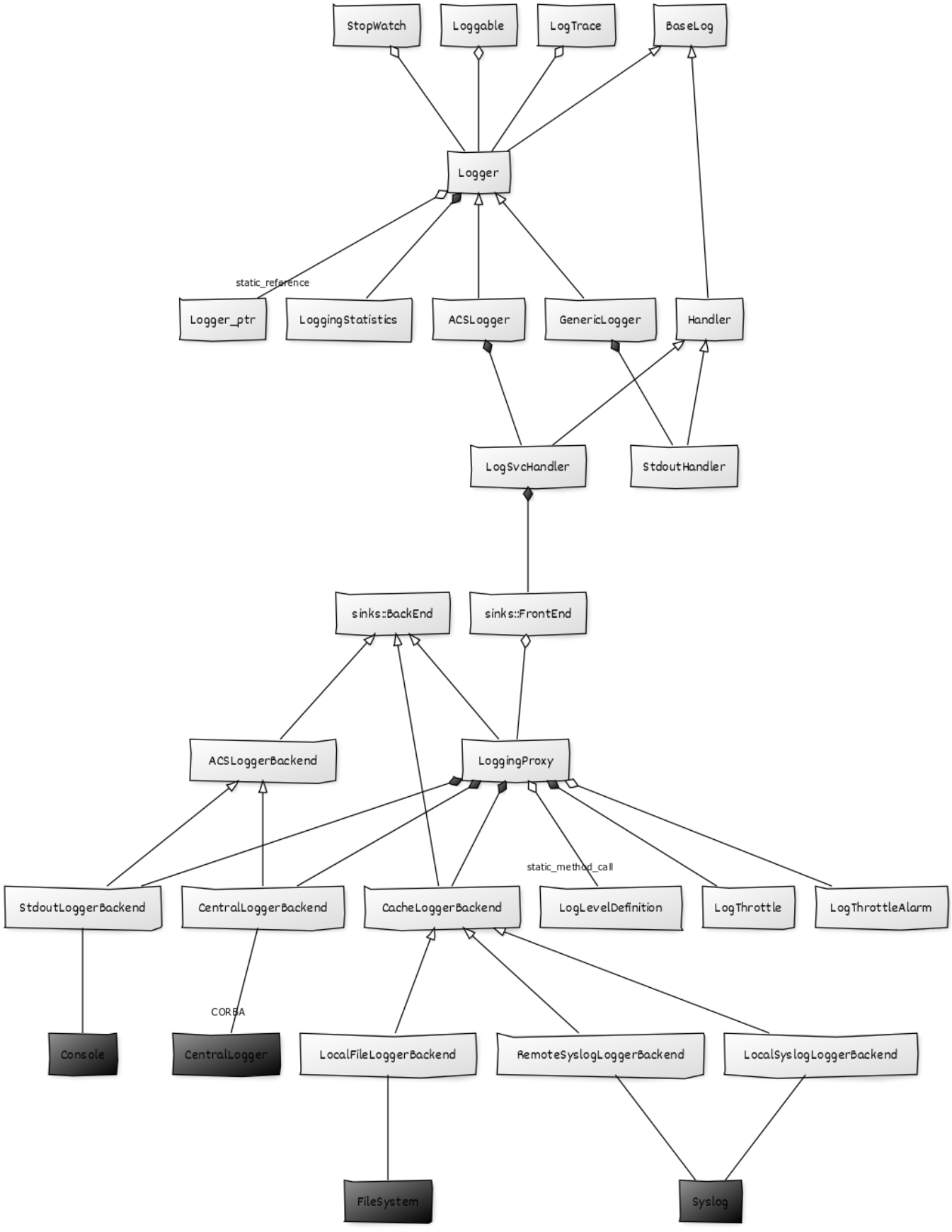
This proposal of the C++ implementation of the logging system is based in Boost Log and will be part of the ACS/LGPL/CommonSoftware/logging module. Additionally there's the ACSLog service in ACS/LGPL/CommonSoftware/acslog module, which offers an entry point for using the logging system by any part of the system. Specifically it is used by Python logging framework for connecting with the central logger.

Class Diagram


StopWatch
Loggable
LogTrace
BaseLog
Logger
static_reference
Logger_ptr
LoggingStatistics
ACSLogger
GenericLogger
Handler
LogSvcHandler
StdoutHandler
sinks::BackEnd
sinks::FrontEnd
ACSLoggerBackend
LoggingProxy
static_method_call
StdoutLoggerBackend
CentralLoggerBackend
CacheLoggerBackend
LogLevelDefinition
LogThrottle
LogThrottleAlarm
CORBA
Console
CentralLogger
LocalFileLoggerBackend
RemoteSyslogLoggerBackend
LocalSyslogLoggerBackend
FileSystem
Syslog
CREATED WITH YUML

Details

BaseLog

The base log is an abstract parent class for several classes in the logging implementation and it also defines some basic information:

- Priority enum defining log levels
- FIELD_UNAVAILABLE
- GLOBAL_LOGGER_NAME
- ANONYMOUS_LOGGER_NAME
- STATIC_LOGGER_NAME

As a parent class it defines several interface methods as well as implementing a couple of implementation-agnostic methods to be used by child classes.

It holds a reference to a LoggingStatistics instance, but it does not use it directly, this is to be used by child classes to calculate statistics about the logging calls. A recommendation is to move this reference out of the BaseLog class to the Logger class, because the BaseLog class is also inherited by the Handler class and their instances don't require the LoggingStatistics reference.

LoggingStatistics

The LoggingStatistics class is utility class for calculating and storing statistics about the logs that have been processed by a specific logger. There are some configurable parameters:

- statisticsCalculationPeriod: Time used for retrieving data before calculating statistics. Default 10 minutes.
- statisticsGranularity: Time unit used for calculations (i.e. logs per second, logs per minute, etc.). Default 1 second.

It calculates 5 statistics:

- messageStatistics: Number of log messages per granularity unit.
- errorStatistics: Number of log errors per granularity unit.
- messageIncrement: Number of log messages since last calculation.
- errorIncrement: Number of log errors since last calculation.
- actualStatisticsPeriod: Actual period of time which was used to gather data before calculating statistics.
    - Usually close to statisticsCalculationPeriod
    - May be different when there's a change of configuration or during the destruction of the logger

Logger

This is the main class in the logging system, which is eventually exposes its interface to other components:

- addHandler: Adding multiple handlers for the log messages
- removeHandler: Remove a registered handler
- setLevels: Setting the log level priority for remote and local handling
- log: Delegates the logging task to each of its logging handlers
    - Handles interaction with loggingStatistics instance (stats) as well
- Other getter and setter methods

Additionally, it handles three singleton loggers (global, anonymous, static). The anonymous and static loggers are initialized the first time they're used as child of the global logger. The global logger is created the first time it's used.

Logger_ptr

Logger_ptr is a sub-class from Logger class, which is used as a singleton to store references to global, anonymous, static and a set of named loggers. This is only used from the Logger class.

LogTrace

A convenience class that logs when its constructor and destructor are called, including the time taken between the two calls.

StopWatch

A convenience class that logs when its constructor and destructor are called. It monitors the time taken between the two calls and reports if the configured limit time is exceeded.

Loggable

A small utility class that enables class logging by providing a simple getLogger() method. The logger to be used is either the global logger or a named logger obtained when the constructor is executed.

GenericLogger

A logger that is initialized with an StdoutHandler. It offers the getLogger method returning named loggers as GenericLogger instances.

ACSLogger

A logger that is initialized with the LogSvcHandler. It offers the getLogger method returning named loggers as ACSLogger instances.

Additionally it offers mutex acquire/release methods for convenience.

Handler

The Handler base class defines a set of common methods that all handlers should have:

- setLevels: Set the remote and local levels for handlers if they're matched with the given type (Implementation specific). By default handlers set the general level (to the given remote level) and ignore the local and type parameters as well as avoid interacting with the remoteLevel and localLevel methods and variables.
- (get|set)Level: Interact with the general log level for the handler.
- (get|set)RemoteLevel/(get|set)LocalLevel: Interact with the remote or local level for the handler.
- (get|set)RemoteLevelType/(get|set)LocalLevelType: Interact with the remote or local level type for the handler.

StdoutHandler

Extends the handler functionality by adding a method to get the name of the handler (Hard-coded to "Stdout").

Provides a basic log method implementation to print to the stdout with printf. It adds the support for setting the log level through the ACS_LOG_STDOUT variable.

LogSvcHandler

Extends the handler functionality by adding a method to get the name of the handler (Hard-coded to "acsLogSvc").

The LogSvcHandler is the first part of the logging system to relate with Boost Log. It offers two convenience functions:

- ace2acsPriority/acs2acePriority: To convert between ACE and ACS log levels (priority)

Overrides the setLevels implementation by considering local/remote levels and the log level type. It defines the following log level types:

- DYNAMIC_LOG_LEVEL (1)
- CDB_REFRESH_LOG_LEVEL (2)
- ENV_LOG_LEVEL (3)
- CDB_LOG_LEVEL (4)
- DEFAULT_LOG_LEVEL (5)
- NOT_DEFINED_LOG_LEVEL (6)

Using these log level types, it sets the log levels if the handler log level type is greater or equal than the given type.

It adds the support for setting the local log level through the ACS_LOG_STDOUT variable and to the remote log level through ACS_LOG_CENTRAL variable.

It makes an implementation of the log method based on the LoggingProxy and the boost::sinks::<FrontEnd> classes to interact with the local (stdout) and remote (logging service) endpoints.

Boost Log sink frontend

The sink frontend is part of the Boost Log implementation. It's purpose is to delegate a log call to one or more sink backend instances. The LoggingProxy is a sink backend extension.

The main purpose of the sink frontend class is to allow formatted error messages to be printed in a thread-safe (if configured) manner to various locations, such as stderr, cerr, a distributed logger, etc. while at the same time setting a priority (log level) to be considered for the logging to take place.

LoggingProxy

The LoggingProxy class is an extension of the Boost Log sink backend class, and is one of the central parts of C++ ACS Logging System.

LogLevelDefinition

A class to encapsulate the different log levels accepted by ACS while offering convenient method to create an instance by value or by name, and also a conversion from ACE priorities to ACS log level.

LogThrottle

A utility class to help throttle the publication of log messages. It informs how many log messages can still be sent during the current time interval (Default 1 second). At the end of each interval the log counter is reset to 0.

LogThrottleAlarm

A helper base class for sending alarms associated with not being able to send any more logs in an interval. This class is extended in a specific class to be used by the maciContainer. The reference is passed to the LoggingProxy instance during initialization of the container.

CentralLogger (AcsLogService)

In the LoggingProxy definition this is actually a CORBA Object reference to an object of type AcsLogService. The AcsLogService interface is an extension of the DsLogAdmin::BasicLog interface, which adds two methods:

- writeRecords: Allows sending a sequence of log messages to the central logger service.
- getStatistics: Retrieves logging statistics from the central logger service.

The AcsLogService receives the sequence of records and distributes them through the Logging notification channel.

Boost Log sink backend

Is the back end processing class from Boost Log. It can be derived to execute different tasks. For instance in ACS the CacheLogger extends the sink backend class. These sub-classes are meant to execute the actual work of the logging system.

CacheLogger

It inherits from sink backend and gives a basic interface for three of its subclasses (LocalFileLogger, LocalSyslogLogger and RemoteSyslogLogger). It includes the following methods:

- open: It depends on the specific backend, but it can open a file, a network connection, etc.
- reset: It depends on the specific backend, but it could reset a network connection, etc.
- close: It depends on the specific backend, but it can close a file, a network connection, etc.
- log: The specific implementation to log to the desired backend.
- getIdentification: A backend indentifier name ("Local file", "Local syslog", "Remote syslog", etc.).
- getDestination: The identifier of the destination depends on the specific backend, but it could be a filename, a single value (i.e. "syslog") or a socket, etc.

LocalFileLogger

A specific CacheLogger extension used as a backend to write to a local file.

LocalSyslogLogger

A specific CacheLogger extension used as a backend to write to a local syslog using the syslog library.

RemoteSyslogLogger

A specific CacheLogger extension used as a backend to write to a remote syslog using a socket connection.

The module has two libraries and to executables:

- libbaselogging.so
- liblogging.so
- loggingService
- loggingClient

Library: baselogging

Basic set of logging translation units that are required by some libraries and executable files.

Library: logging

A more extended set of logging translation units that are required by more specific libraries and executable files.

Executable: loggingService

The loggingService is usually referred as the Central Logger. It is a CORBA servant in charge of receiving the logging records from several containers, components, clients and applications, to then distribute them through a notification channel to logging consumers.

Executable: loggingClient

The loggingClient is a service that is able to consume logging or archiving events from the notification channel, and logs them either to syslog, a file, or to stdout.

The module has only one executable

- acsLogSvc

Executable: acsLogSvc

The acsLogSvc is a helper service used by the Python ACS logging system and some other pieces of software to proxy their logging calls to the Central Logger. It is a CORBA servant that receives single log calls and uses the C++ logging system to merge them into the usual flow for logging messages, making sure they reach the Central Logger and are distributed through the logging notification channel.