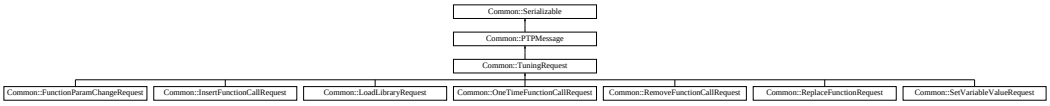
- AC/Tuner.h
- AC/Tuner.cpp

4.140 Common::TuningRequest Class Reference

Encapsulates a tuning request from the analyzer.

`#include <PTPMsg.h>`

Inheritance diagram for Common::TuningRequest:



Public Member Functions

- ~TuningRequest ()

  Destructor.
- int GetPid () const

  Returns the pid of the process associated with the message.
- Breakpoint * GetBreakpoint () const

  Returns breakpoint.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.
- void ClearBreakpoint ()

  Clears breakpoint.

Protected Member Functions

- **TuningRequest** (int pid, Breakpoint const *brkpt=0)

Protected Attributes

- int **_pid**
- Breakpoint * **_brkpt**

4.140.1 Detailed Description

Encapsulates a tuning request from the analyzer.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2003

The documentation for this class was generated from the following files:

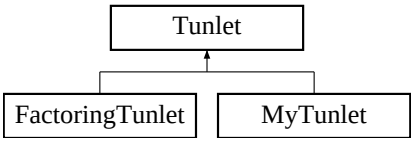
- Common/PTPMsg.h
- Common/PTPMsg.cpp

4.141 Tunlet Class Reference

Tunlet class that contains the virtual methods to be inherited.

`#include <Tunlet.h>`

Inheritance diagram for Tunlet:



Public Member Functions

- virtual void Initialize (Model::Application &app)=0

  Initializes the tunlet.
- virtual void BeforeAppStart ()

  Asserts that _app != 0 and sets the task handler of the app to the current one.
- virtual void AppStarted ()

  Sets tl=0.3 and lambda=0.5.
- virtual void Destroy ()=0

  Sets _app = 0.
- virtual void Initialize (Model::Application &app)=0

  Initializes the tunlet.
- virtual void BeforeAppStart ()

  Asserts that _app != 0 and sets the task handler of the app to the current one.
- virtual void Destroy ()=0

  Sets _app = 0.
- virtual void Initialize (Model::Application &app)=0

  Virtual Initialize method.
- virtual void BeforeAppStart ()

  Virtual BeforeAppStart method.
- virtual void AppStarted ()

  Virtual AppStarted method.
- virtual void Destroy ()=0

  Virtual Destroy method.

4.141.1 Detailed Description

Tunlet class that contains the virtual methods to be inherited.

4.141.2 Member Function Documentation

4.141.2.1 virtual void Tunlet::Initialize (Model::Application & *app*) [pure virtual]

Initializes the tunlet.

Parameters

app	App associated to the tunlet

Implemented in FactoringTunlet, and MyTunlet.

4.141.2.2 virtual void Tunlet::Initialize (Model::Application & *app*) [pure virtual]

Initializes the tunlet.

Parameters

app	App associated to the tunlet

Implemented in FactoringTunlet, and MyTunlet.

The documentation for this class was generated from the following files:

- Analyzer/FactoringTunlet_nw.h
- Analyzer/MyTunlet.h
- Analyzer/Tunlet.h

4.142 TunletContainer Class Reference

TO BE IMPLEMENTED.

`#include <TunletsContainer.h>`

4.142.1 Detailed Description

TO BE IMPLEMENTED.

The documentation for this class was generated from the following file:

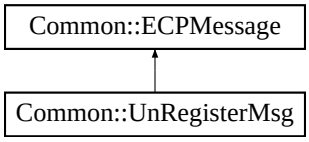
- Analyzer/TunletsContainer.h

4.143 Common::UnRegisterMsg Class Reference

Represents message that is sent when DMLib is unregistered with analyzer.

`#include <ECPMsg.h>`

Inheritance diagram for Common::UnRegisterMsg:



Public Member Functions

- UnRegisterMsg (int pid=0)

  Constructor.
- ECPMsgType GetType () const

  Returns the type of event.
- int GetPid () const

  Returns Id of the process where the library will be unregistered.
- void Serialize (Serializer &out) const

  Sends the message.
- void DeSerialize (DeSerializer &in)

  Receives the message.

Additional Inherited Members

4.143.1 Detailed Description

Represents message that is sent when DMLib is unregistered with analyzer.

Version

1.0b

Since

1.0b

Author

Ania Sikora, 2002

The documentation for this class was generated from the following file:

- Common/ECPMsg.h

4.144 Ventana Struct Reference

Window that will store statistics of the workers.

`#include <FactoringTunlet_nw.h>`

Public Attributes

- int TAM

  Size of the window.
- Stats * historico

  Pointer to a set of stats like the mean, std deviation etc.

4.144.1 Detailed Description

Window that will store statistics of the workers.

The documentation for this struct was generated from the following file:

- Analyzer/FactoringTunlet_nw.h

4.145 WorkerData Class Reference

Worker task statistics for a single batch.

`#include <FactoringStats_nw.h>`

Public Member Functions

- WorkerData ()

  Constructor.
- void OnCalcStart (long_t time)

  Sets the calculation start time.
- void OnCalcEnd (long_t time)

  Asserts that the final time is greater than the starting one, computes the elapsed time in milliseconds and sets the flag _taken to 1.
- bool IsComplete () const

  Checks if the WorkerData is complete.
- int GetNumProcessedTuples () const

  Getter of the total number of tasks received.
- int GetSizeProcessedTuples () const

  Getter of the task size in bytes.
- double GetTotalCalcTime () const

  Getter of the total computing time in ms.
- void OnTupleStart (int nTuples, int sizeBytes)

  Initializes the tuple by setting the number of tasks received to nTuples and the size of the tasks to sizeBytes. It also sets the flag _initialized to 1.
- bool IsTaken ()

  Returns the value of the _taken flag.
- bool IsInitialized ()

  Returns if the task has been initialized or not.

4.145.1 Detailed Description

Worker task statistics for a single batch.

4.145.2 Member Function Documentation

4.145.2.1 int WorkerData::GetNumProcessedTuples ( ) const [inline]

Getter of the total number of tasks received.

Returns

_nNumTaskReceived

4.145.2.2 int WorkerData::GetSizeProcessedTuples ( ) const [inline]

Getter of the task size in bytes.

Returns

_sizeTaskBytes

4.145.2.3 double WorkerData::GetTotalCalcTime ( ) const [inline]

Getter of the total computing time in ms.

Returns

_ComputingTimeMs

4.145.2.4 bool WorkerData::IsComplete ( ) const [inline]

Checks if the WorkerData is complete.

Returns

Returns true if the start and end calc times are greater than 0 and if the worker was initialized

4.145.2.5 bool WorkerData::IsInitialized ( ) [inline]

Returns if the task has been initialized or not.

Returns

_initialized

4.145.2.6 bool WorkerData::IsTaken ( ) [inline]

Returns the value of the _taken flag.

Returns

_taken

4.145.2.7 void WorkerData::OnCalcEnd (long_t *time*)

Asserts that the final time is greater than the starting one, computes the elapsed time in milliseconds and sets the flag _taken to 1.

Parameters

time	Final time of computation

4.145.2.8 void WorkerData::OnCalcStart (long_t *time*)

Sets the calculation start time.

Parameters

time	Time to start in ms

4.145.2.9 void WorkerData::OnTupleStart (int *nTuples,* int *sizeBytes*)

Initializes the tuple by setting the number of tasks received to nTuples and the size of the tasks to sizeBytes. It also sets the flag _initialized to 1.

Parameters

nTuples	Number of tasks received
sizeBytes	Size of the task in bytes

The documentation for this class was generated from the following files:

- Analyzer/FactoringStats_nw.h
- Analyzer/FactoringStats_nw.cpp