Document Title	Specification of Log and Trace
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	853

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R20-11

Document Change History

Date	Release	Changed by	Description
2020-11-30	R20-11	AUTOSAR Release Management	• Introduced `Non-modeled messages` and `Modeled messages` to Chapter 7.3 Log Messages • Introduced `Logger::WithLevel()` API, to log messages and pass the LogLevel as an API parameter • Refactoring and editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Removed Class LogManager. Moved `remoteClientState()` to Chapter 8.2 Function definitions (logging.h) • Added Functional Cluster shutdown behavior. Added Funtional Cluster initialization via `ara::core::Initialize()` • Removed TSYNC related spec items from Chapter 7.4 • Refactoring and editorial changes • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Changed APIs (Logstream, Logmanager, Logging) • Refactoring and editorial changes
2018-10-31	18-10	AUTOSAR Release Management	• Changed initialization APIs • Improved references • Log file definition

2018-03-29	18-03	AUTOSAR Release Management	• Refactoring and editorial changes • Log and Trace extensions added
2017-10-27	17-10	AUTOSAR Release Management	No content changes
2017-03-31	17-03	AUTOSAR Release Management	Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

This specification specifies the functionality of the AUTOSAR Adaptive Platform Log and Trace.

The Log and Trace provides interfaces for Adaptive Applications to forward logging information onto the communication bus, the console, or to the file system. Each of the provided logging information has its own severity level. For each severity level, a separate method is provided to be used by applications or Adaptive Platform Services, e.g. ara::com. In addition, utility methods are provided to convert decimal values into the hexadecimal numeral system, or into the binary numeral system.

To pack the provided logging information into a standardized delivery and presentation format, a protocol is needed. For this purpose, the LT protocol can be used, which is standardized within the AUTOSAR consortium.

The LT protocol can add additional information to the provided logging information. This information can be used by a Logging client to relate, sort or filter the received logging frames.

Detailed information regarding the use cases and the LT protocol itself are provided by the PRS Log and Trace protocol specification. For more information regarding the LT protocol refer to [1].

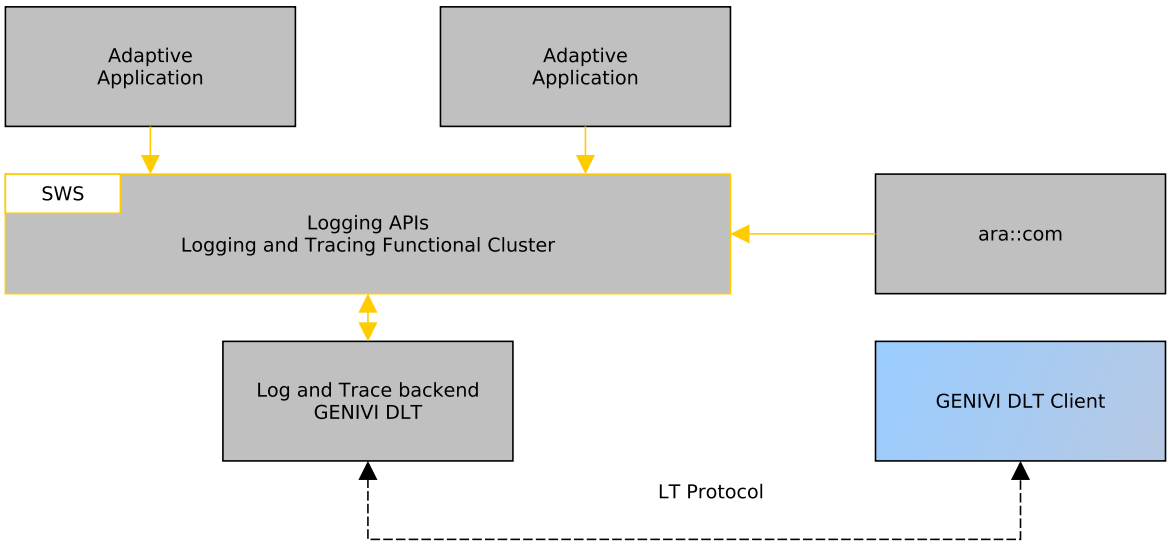

Figure 1.1: Architecture overview

Furthermore, this document introduces additional specification extensions for the AUTOSAR Adaptive Platform Log and Trace.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Log and Trace module that are not included in the [2, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
Log and Trace	The official `Functional Cluster` name that manages the logging
L&T	Acronym for `Log and Trace`
LT protocol	Original name of the protocol itself (Log and Trace), specified in the PRS document [1]
Logging API	The main logging interface towards user applications as a library
Logging back-end	Implementation of the `LT protocol`, e.g. `DLT`
Logging Client	An external tool which can remotely interact with the `Logging framework`
Logging framework	Implementation of the software solution used for logging purposes
Logging instance	The class that enables the logging functionality and handles a single logging context
Log message	Log message, including message header(s)
Log severity level	Meta information about the severity of a passed logging information
DLT	Diagnostics Log and Trace - a GENIVI Log and Trace daemon implementation of the `LT protocol`
Application process	An executable instance (process) that is running on a `Machine`

The following technical terms used throughout this document are defined in the official [2] AUTOSAR Glossary or [3] TPS Manifest Specification – they are repeated here for tracing purposes.

Term	Description
Adaptive Application	see [2] AUTOSAR Glossary
Application	see [2] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [2] AUTOSAR Glossary
Adaptive Platform Foundation	see [2] AUTOSAR Glossary
Manifest	see [2] AUTOSAR Glossary
Executable	see [2] AUTOSAR Glossary
Functional Cluster	see [2] AUTOSAR Glossary
Adaptive Platform Service	see [2] AUTOSAR Glossary
Machine	see [2] AUTOSAR Glossary
Service	see [2] AUTOSAR Glossary
Service Interface	see [2] AUTOSAR Glossary
Service Discovery	see [2] AUTOSAR Glossary

Table 2.1: Glossary-defined Technical Terms

3 Input documents & related standards and norms

3.1 Input documents

[1] Log and Trace Protocol Specification
AUTOSAR_PRS_LogAndTraceProtocol

[2] Glossary
AUTOSAR_TR_Glossary

[3] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[4] Specification of the Adaptive Core
AUTOSAR_SWS_AdaptiveCore

[5] Requirements on Log and Trace
AUTOSAR_RS_LogAndTrace

[6] Specification of Time Synchronization for Adaptive Platform
AUTOSAR_SWS_TimeSync

3.2 Further applicable specification

AUTOSAR provides a core specification [4, SWS AdaptiveCore] which is also applicable for `Log and Trace`. The chapter "General requirements for all Functional Clusters" of this specification shall be considered as an additional and required specification for implementation of `Log and Trace`.

4 Constraints and assumptions

4.1 Known limitations

The provided Logging framework API is designed to be independent from the underlying Logging back-end implementation and as such doesn't impose limitations.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other Functional Clusters

There are no dependencies to other Functional Clusters.

5.1 Platform dependencies

This specification is part of the AUTOSAR AUTOSAR Adaptive Platform and therefore depends on it.

6 Requirements Tracing

The following table references the requirements specified in RS Log And Trace [5] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_LT_00003]	Applications shall have the possibility to send log or trace messages to the LT module.	[SWS_LOG_00001] [SWS_LOG_00002] [SWS_LOG_00004] [SWS_LOG_00005] [SWS_LOG_00006] [SWS_LOG_00007] [SWS_LOG_00008] [SWS_LOG_00009] [SWS_LOG_00010] [SWS_LOG_00011] [SWS_LOG_00012] [SWS_LOG_00013] [SWS_LOG_00015] [SWS_LOG_00016] [SWS_LOG_00017] [SWS_LOG_00018] [SWS_LOG_00019] [SWS_LOG_00021] [SWS_LOG_00022] [SWS_LOG_00023] [SWS_LOG_00024] [SWS_LOG_00025] [SWS_LOG_00026] [SWS_LOG_00027] [SWS_LOG_00028] [SWS_LOG_00029] [SWS_LOG_00030] [SWS_LOG_00031] [SWS_LOG_00032] [SWS_LOG_00033] [SWS_LOG_00034] [SWS_LOG_00035] [SWS_LOG_00036] [SWS_LOG_00037] [SWS_LOG_00039] [SWS_LOG_00040] [SWS_LOG_00041] [SWS_LOG_00042] [SWS_LOG_00043] [SWS_LOG_00044] [SWS_LOG_00045] [SWS_LOG_00046] [SWS_LOG_00047] [SWS_LOG_00048] [SWS_LOG_00049] [SWS_LOG_00050] [SWS_LOG_00051] [SWS_LOG_00053] [SWS_LOG_00054] [SWS_LOG_00055] [SWS_LOG_00056] [SWS_LOG_00057] [SWS_LOG_00058] [SWS_LOG_00059] [SWS_LOG_00060] [SWS_LOG_00062] [SWS_LOG_00063] [SWS_LOG_00064] [SWS_LOG_00065] [SWS_LOG_00066] [SWS_LOG_00067] [SWS_LOG_00068] [SWS_LOG_00069] [SWS_LOG_00070] [SWS_LOG_00082] [SWS_LOG_00083] [SWS_LOG_00091] [SWS_LOG_00095] [SWS_LOG_00098] [SWS_LOG_00101] [SWS_LOG_00108] [SWS_LOG_00109] [SWS_LOG_00110] [SWS_LOG_00111] [SWS_LOG_00112] [SWS_LOG_00113] [SWS_LOG_00114] [SWS_LOG_00115] [SWS_LOG_00120] [SWS_LOG_00122] [SWS_LOG_00123] [SWS_LOG_00124] [SWS_LOG_00128] [SWS_LOG_00129] [SWS_LOG_00130] [SWS_LOG_00131] [SWS_LOG_00201] [SWS_LOG_00203] [SWS_LOG_00204]

Requirement	Description	Satisfied by
[RS_LT_00017]	Each log and trace message shall contain a timestamp, which will be added to the message during reception of the message in the LT module.	[SWS_LOG_00082] [SWS_LOG_00083] [SWS_LOG_00091]
[RS_LT_00030]	Logging shall be able to monitor and shape the amount of LT log and trace events.	[SWS_LOG_00095]
[RS_LT_00045]	Logging shall enable applications to check the current severity level.	[SWS_LOG_00007]
[RS_LT_00046]	Logging shall provide conversion functions for hexadecimal and binary values.	[SWS_LOG_00015] [SWS_LOG_00016] [SWS_LOG_00017] [SWS_LOG_00120]
[RS_LT_00047]	Logging shall support initialization and registration.	[SWS_LOG_00004]
[RS_LT_00048]	Logging shall enable applications to provide meta information.	[SWS_LOG_00004]
[RS_LT_00049]	Logging shall enable applications to provide Logging Information.	[SWS_LOG_00008] [SWS_LOG_00009] [SWS_LOG_00010] [SWS_LOG_00011] [SWS_LOG_00012] [SWS_LOG_00013] [SWS_LOG_00125] [SWS_LOG_00126] [SWS_LOG_00130]
[RS_LT_00050]	Logging shall support grouping of Logging Information.	[SWS_LOG_00005] [SWS_LOG_00006]
[RS_LT_00052]	Logging shall provide early logging capabilities.	[SWS_LOG_00001]

7 Functional specification

This specification defines the usage of the defined C++ Logging API for the Log and Trace. Adaptive Applications can use these functions to forward Log messages to various sinks, for example the network, a serial bus, the console or the file system.

The following functionalities are provided:

1) Methods for initializing the Logging framework (see 7.3)
2) Utility methods to convert decimal values into hexadecimal or binary values (see 7.4)
3) Automatic timestamping of Log messages (see 7.5)
4) Log and trace network bandwith limitation (see chapter 7.6)

Adaptive Applications and Functional Clusters can startup (see 7.1.1) and shutdown (see 7.1.2) all Functional Clusters with direct ARA interfaces (e.g. the Logging framework), by calling `ara::core::Initialize()` or `ara::core::Deinitialize()`.

7.1 Functional Cluster Lifecyle

7.1.1 Startup

In order to initialize the Logging framework, mandatory information needs to be provided to the Logging framework. These information are extracted from the application execution manifest and the AUTOSAR Meta-Model. The execution manifest parameter `Executable.loggingBehavior` defines if the logging functionality should be initialized. Initialization of the Logging framework (via `ara::core::Initialize`) is mandatory before usage of any `ara::log` API. Failure to do so will result in undefined behavior.

[SWS_LOG_00001] ⌈Log message logged before the Logging framework is able to process them (e.g. daemon communication is not established) shall be queued. The queue size is defined by LogAndTraceInstantiation.queueSize. If this size is exceeded the oldest entries shall be discarded.⌋ *(RS_LT_00003, RS_LT_00052)*

7.1.2 Shutdown

[SWS_LOG_00122]{DRAFT} ⌈When ara::core::Deinitialize() is called, the Logging framework shall make sure, that no new client connections can be established.⌋ *(RS_LT_00003)*

[SWS_LOG_00123]{DRAFT} ⌈When ara::core::Deinitialize() is called, the Logging framework shall take care that all remaining messages in the buffer can be collected, if a client is connected.⌋ *(RS_LT_00003)*

7.2 Necessary Parameters and Initialization

The concept of identifying the user application:
To be able to distinguish the logs of different application instances within a system (e.g. an ECU or even the whole vehicle), every Application process, in that system, has to get a particular ID and a description.

The concept of log contexts:
In order to be able to distinguish the logs from different logical groups within an Application process, for every context within an Application process a particular ID and a description has to be assigned. Every Application process can have an arbitrary amount of contexts, but at least one – the default context.

Machine-specific configuration settings for the Log and Trace functional cluster are collected in LogAndTraceInstantiation. The Application processes using the Logging framework need to supply the following configuration through the application execution manifest:

- Application ID
- Application description
- The default log level, if not set through the manifest a default predefined value is set
- The log mode
- The log file path, in case of a specific log mode that indicates logging to a file

The Application process using the Logging framework creates a Logging instance per context. The context is defined at creation of the Logging instance and the following information should be provided:

- Context ID
- Context description
- The default log level, if not set through the manifest a default predefined value is set

7.2.1 Application ID

The Application ID is an identifier that allows to associate generated logging information with its user application. The Application ID is passed as a string value. Depending on the Logging framework actual implementation, i.e. Logging back-end,

the length of the Application ID might be limited. To be able to unambiguously associate the received logging information to the origin, it is recommended to assign unique Application IDs within one ECU. There is no need for uniqueness of Application IDs across ECUs as the ECU ID will be the differentiator. The system integrator has the overall responsibility to ensure that each Application process instance has a unique Application ID. By having this value defined in the manifest the integrator is able to perform consistency checks. The applicationId in the DltLogChannel identifies the application instance and is put as ApplicationId into the log and trace message.

Note:
The Application IDs are unique IDs per Application process, meaning if the same Application process is started multiple times it shall have an own ID per instance.

7.2.1.1 Application Description

Since the length of the Application ID can be quite short, an additional descriptive text can be provided. This description is passed as a string and the maximum length is implementation dependent. The applicationDescription in the DltLogChannel is an optional setting that allows to describe the applicationId as descriptive text.

7.2.2 Default Log Level

The Log severity level represents the severity of the log messages. Severity levels are defined in chapter 7.3. logTraceDefaultLogLevel in the DltLogChannel defines the initial log reporting level for the application instance.

Each initiated log message is qualified with such a severity level. The default Log severity level is set through the application configuration per Application process. The Log severity level acts as a reporting filter. Only log messages having a higher or the same severity will be processed by the Logging framework, while the others are ignored.

The default Log severity level is the initially configured log reporting level for a certain Application process, though it can be overriden per context.

The Application process wide log reporting level shall be adjustable during runtime. The realization is an implementation detail of the underlying back-end. E.g. remotely via a Logging client for example DLT Viewer. The same applies for the context reporting level.

The design rationale for providing an initial default Log severity level application wide against having per context default Log severity levels is the following:

- It simplifies the API usage. Otherwise the user will have to define a context default Log severity level for each group before using the API.

- The context separation of Log messages is possible during runtime.

7.2.3 Log Mode

Depending on the Logging framework implementation, the passed logging information can be processed in different ways. The destination (the Log message sink) can be the console output, a file on the file system or the communication bus. The system integrator is responsible to populate this information in the machine manifest. A direct API for dynamically changing this value for development purposes is provided. In the AUTOSAR Meta-Model the logTraceLogMode is equivalent to the log mode described here, for more information see [3]. logTraceLogMode in the DltLogChannel of the LogAndTraceInstantiation defines the destination to which the log messages will be forwarded.

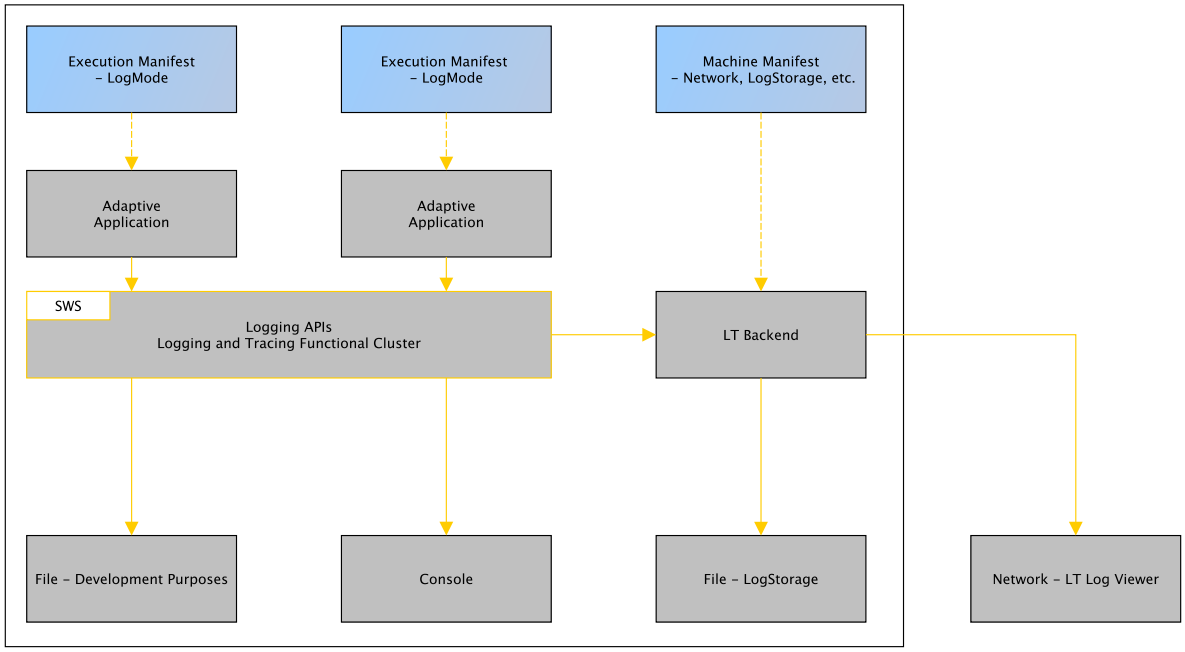

Figure 7.1: Log mode

As shown in the diagram, once the log mode is set to use the Logging back-end the configuration is of that back-end is centralized in the Machine manifest configuration. For example, the Logging back-end can be configured to store the logging information locally and that configuration would be kept in the Machine-specific manifest. Furthermore, the output channel on Ethernet for Log messages is configured with the PlatformModuleEthernetEndpointConfiguration that is aggregated by the LogAndTraceInstantiation via DltLogChannel in the role endpointConfiguration.

7.2.3.1 Log File Path

In case the log mode is set to log to a file, a destination directory path needs to be provided. logTraceFilePath in the DltLogChannel defines the destination file to which the logging information is passed. This option is provided for development, integration and prototyping purposes and is not suitable for production.

7.2.4 Context ID

The Context ID is an identifier that is used to logically group logging information within the scope of an Application process. The Context ID is passed as a string value. Depending on the actual implementation of the Logging back-end, the length of the Context ID might be limited. Context ID is unique in the scope of an Application process and as such the developer is responsible for assigning it and this information is not modeled in the manifest. There is no need for uniqueness of Context IDs across multiple different Application processes as the Application ID will be the differentiator.

Note:
Special attention should be paid to library components. The libraries are meant to be used by Application processes and therefore are running within the Application process' scope. Logging executed from those libraries will end up inside the scope of the parent Application process. In order to distinguish the internal library logs from the Application process logs or from other library logs within same process, each library might need to reserve its own Context IDs system wide – at least when it shall be used by more than one Application process.

7.2.5 Context Description

Since the length of the Context ID can be quite short, an additional descriptive text must be provided. This Context description is passed as a string. The maximum length of the Context description is implementation dependent.

7.2.6 Initialization of the Logging framework

The Application ID and description are used to identify and to associate the provided logging information with the exact process. The log mode and sink information defines where the logging information is routed. Possible destinations are the console, the file system or the communication bus.

From the Application process' perspective, the Logging framework is intialized and a logger instance is created when an Application process decides to register a logging context. These contexts are used to logically cluster logging information.

[SWS_LOG_00002]{DRAFT} ⌈In case of any errors occurring inside the Logging framework or underlying system, it is intended to not bother the Application process and silently discard the function calls. For this purpose, the relevant interfaces neither specify return values nor throw exceptions.⌋*(RS_LT_00003)*

[SWS_LOG_00004]{DRAFT} ⌈The application execution manifest should provide the following information for the Logging framework to be initialized:

- A unique application ID
- An application description
- The default Log severity level
- The log mode
- The directory path (only necessary if LogMode::kFile is given as log mode)

⌋*(RS_LT_00003, RS_LT_00047, RS_LT_00048)*

Note:
Depending on the Logging framework implementation not all of the features might be supported, hence not all of the properties will be used.

[SWS_LOG_00005] ⌈The function CreateLogger() shall create a logger context instance internally inside the Logging framework and return it as reference to the using application. Before a Log message can be processed, at least one logger context shall be available.⌋*(RS_LT_00003, RS_LT_00050)*

Note:
This strong ownership relationship of contexts to the Logging framework ensure correct housekeeping of the involved resources. The design rationale is, once a context is registered against the Logging back-end, its lifetime must be ensured until the end of the Application process.

[SWS_LOG_00006] ⌈By calling CreateLogger(), the following parameters need to be provided:

- The context ID
- The context description
- The Log severity level (as an optional parameter, defaults to LogLevel::kWarn)

⌋*(RS_LT_00003, RS_LT_00050)*

[SWS_LOG_00007] ⌈Application processes should be able to check if a desired Log severity level is configured through the function IsLogEnabled(). This mechanism conserves CPU and memory resources that are used during preparation of

logging information, as this logging information is filtered by the `Logging framework` later on.⌋ *(RS_LT_00003, RS_LT_00045)*

7.3 Log Messages

Log messages can generally be output to different targets. The Log and Trace Functional Cluster supports these logging targets:

- the console
- a file on a local file system
- a network

Most of the discussion in this section assumes that messages are being output to a network, as this use case requires the additional consideration of minimizing network load.

The Log And Trace Functional Cluster offers two principal “classes” of log messages: *Modeled* and *Non-Modeled* messages. Both these support adding one or more “arguments” to a log message. A log message without any arguments serves no purpose and is discarded.

Non-Modeled messages are the traditional way of composing log messages: All arguments of the message are added to an internal message buffer and then eventually serialized for output, either to a console/file, or via network. All parts of the messages will be sent via network. In the DLT protocol, these messages are called “verbose” messages.

Modeled messages are designed to reduce traffic on the network, by omitting certain static (i.e. unchanging) parts of a message from the network. As the name suggests, these parts are instead added to the application ARXML model. In the DLT protocol, these messages are called “non-verbose” messages. A log message viewer application is able to display the full message by combining the static parts from the model with the dynamic parts from the received message.

Non-modeled messages are mainly used during development, as the information required for the modeled messages may not be available at that time. However, non-modeled messages can impose a high load on the network, making modeled messages usually the preferred choice in production systems.

The `ara::log` Functional Cluster supports defining and using both modeled and non-modeled messages in a single application at the same time.

7.3.1 Non-modeled messages

The `ara::log` Functional Cluster defines a “Builder”-pattern inspired set of APIs for constructing non-modeled messages. The `ara::log::Logger::WithLevel` member function is used for creating a `ara::log::LogStream` object which is then subsequently filled with message content (i.e. message arguments). Alternatively to `ara::log::Logger::WithLevel`, there are also separate member functions for creating a `ara::log::LogStream` object, one per supported log level.

Arguments are added to a verbose message by calling an appropriate `operator<<` overload for the desired argument:

```
logger.WithLevel(LogLevel::kInfo) << "text" << 4.2;
```

The `ara::log` Functional Cluster defines such `operator<<` overloads for all C++ arithmetic types, for bool, for string types, and for a number of `ara::core` types. Application-defined data types can be logged as well, by providing suitable `operator<<` overloads for them.

As the application model allows "annotating" arguments with attributes, the `ara::log` API for non-modeled messages also supports this. Arguments of certain types can be annotated with a "name" and possibly also a "unit" attribute. For instance:

```
1 logger.WithLevel(LogLevel::kInfo)
2     << Arg("text", "identifier")
3     << Arg(4.2, "velocity", "m/s");
```

The string argument "text" is annotated with a "name" attribute called "identifier". The double argument 4.2 is annotated with a "name" attribute "velocity" and a "unit" of "m/s". These attributes can only be set for some of the built-in types that the `ara::log` API supports, i.e. all arithmetic types, bool, strings, and raw data blobs.

Non-modeled messages can also contain information about the location of the log message call in source code. For this purpose, the member function `ara::log::LogStream::WithLocation` is called with the filename and line number of the call site. These should usually come from the compiler-defined `__FILE__` and `__LINE__` symbols:

```
1 logger
2     .WithLevel(LogLevel::kInfo)
3     .WithLocation(__FILE__, __LINE__) << ...;
```

These are easiest set via a macro-based frontend for `ara::log`, but no such macro has yet been defined in the Adaptive Platform.

7.3.2 Modeled messages

7.3.2.1 API principles

The `ara::log` Functional Cluster defines a single member function `ara::log::Logger::Log` for sending modeled messages. Unlike the non-modeled message APIs, it represents a single-call interface, i.e. a single function call passes all arguments to the `Logger` instance and performs all necessary actions to generate and send the message.

This has the advantage that the runtime cost for a modeled message that is eventually not being output (because the message's log level does not reach the configured log level threshold) can be made very small: after parameter passing and function call, a single `if` clause checks the log level threshold and immediately returns if the threshold

is not reached. This contrasts with the non-modeled message APIs, where multiple function calls are performed for constructing a message object, even if that is then eventually discarded.

7.3.2.2 Log message model

All modeled messages are defined as DltMessages, which are aggregated by a DltMessageCollectionSet that is referenced by the System. Each DltMessage contains a messageId, which needs to be unique within an ECU, and the messageTypeInfo denoting the log level, and optionally the messageSourceFile and messageLineNumber. The DltMessage aggregates an ordered list of DltArguments, which in turn refer to an SwDataDefProps in the role networkRepresentation. The name of a log message argument is taken from the shortName of the DltArgument, while the type and unit are taken from the SwDataDefProps.

At design time, the DltMessages are allocated to a DltLogChannelDesign, which is mapped to the application using a DltLogChannelDesignToProcessDesignMapping to a ProcessDesign, which in turn refers to the Executable.

At deployment time, the DltLogChannelDesign is referenced by a DltLogChannel, which inherits the DltMessages from the design phase. The DltLogChannel is mapped to the application using a DltLogChannelToProcessMapping referencing a Process of the Executable. All DltLogChannels are aggregated by the LogAndTraceInstantiation of the ECU, that carries the dltEcuId and queueSize, and with sessionIdSupport the information whether session IDs are used. A DltLogChannel contains the applicationId and contextId and the corresponding applicationDescription and contextDescription. It may refer to a ServiceInstanceToPortPrototypeMapping, in which case the log messages from this port will use the contextId of the DltLogChannel. A DltLogChannel also contains the sessionId, and with nonVerboseMode the information whether modeled messages will be sent as verbose messages as if they were non-modeled messages. And finally, a DltLogChannel contains a logTraceDefaultLogLevel giving the initial threshold for log messages, logTraceLogMode configuring the destination of log messages, and the logTraceFilePath when the destination is file.

7.3.2.3 Usage

The C++ API assumes the existence of a tool that scans source code for modeled log message call sites and generates the expected symbols with unique IDs on-demand.

The framework is required to scan all source code for invocations of the ara::log::Logger::Log member function, and generate a symbol that matches the first argument of that member function call.

For instance, if the ARXML representation of the manifest contains the following:

```
<DLT-MESSAGE-COLLECTION-SET>
```

```
  <SHORT-NAME>DltMessages</SHORT-NAME>
  <DLT-MESSAGES>
    <DLT-MESSAGE>
      <SHORT-NAME>SpeedMsg</SHORT-NAME>
      ...
    </DLT-MESSAGE>
  </DLT-MESSAGES>
</DLT-MESSAGE-COLLECTION-SET>
```

and the source code contains this code sequence:

```
Logger& logger = ...
logger.Log(SpeedMsg, 4.2);
```

then the framework will define a global constexpr variable called `SpeedMsg` of an implementation-defined type. This variable contains knowledge about the message's modeled aspects, such as parameter types and log level, allowing the `ara::log` implementation to verify that the number of types of parameters given to `ara::log::Logger::Log` matches the model of the particular message.

The message variable definitions will be made available via `ara/log/logger.h`.

Store LogStream objects in a variable:
It is also possible to use the `Logging API` in an alternative way by storing a `ara::log::LogStream` object locally in some named variable. The difference to the temporary object is that it won't go out of scope already at the end of the statement, but stays valid and re-usable as long as the variable exists. Hence, it can be fed with data distributed over multiple lines of code. To get the message buffer processed by the `Logging framework`, the `ara::log::LogStream::Flush` method needs to be called, otherwise the buffer will be processed when the object dies, i.e. when the variable goes out of scope, at the end of the function block.

Performance remark:
Due to the fact that a `ara::log::LogStream` is no longer created per message but rather could be re-used for multiple messages, the costs for this object creation is paid only once – per log level. How much this really influences the actual performance depends on the `Logging framework` implementation. However the main goal of this alternative usage of the `Logging API` is to get the multi-line builder functionality.

Note:
It is highly advised NOT to hold global `ara::log::LogStream` objects in multi-threaded `Application`s, because then concurrent access protection will no longer be covered by the `Logging API`.

Usage examples:

```
Logger& ctx0 = CreateLogger("CTX0", "Context Description CTX0");
ctx0.LogInfo() << "Some log information" << 123;
```

```
// Locally stored LogStream object will process the arguments
// until either Flush() is called or it goes out of scope from
// the block is was created
Logger& ctx1 = CreateLogger("CTX1", "Context Description CTX1");
```

```
LogStream localLogInfo = ctx1.LogInfo();
localLogInfo << "Some log information" << 123;
localLogInfo << "Some other information";
localLogInfo.Flush();
localLogInfo << "a new message..." << 456;
```

Exception safety: All `Log*()` interfaces are designed to guarantee no-throw behavior. This applies for the whole `Logging API`.

New line: Because of convenience purposes the `Logging framework` automatically appends a newline to the `Log message`.

Multiple payload arguments: When one message consists of more than one payload argument, payload arguments a separated by single whitespaces for console output.

[SWS_LOG_00008]{DRAFT} ⌈To initiate a `Log message` with the Log level `Fatal`, the API `ara::log::Logger::LogFatal` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00009]{DRAFT} ⌈To initiate a `Log message` with the Log level `Error`, the API `ara::log::Logger::LogError` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00010]{DRAFT} ⌈To initiate a `Log message` with the Log level `Warning`, the API `ara::log::Logger::LogWarn` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00011]{DRAFT} ⌈To initiate a `Log message` with the Log level `Info`, the API `ara::log::Logger::LogInfo` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00012]{DRAFT} ⌈To initiate a `Log message` with the Log level `Debug`, the API `ara::log::Logger::LogDebug` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00013]{DRAFT} ⌈To initiate a `Log message` with the Log level `Verbose`, the API `ara::log::Logger::LogVerbose` shall be called. This API returns a `ara::log::LogStream` object that has to be used by passing arguments via the insert stream operator<<.⌋*(RS_LT_00003, RS_LT_00049)*

[SWS_LOG_00130]{DRAFT} ⌈To write a Log message with a programmatically determined log level, the API Logger::WithLevel(LogLevel logLevel) shall be called.⌋*(RS_LT_00003, RS_LT_00049)*

7.4 Conversion Functions

Sometimes it makes sense to represent integer numbers in hexadecimal or binary format instead of decimal format.
For this purpose, the following functions are defined to convert provided decimal numbers into the hexadecimal or binary system.

[SWS_LOG_00120]{DRAFT} ⌈Dedicated conversion functions are provided for conversion of positive decimal numbers into a string with hexadecimal or binary representation.⌋*(RS_LT_00003, RS_LT_00046)*

[SWS_LOG_00015]{DRAFT} ⌈Dedicated conversion functions are provided for conversion of decimal numbers into a string with hexadecimal or binary representation, where the most significant bit shall be set to '1' for negative numbers.⌋*(RS_LT_00003, RS_LT_00046)*

[SWS_LOG_00016]{DRAFT} ⌈Function `HexFormat()` shall provide functionality to convert an integer decimal number into a string with hexadecimal representation.⌋*(RS_LT_00003, RS_LT_00046)*

[SWS_LOG_00017]{DRAFT} ⌈Function `BinFormat()` shall provide functionality to convert an integer decimal number into a string with binary representation.⌋*(RS_LT_00003, RS_LT_00046)*

7.5 Log and Trace Timestamp

The Log and Trace information is transmitted by means of the LT protocol which is bus agnostic.
This protocol offers the possibility to include a timestamp in each sent message, as long as such messages are sent with an extended header (refer to [5] for more information).

The synchronized time base is supplied by the Time Synchronization Functional Cluster. The now() method is used by the Adaptive Applications in order to retrieve the current time from the TS (refer to [6] for more information).

According to the requirement [TPS_MANI_03162], the reference time base is derived from the machine manifest timeBaseResource.

[SWS_LOG_00082] ⌈Log and Trace should have accesss to a synchronized time base. The attribute timeBaseResource in LogAndTraceInstantiation shall be used to identify the time base.⌋*(RS_LT_00003, RS_LT_00017)*

[SWS_LOG_00083] ⌈In case there is no time base resource referenced by the Log and Trace module in the manifest configuration, no timestamp information shall be transmitted.⌋*(RS_LT_00003, RS_LT_00017)*

[SWS_LOG_00091]{DRAFT} ⌈When the CreateLogger() function is called, Log and Trace shall send a message, "local time base used" in case the used time base is a local time base or "global time base used" in case the used time base is a globally synchronized time base.

⌋*(RS_LT_00003, RS_LT_00017)*

7.6 Log and Trace data loss prevention

[SWS_LOG_00095]{DRAFT} ⌈When `Log and Trace` receives simultaneously a high load of trace information generated by multiple `Adaptive Application`s, it shall buffer this data internally to prevent the data loss during its continuous transmission.⌋ *(RS_LT_00003, RS_LT_00030)*

8 API specification

8.1 API Common Data Types

8.1.1 LogLevel

[SWS_LOG_00018]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	LogLevel	
Scope:	namespace ara::log	
Underlying type:	uint8_t	
Syntax:	`enum class LogLevel : uint8_t {...};`	
Values:	kOff= 0x00	No logging.
	kFatal= 0x01	Fatal error, not recoverable.
	kError= 0x02	Error with impact to correct functionality.
	kWarn= 0x03	Warning if correct behavior cannot be ensured.
	kInfo= 0x04	Informational, providing high level understanding.
	kDebug= 0x05	Detailed information for programmers.
	kVerbose= 0x06	Extra-verbose debug messages (highest grade of information)
Header file:	#include "ara/log/common.h"	
Description:	List of possible severity levels .	

⌋(*RS_LT_00003*)

8.1.2 LogMode

[SWS_LOG_00019]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	LogMode	
Scope:	namespace ara::log	
Underlying type:	uint8_t	
Syntax:	`enum class LogMode : uint8_t {...};`	
Values:	kRemote= 0x01	Sent remotely.
	kFile= 0x02	Save to file.
	kConsole= 0x04	Forward to console.
Header file:	#include "ara/log/common.h"	
Description:	Log mode. Flags, used to configure the sink for log messages.	
Notes:	In order to combine flags, at least the OR and AND operators needs to be provided for this type.	

⌋(*RS_LT_00003*)

8.1.3 LogHex8

[SWS_LOG_00108]{DRAFT} ⌈

Kind:	struct
Symbol:	LogHex8
Scope:	namespace ara::log
Syntax:	`struct LogHex8 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 8 bit hexadecimal value data type . Helper struct that is utilized as custom type. Holds an integer value that will be logged with a special format.

⌋(*RS_LT_00003*)

8.1.4 LogHex16

[SWS_LOG_00109]{DRAFT} ⌈

Kind:	struct
Symbol:	LogHex16
Scope:	namespace ara::log
Syntax:	`struct LogHex16 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 16 bit hexadecimal value data type .

⌋(*RS_LT_00003*)

8.1.5 LogHex32

[SWS_LOG_00110]{DRAFT} ⌈

Kind:	struct
Symbol:	LogHex32
Scope:	namespace ara::log
Syntax:	`struct LogHex32 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 32 bit hexadecimal value data type .

⌋(*RS_LT_00003*)

8.1.6 LogHex64

[SWS_LOG_00111]{DRAFT} ⌈

Kind:	struct
Symbol:	LogHex64
Scope:	namespace ara::log
Syntax:	`struct LogHex64 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 64 bit hexadecimal value data type .

⌋(*RS_LT_00003*)

8.1.7 LogBin8

[SWS_LOG_00112]{DRAFT} ⌈

Kind:	struct
Symbol:	LogBin8
Scope:	namespace ara::log
Syntax:	`struct LogBin8 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 8 bit binary data type .

⌋(*RS_LT_00003*)

8.1.8 LogBin16

[SWS_LOG_00113]{DRAFT} ⌈

Kind:	struct
Symbol:	LogBin16
Scope:	namespace ara::log
Syntax:	`struct LogBin16 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 16 bit binary data type .

⌋(*RS_LT_00003*)

8.1.9 LogBin32

[SWS_LOG_00114]{DRAFT} ⌈

Kind:	struct
Symbol:	LogBin32
Scope:	namespace ara::log
Syntax:	`struct LogBin32 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 32 bit binary data type .

⌋(*RS_LT_00003*)

8.1.10 LogBin64

[SWS_LOG_00115]{DRAFT} ⌈

Kind:	struct
Symbol:	LogBin64
Scope:	namespace ara::log
Syntax:	`struct LogBin64 {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	Represents a 64 bit binary data type .

⌋(*RS_LT_00003*)

8.1.11 ClientState

[SWS_LOG_00098]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ClientState	
Scope:	namespace ara::log	
Underlying type:	int8_t	
Syntax:	`enum class ClientState : int8_t {...};`	
Values:	kUnknown= -1	–
	kNotConnected	–
	kConnected	–
Header file:	#include "ara/log/common.h"	
Description:	Client state representing the connection state of an external client. .	

⌋(*RS_LT_00003*)

8.2 Function definitions

8.2.1 CreateLogger

[SWS_LOG_00021]{DRAFT} ⌈

Kind:	function	
Symbol:	CreateLogger(ara::core::StringView ctxId, ara::core::StringView ctxDescription, LogLevel ctxDef LogLevel=LogLevel::kWarn)	
Scope:	namespace ara::log	
Syntax:	`Logger& CreateLogger (ara::core::StringView ctxId, ara::core::String View ctxDescription, LogLevel ctxDefLogLevel=LogLevel::kWarn) noexcept;`	
Parameters (in):	ctxId	The context ID.
	ctxDescription	The description of the provided context ID.
	ctxDefLogLevel	The default log level, set to Warning severity if not explicitly specified.
Return value:	Logger &	Reference to the internal managed instance of a Logger object. Ownership stays within the Logging framework
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Creates a Logger object, holding the context which is registered in the Logging framework.	

⌋(*RS_LT_00003*)

8.2.2 HexFormat (uint8)

[SWS_LOG_00022]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(uint8_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex8 HexFormat (uint8_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex8	LogHex8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint8 into a hexadecimal value. Negatives are represented in 2's complement. The number of represented digits depends on the overloaded parameter type length.	
Notes:	Logs decimal numbers in hexadecimal format.	

⌋(*RS_LT_00003*)

8.2.3 HexFormat (int8)

[SWS_LOG_00023]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(int8_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex8 HexFormat (int8_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex8	LogHex8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int8 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format. Negatives are represented in 2's complement.	

⌋*(RS_LT_00003)*

8.2.4 HexFormat (uint16)

[SWS_LOG_00024]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(uint16_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex16 HexFormat (uint16_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex16	LogHex16 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint16 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format.	

⌋*(RS_LT_00003)*

8.2.5 HexFormat (int16)

[SWS_LOG_00025]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(int16_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex16 HexFormat (int16_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex16	LogHex16 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int16 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format. Negatives are represented in 2's complement.	

⌋(*RS_LT_00003*)

8.2.6 HexFormat (uint32)

[SWS_LOG_00026]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(uint32_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex32 HexFormat (uint32_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex32	LogHex32 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint32 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format.	

⌋(*RS_LT_00003*)

8.2.7 HexFormat (int32)

[SWS_LOG_00027]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(int32_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex32 HexFormat (int32_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex32	LogHex32 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int32 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format. Negatives are represented in 2's complement.	

⌋(*RS_LT_00003*)

8.2.8 HexFormat (uint64)

[SWS_LOG_00028]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(uint64_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex64 HexFormat (uint64_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex64	LogHex64 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint64 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format.	

⌋(*RS_LT_00003*)

8.2.9 HexFormat (int64)

[SWS_LOG_00029]{DRAFT} ⌈

Kind:	function	
Symbol:	HexFormat(int64_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogHex64 HexFormat (int64_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into hexadecimal number system.
Return value:	LogHex64	LogHex64 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int64 into a hexadecimal value.	
Notes:	Logs decimal numbers in hexadecimal format. Negatives are represented in 2's complement.	

⌋(*RS_LT_00003*)

8.2.10 BinFormat (uint8)

[SWS_LOG_00030]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(uint8_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin8 BinFormat (uint8_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin8	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint8 into a binary value.	
Notes:	Logs decimal numbers in binary format.	

⌋(*RS_LT_00003*)

8.2.11 BinFormat (int8)

[SWS_LOG_00031]{DRAFT} ⌈

Kind:	function
Symbol:	BinFormat(int8_t value)
Scope:	namespace ara::log

▽

△

Syntax:	`constexpr LogBin8 BinFormat (int8_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin8	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int8 into a binary value.	
Notes:	Logs decimal numbers in binary format. Negatives are represented in 2's complement.	

⌋(*RS_LT_00003*)

8.2.12 BinFormat (uint16)

[SWS_LOG_00032]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(uint16_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin16 BinFormat (uint16_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin16	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint16 into a binary value.	
Notes:	Logs decimal numbers in binary format.	

⌋(*RS_LT_00003*)

8.2.13 BinFormat (int16)

[SWS_LOG_00033]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(int16_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin16 BinFormat (int16_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin16	LogBin8 type that has a built-in stream handler.

▽

△

Exception Safety:	noexcept
Thread Safety:	reentrant
Header file:	#include "ara/log/logger.h"
Description:	Conversion of a int16 into a binary value.
Notes:	Logs decimal numbers in binary format. Negatives are represented in 2's complement.

⌋(*RS_LT_00003*)

8.2.14 BinFormat (uint32)

[SWS_LOG_00034]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(uint32_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin32 BinFormat (uint32_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin32	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint32 into a binary value.	
Notes:	Logs decimal numbers in binary format.	

⌋(*RS_LT_00003*)

8.2.15 BinFormat (int32)

[SWS_LOG_00035]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(int32_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin32 BinFormat (int32_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin32	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	

▽

△

Description:	Conversion of a int32 into a binary value.
Notes:	Logs decimal numbers in binary format. Negatives are represented in 2's complement.

⌋(*RS_LT_00003*)

8.2.16 BinFormat (uint64)

[SWS_LOG_00036]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(uint64_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin64 BinFormat (uint64_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin64	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a uint64 into a binary value.	
Notes:	Logs decimal numbers in binary format.	

⌋(*RS_LT_00003*)

8.2.17 BinFormat (int64)

[SWS_LOG_00037]{DRAFT} ⌈

Kind:	function	
Symbol:	BinFormat(int64_t value)	
Scope:	namespace ara::log	
Syntax:	`constexpr LogBin64 BinFormat (int64_t value) noexcept;`	
Parameters (in):	value	Decimal number to be converted into a binary value.
Return value:	LogBin64	LogBin8 type that has a built-in stream handler.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Conversion of a int64 into a binary value.	
Notes:	Logs decimal numbers in binary format. Negatives are represented in 2's complement.	

⌋(*RS_LT_00003*)

8.2.18 remoteClientState

[SWS_LOG_00101]{DRAFT} ⌈

Kind:	function	
Symbol:	remoteClientState()	
Scope:	namespace ara::log	
Syntax:	`ClientState remoteClientState () noexcept;`	
Return value:	ClientState	The current client state.
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/logger.h"	
Description:	Fetches the connection state from the DLT back-end of a possibly available remote client.	

⌋(*RS_LT_00003*)

8.2.19 Wrapper object creator

[SWS_LOG_00201]{DRAFT} ⌈

Kind:	function	
Symbol:	Arg(T &&arg, const char *name=nullptr, const char *unit=nullptr)	
Scope:	namespace ara::log	
Syntax:	`template <typename T>` `Argument<T> Arg (T &&arg, const char *name=nullptr, const char *unit=nullptr) noexcept;`	
Parameters (in):	arg	an argument payload object
	name	an optional "name" attribute for arg
	unit	an optional "unit" attribute for arg
Return value:	Argument< T >	a wrapper object holding the supplied arguments
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Create a wrapper object for the given arguments. Calling this function shall be ill-formed if any of these conditions are met: T is not an arithmetic type and not "bool" and not convertible to "ara::core::StringView" and not convertible to "ara::core::Span<const ara::core::Byte>" T is convertible to "ara::core::StringView" or convertible to "ara::core::Span<const ara::core::Byte>" or "bool", and "unit" is not "nullptr"	

⌋(*RS_LT_00003*)

8.2.20 Logger of an argument with attributes

[SWS_LOG_00203]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const Argument< T > &arg)	
Scope:	namespace ara::log	
Syntax:	`template <typename T>` `LogStream& operator<< (const Argument< T > &arg) noexcept;`	
Template param:	T	the argument payload type
Parameters (in):	arg	the argument wrapper object
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Log an argument with attributes. When output to the console, the value and all its attributes shall be shown as a single argument.	

⌋(*RS_LT_00003*)

8.2.21 Logger of modeled message

[SWS_LOG_00204]{DRAFT} ⌈

Kind:	function	
Symbol:	Log(const MsgId &id, const Params &... args)	
Scope:	class ara::log::Logger	
Syntax:	`template <typename MsgId, typename... Params>` `void Log (const MsgId &id, const Params &... args) noexcept;`	
Template param:	MsgId	the type of the id parameter
	Args	the types of the args parameters
Parameters (in):	id	an implementation-defined type identifying the message object
	args	the arguments to add to the message
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Log a modeled message. If this function is called with an argument list that does not match the modeled message, the program is ill-formed.	

⌋(*RS_LT_00003*)

8.3 Class definitions

8.3.1 Class LogStream

The class `ara::log::LogStream` represents a `Log message`, allowing stream operators to be used for appending data.

[ID_NOT_DEFINED] ⌈

Kind:	class
Symbol:	LogStream
Scope:	namespace ara::log
Syntax:	`class LogStream final {...};`
Header file:	#include "ara/log/log_stream.h"
Description:	–

⌋()

Note:
Normally `Application processes` would not use this class directly. Instead one of the log methods provided in the main `Logging API` shall be used. Those methods automatically setup a temporary object of this class with the given log severity level. The only reason to use this class directly is, if the user wants to hold a `ara::log::LogStream` object longer than the default one-statement scope. This is useful in order to create log messages that are distributed over multiple code lines. See the `ara::log::LogStream::Flush` method for further information. Once this temporary object gets out of scope, its destructor takes care that the message buffer is ready to be processed by the Logging framework.

8.3.1.1 Extending the Logging API to understand custom types

The `ara::log::LogStream` class supports natively the formats stated in chapter 8.2, it can be easily extended for other derived types by providing a stream operator that makes use of already supported types.

Example:

```
struct MyCustomType {
  int8_t foo;
  ara::core::String bar;
};

LogStream& operator<<(LogStream& out, const MyCustomType& value) {
  return (out << value.foo << value.bar);
}

// Producing the output "42 the answer is."
Logger& ctx0 = CreateLogger("CTX0", "Context Description CTX0");
```

```
12 ctx0.LogDebug () << MyCustomType{42, " the answer is."};
```

8.3.1.2 LogStream::Flush

[SWS_LOG_00039]{DRAFT} ⌈

Kind:	function
Symbol:	Flush()
Scope:	class ara::log::LogStream
Syntax:	`void Flush () noexcept;`
Return value:	None
Exception Safety:	noexcept
Thread Safety:	reentrant
Header file:	#include "ara/log/log_stream.h"
Description:	Sends out the current log buffer and initiates a new message stream.

⌋*(RS_LT_00003)*

Note:
Calling `ara::log::LogStream::Flush` is only necessary if the `ara::log::LogStream` object is going to be re-used within the same scope. Otherwise, if the object goes out of scope (e.g. end of function block) then the flushing operation will be done internally by the destructor. It is important to note that the `ara::log::LogStream::Flush` command does not empty the buffer, but it forwards the buffer's current contents to the `Logging framework`.

8.3.1.3 Built-in operators for natively supported types

[SWS_LOG_00040]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(bool value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (bool value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Appends given value to the internal message buffer.	

⌋*(RS_LT_00003)*

[SWS_LOG_00041]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(uint8_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (uint8_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int 8 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00042]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(uint16_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (uint16_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int 16 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00043]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(uint32_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (uint32_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int 32 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00044]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(uint64_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (uint64_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int 64 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00045]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(int8_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (int8_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes signed int 8 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00046]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(int16_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (int16_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes signed int 16 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00047]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(int32_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (int32_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes signed int 32 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00048]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(int64_t value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (int64_t value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes signed int 64 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00049]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(float value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (float value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes float 32 bit parameter into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00050]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(double value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (double value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes float 64 bit parameter into message.	

⌋(*RS_LT_00003*)

8.3.1.4 Built-in operators for conversion types

[SWS_LOG_00053]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogHex8 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogHex8 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as hexadecimal 8 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00054]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogHex16 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogHex16 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	

▽

△

Header file:	#include "ara/log/log_stream.h"
Description:	Writes unsigned int parameter into message, formatted as hexadecimal 16 digits.

⌋(*RS_LT_00003*)

[SWS_LOG_00055]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogHex32 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogHex32 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as hexadecimal 32 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00056]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogHex64 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogHex64 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as hexadecimal 64 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00057]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogBin8 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogBin8 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.

▽

△

Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as binary 8 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00058]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogBin16 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogBin16 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as binary 16 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00059]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogBin32 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogBin32 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as binary 32 digits.	

⌋(*RS_LT_00003*)

[SWS_LOG_00060]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const LogBin64 &value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const LogBin64 &value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes unsigned int parameter into message, formatted as binary 64 digits.	

⌋(*RS_LT_00003*)

8.3.1.5 Built-in operators for extra types

[SWS_LOG_00062]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const ara::core::StringView value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const ara::core::StringView value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Writes ara::core::StringView into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00051]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(const char *const value)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (const char *const value) noexcept;`	
Parameters (in):	value	Value to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	

▽

△

Header file:	#include "ara/log/log_stream.h"
Description:	Writes null terminated UTF8 string into message. (NOT sPECIFIED. WILL BE REMOVED IN FUTURE!)

⌋(*RS_LT_00003*)

[SWS_LOG_00063]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(LogStream &out, LogLevel value)	
Scope:	namespace ara::log	
Syntax:	`LogStream& operator<< (LogStream &out, LogLevel value) noexcept;`	
Parameters (in):	out	LogStream Object which is used to append the logged LogLevel (value) to
	value	LogLevel enum parameter as text to be appended to the internal message buffer.
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Thread Safety:	reentrant	
Header file:	#include "ara/log/log_stream.h"	
Description:	Appends LogLevel enum parameter as text into message.	

⌋(*RS_LT_00003*)

[SWS_LOG_00124]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(LogStream &out, const core::ErrorCode &ec)	
Scope:	namespace ara::log	
Syntax:	`LogStream& operator<< (LogStream &out, const core::ErrorCode &ec) noexcept;`	
Parameters (in):	out	the LogStream object into which to add the value
	ec	the ErrorCode instance to log
Return value:	LogStream &	out
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Write a core::ErrorCode instance into the message. When output to the console, the ErrorCode shall be shown in an implementation-defined way as a String holding the result of ErrorCode:Domain().Name() (i.e. the ErrorDomain's Shortname), and the integral error code number.	

⌋(*RS_LT_00003*)

[SWS_LOG_00125]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(LogStream &out, const std::chrono::duration< Rep, Period > &value)	
Scope:	namespace ara::log	
Syntax:	`template <typename Rep, typename Period>` `LogStream& operator<< (LogStream &out, const std::chrono::duration< Rep, Period > &value) noexcept;`	
Template param:	Rep	arithmetic type representing the number of ticks in this duration
	Period	a std::ratio type representing the tick period of the clock, in seconds
Parameters (in):	out	the LogStream object into which to add the value
	value	the duration instance to log
Return value:	LogStream &	out
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Write a std::chrono::duration instance into the message. When output to the console, the duration shall be shown as a decimal integer value, together with the duration's unit in SI notation, for at least all units in [std::nano, std::micro, std::milli, std::centi, std::deci, std::ratio<1>].	

⌋(*RS_LT_00049*)

[SWS_LOG_00126]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(LogStream &out, const ara::core::InstanceSpecifier &value)	
Scope:	namespace ara::log	
Syntax:	`LogStream& operator<< (LogStream &out, const ara::core::InstanceSpecifier &value) noexcept;`	
Parameters (in):	out	the LogStream object into which to add the value
	value	the InstanceSpecifier to log
Return value:	LogStream &	out
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Write a core::InstanceSpecifier into the message. The InstanceSpecifier shall be shown as the result of calling InstanceSpecifier::ToString.	

⌋(*RS_LT_00049*)

[SWS_LOG_00128]{DRAFT} ⌈

Kind:	function	
Symbol:	operator<<(core::Span< const core::Byte > data)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& operator<< (core::Span< const core::Byte > data) noexcept;`	
Parameters (in):	data	a Span<const Byte> covering the range to be logged

▽

△

Return value:	LogStream &	*this
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Write a byte sequence into message. This call shall copy the sequence of core::Byte objects as-is into the message. When output to the console, this byte sequence shall be shown as a sequence of apostrophe-separated list of hexadecimal octet-pairs, for instance: "48'65'6c'6c'6f"	

⌋(*RS_LT_00003*)

[SWS_LOG_00129]{DRAFT} ⌈

Kind:	function	
Symbol:	WithLocation(core::StringView file, int line)	
Scope:	class ara::log::LogStream	
Syntax:	`LogStream& WithLocation (core::StringView file, int line) noexcept;`	
Parameters (in):	file	the source file identifier
	line	the source file line number
Return value:	LogStream &	*this
Exception Safety:	noexcept	
Header file:	#include "ara/log/log_stream.h"	
Description:	Add source file location into the message. This function has no effect if another member function that adds content to the current message has already been called.	

⌋(*RS_LT_00003*)

8.3.2 Class Logger

The class `Logger` represents a logger context. The `Logging framework` defines contexts which can be seen as logger instances within one `Application process` or process scope.

The contexts have the following properties:

1) Context ID
2) Description of the Context ID
3) Default log level

A context will be automatically registered against the `Logging back-end` during creation phase, as well as automatically deregistered during process shutdown phase. So the end user does not care for the objects life time. To ensure such housekeeping functionality, a strong ownership of the logger instances needs to be ensured towards the `Logging framework`. This means that the `Application process` are not supposed to call the Logger constructor themselves.

The user is not allowed to create a Logger object by himself. Logger context needs to be created by the provided API call `CreateLogger()`.

8.3.2.1 Logger::LogFatal

[SWS_LOG_00064]{DRAFT} ⌈

Kind:	function	
Symbol:	LogFatal()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogFatal () const noexcept;`	
Return value:	LogStream	LogStream object of Fatal severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Creates a LogStream object. Returned object will accept arguments via the insert stream operator "@c <<".	
Notes:	In the normal usage scenario, the object's life time of the created LogStream is scoped within one statement (ends with ; after last passed argument). If one wants to extend the LogStream object's life time, the object might be assigned to a named variable.	

⌋*(RS_LT_00003)*

8.3.2.2 Logger::LogError

[SWS_LOG_00065]{DRAFT} ⌈

Kind:	function	
Symbol:	LogError()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogError () const noexcept;`	
Return value:	LogStream	LogStream object of Error severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Same as Logger::LogFatal().	

⌋(*RS_LT_00003*)

8.3.2.3 Logger::LogWarn

[SWS_LOG_00066]{DRAFT} ⌈

Kind:	function	
Symbol:	LogWarn()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogWarn () const noexcept;`	
Return value:	LogStream	LogStream object of Warn severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Same as Logger::LogFatal().	

⌋(*RS_LT_00003*)

8.3.2.4 Logger::LogInfo

[SWS_LOG_00067]{DRAFT} ⌈

Kind:	function	
Symbol:	LogInfo()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogInfo () const noexcept;`	
Return value:	LogStream	LogStream object of Info severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Same as Logger::LogFatal().	

⌋(*RS_LT_00003*)

8.3.2.5 Logger::LogDebug

[SWS_LOG_00068]{DRAFT} ⌈

Kind:	function	
Symbol:	LogDebug()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogDebug () const noexcept;`	
Return value:	LogStream	LogStream object of Debug severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Same as Logger::LogFatal().	

⌋(*RS_LT_00003*)

8.3.2.6 Logger::LogVerbose

[SWS_LOG_00069]{DRAFT} ⌈

Kind:	function	
Symbol:	LogVerbose()	
Scope:	class ara::log::Logger	
Syntax:	`LogStream LogVerbose () const noexcept;`	
Return value:	LogStream	LogStream object of Verbose severity.
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Same as Logger::LogFatal().	

⌋(*RS_LT_00003*)

8.3.2.7 Logger::IsEnabled

[SWS_LOG_00070]{DRAFT} ⌈

Kind:	function	
Symbol:	IsEnabled(LogLevel logLevel)	
Scope:	class ara::log::Logger	
Syntax:	`bool IsEnabled (LogLevel logLevel) const noexcept;`	
Parameters (in):	logLevel	The to be checked log level.
Return value:	bool	True if desired log level satisfies the configured reporting level.

▽

△

Exception Safety:	noexcept
Header file:	#include "ara/log/logger.h"
Description:	Check current configured log reporting level. Applications may want to check the actual configured reporting log level of certain loggers before doing log data preparation that is runtime intensive.

⌋(*RS_LT_00003*)

8.3.2.8 Logger::WithLevel

[SWS_LOG_00131]{DRAFT} ⌈

Kind:	function	
Symbol:	WithLevel(LogLevel logLevel)	
Scope:	class ara::log::Logger	
Syntax:	`LogStream WithLevel (LogLevel logLevel) const noexcept;`	
Parameters (in):	logLevel	the log level to use for this LogStream instance
Return value:	LogStream	a new LogStream instance with the given log level
Exception Safety:	noexcept	
Header file:	#include "ara/log/logger.h"	
Description:	Log message with a programmatically determined log level can be written.	

⌋(*RS_LT_00003*)

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document.

Class	**DltArgument**			
Package	M2::AUTOSARTemplates::SystemTemplate::Dlt			
Note	This element defines an Argument in a DltMessage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
network Representation	SwDataDefProps	0..1	aggr	Definition of the networkRepresentation of the Dlt Argument.

Table A.1: DltArgument

Class	**DltLogChannel**			
Package	M2::AUTOSARTemplates::SystemTemplate::Dlt			
Note	This element contains the settings for the log/trace message output for a tuple of ApplicationId and ContextId (verbose mode) or a SessionId (non-verbose mode).			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
application Description	String	0..1	attr	This attribute can be used to describe the applicationId that is used in the log and trace message in more detail.
applicationId	String	1	attr	This attribute identifies the SW-C/BSW module in the log and trace message.
context Description	String	0..1	attr	This attribute can be used to describe the contextId that is used in the log and trace message in more detail.
contextId	String	1	attr	This attribute is used to group log and trace messages produced by a SW-C/BSW modules to distinguish functionality (representing e.g. a library of the adaptive foundation linked into the application).
dltLogChannel Design	DltLogChannelDesign	0..1	ref	This reference represents the identification of the design-time representation for the DltLogChannel that owns the reference. **Tags:**atp.Status=draft
dltMessage	DltMessage	*	ref	Reference to DltMessages that can be transported over the DltLogChannel in the DltPdu.
endpoint Configuration	PlatformModule EthernetEndpoint Configuration	0..1	ref	Network configuration (Protocol, Port, IP Address) for transmission of dlt messages on a specific VLAN. **Tags:**atp.Status=draft
logTraceDefault LogLevel	LogTraceDefaultLog LevelEnum	0..1	attr	This attribute allows to set the initial log reporting level for a logTraceProcessId (ApplicationId). **Tags:**atp.Status=draft
logTraceFile Path	UriString	0..1	attr	This attribute defines the destination file to which the logging information is passed. **Tags:**atp.Status=draft
logTraceLog Mode	LogTraceLogMode Enum	*	attr	This attribute defines the destination of log messages provided by the process. **Tags:**atp.Status=draft

△

Class	DltLogChannel			
nonVerbose Mode	Boolean	0..1	attr	This attribute defines whether this channel supports non-Verbose Dlt messages. If disabled only verbose mode messages shall be used. **Tags:**atp.Status=draft
serviceInstance ToPortPrototype Mapping	ServiceInstanceToPort PrototypeMapping	0..1	ref	Optional reference to a PortPrototype of the monitored Application in case that the communication over this port is monitored and defines the ContextId. **Tags:**atp.Status=draft
sessionId	PositiveInteger	0..1	attr	This attribute allows distinguishing log/trace messages from different instances of the same SW-C. It is required if sessionIdSupport of the aggregating DltConfig is True.

Table A.2: DltLogChannel

Class	DltLogChannelDesign			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SystemDesign			
Note	This meta-class has the ability to stand in for a DltLogChannel at the time when the DltLogChannel does not yet exist. But its future existence already needs to be considered during design phase and for that a dedicated model element is required. **Tags:** atp.Status=draft atp.recommendedPackage=DltLogChannelDesigns			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dltMessage	DltMessage	*	ref	Reference to DltMessages that can be transported over the DltLogChannel. **Tags:**atp.Status=draft
endpoint Configuration	PlatformModule EthernetEndpoint Configuration	0..1	ref	Network configuration (Protocol, Port, IP Address) for transmission of dlt messages on a specific VLAN. **Tags:**atp.Status=draft

Table A.3: DltLogChannelDesign

Class	DltLogChannelDesignToProcessDesignMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SystemDesign			
Note	This meta-class represents the ability to assign a Log&Trace Channel in the Design to a ProcessDesign. **Tags:** atp.Status=draft atp.recommendedPackage=DltLogChannelDesignToProcessDesignMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dltLogChannel Design	DltLogChannelDesign	1	ref	Reference to the Log&Trace channel that contains the log/trace message output. **Tags:**atp.Status=draft

▽

△

Class	DltLogChannelDesignToProcessDesignMapping			
processDesign	ProcessDesign	0..1	ref	Reference to the ProcessDesign that is monitored by the DltLogChannel. **Tags:**atp.Status=draft

Table A.4: DltLogChannelDesignToProcessDesignMapping

Class	DltLogChannelToProcessMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::LogAndTrace			
Note	This meta-class represents the ability to assign a Log&Trace Channel to a Process. **Tags:** atp.Status=draft atp.recommendedPackage=DltLogChannelToProcessMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dltLogChannel	DltLogChannel	0..1	ref	Reference to the Log&Trace channel that contains the settings for the log/trace message output for a tuple of ApplicationId and ContextId (verbose mode) or a Session Id (non-verbose mode). **Tags:**atp.Status=draft
process	Process	0..1	ref	Reference to the Process that is monitored by the DltLog Channel. **Tags:**atp.Status=draft

Table A.5: DltLogChannelToProcessMapping

Class	DltMessage			
Package	M2::AUTOSARTemplates::SystemTemplate::Dlt			
Note	This element defines a DltMessage.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dltArgument (ordered)	DltArgument	*	aggr	Ordered collection of DltArguments in the DltMessage.
messageId	PositiveInteger	1	attr	This attribute defines the unique Id for the DltMessage.
messageLine Number	PositiveInteger	0..1	attr	This attribute describes the position in the source file in which this log message was called.
messageSource File	String	0..1	attr	This attribute describes the source file in which this log message was called.
messageType Info	String	1	attr	This attribute describes the message Type

Table A.6: DltMessage

Class	**DltMessageCollectionSet**			
Package	M2::AUTOSARTemplates::SystemTemplate::Dlt			
Note	Collection of DltMessages **Tags:**atp.recommendedPackage=DltMessageCollectionSets			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dltMessage	DltMessage	*	aggr	Collection of DltMessages in the DltMessageCollection Set.

Table A.7: DltMessageCollectionSet

Class	**Executable**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	This meta-class represents an executable program. **Tags:** atp.Status=draft atp.recommendedPackage=Executables			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildType	BuildTypeEnum	0..1	attr	This attribute describes the buildType of a module and/or platform implementation.
loggingBehavior	LoggingBehaviorEnum	0..1	attr	This attribute indicates the intended logging behavior of the enclosing Executable.
minimumTimer Granularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable. **Tags:**atp.Status=draft
reporting Behavior	ExecutionState ReportingBehavior Enum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable.
rootSw Component Prototype	RootSwComponent Prototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context. **Tags:**atp.Status=draft
version	StrongRevisionLabel String	0..1	attr	Version of the executable. **Tags:**atp.Status=draft

Table A.8: Executable

Class	**LogAndTraceInstantiation**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::LogAndTrace			
Note	This meta-class defines the attributes for the Log&Trace configuration on a specific machine. **Tags:**atp.Status=draft			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *Identifiable*, *MultilanguageReferrable*, *NonOsModule Instantiation*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**LogAndTraceInstantiation**			
dltEcuId	String	0..1	attr	This attribute defines the name of the ECU for use within the Dlt protocol.
dltLogChannel	DltLogChannel	*	aggr	DltLogChannels that are configured for the log/trace message output **Tags:**atp.Status=draft
queueSize	PositiveInteger	0..1	attr	Length of the queue (in which messages can be stored before processing) in the unit "Log message".
sessionId Support	Boolean	0..1	attr	This attribute defines whether the sessionId is used or not.
timeBase Resource	TimeBaseResource	0..1	ref	This reference is used to describe to which time base the Log and Trace module has access. From the Time Base Resource the Log and Trace module gets the needed information to generate the time stamp. **Tags:**atp.Status=draft

Table A.9: LogAndTraceInstantiation

Enumeration	**LogTraceLogModeEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::LogAndTrace
Note	This enum defines the possible destinations of a log&trace message. **Tags:**atp.Status=draft
Literal	*Description*
console	Destination of log message will be the console output. **Tags:**atp.EnumerationLiteralIndex=0
file	Destination of log message will be a file on the file system. **Tags:**atp.EnumerationLiteralIndex=1
network	Log message will be transmitted over the communication bus. **Tags:**atp.EnumerationLiteralIndex=2

Table A.10: LogTraceLogModeEnum

Class	**Machine**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::MachineManifest			
Note	Machine that represents an Adaptive Autosar Software Stack. **Tags:** atp.Status=draft atp.recommendedPackage=Machines			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
default Application Timeout	EnterExitTimeout	0..1	aggr	This aggration defines a default timeout in the context of a given Machine with respect to the launching and termination of applications. **Tags:**atp.Status=draft

▽

△

Class	**Machine**			
environment Variable	TagWithOptionalValue	*	aggr	This aggregation represents the collection of environment variables that shall be added to the environment defined on the level of the enclosing Machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=environmentVariable, environment Variable.variationPoint.shortLabel atp.Status=draft
machineDesign	MachineDesign	1	ref	Reference to the MachineDesign this Machine is implementing. **Tags:**atp.Status=draft
module Instantiation	AdaptiveModule Instantiation	*	aggr	Configuration of Adaptive Autosar module instances that are running on the machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation.shortName atp.Status=draft
processor	Processor	1..*	aggr	This represents the collection of processors owned by the enclosing machine. **Tags:**atp.Status=draft
secure Communication Deployment	SecureCommunication Deployment	*	aggr	Deployment of secure communication protocol configuration settings to crypto module entities. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=secureCommunicationDeployment.short Name atp.Status=draft
trustedPlatform Executable LaunchBehavior	TrustedPlatform ExecutableLaunch BehaviorEnum	1	attr	This attribute controls the behavior of how authentication affects the ability to launch for each Executable.

Table A.11: Machine

Class	**PlatformModuleEthernetEndpointConfiguration**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::AdaptiveModule Implementation			
Note	This meta-class defines the attributes for the configuration of a port, protocol type and IP address of the communication on a VLAN. **Tags:** atp.Status=draft atp.recommendedPackage=PlatformModuleEndpointConfigurations			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *PlatformModuleEndpointConfiguration*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
communication Connector	EthernetCommunication Connector	0..1	ref	Reference to the CommunicationConnector (VLAN) for which the network configuration is defined. **Tags:**atp.Status=draft
ipv4MulticastIp Address	Ip4AddressString	0..1	attr	Multicast IPv4 Address to which the message will be transmitted.
ipv6MulticastIp Address	Ip6AddressString	0..1	attr	Multicast IPv6 Address to which the message will be transmitted.

▽

△

Class	**PlatformModuleEthernetEndpointConfiguration**			
tcpPort	PositiveInteger	0..1	attr	This attribute allows to configure a tcp port number.
udpPort	PositiveInteger	0..1	attr	This attribute allows to configure a udp port number.

Table A.12: PlatformModuleEthernetEndpointConfiguration

Class	**Process**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class provides information required to execute the referenced executable. **Tags:** atp.Status=draft atp.recommendedPackage=Processes			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference. **Tags:**atp.Status=draft
deterministic Client	DeterministicClient	0..1	ref	This reference adds further execution characteristics for deterministic clients. **Tags:**atp.Status=draft
executable	Executable	0..1	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the process is affiliated with.
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. **Tags:**atp.Status=draft
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the enclosing SoftwareCluster. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=draft
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations. **Tags:**atp.Status=draft

Table A.13: Process

Class	**ProcessDesign**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ProcessDesign			
Note	This meta-class has the ability to stand in for a Process at the time when the Process does not yet exist. But its future existence already needs to be considered during design phase and for that a dedicated model element is required.. **Tags:** atp.Status=draft atp.recommendedPackage=ProcessDesigns			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
deterministic ClientResource Needs	DeterministicClient ResourceNeeds	*	aggr	This aggregation represents the collection of applicable resource needs for the design of deterministic clients. **Tags:**atp.Status=draft
executable	Executable	0..1	ref	Reference to executable that is executed in the process. **Tags:**atp.Status=draft

Table A.14: ProcessDesign

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, DiagnosticDebounceAlgorithmProps, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, Event Handler, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, LinSlaveConfigIdent, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, PduActivationRoutingGroup, PncMapping Ident, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequired EventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:**xml.sequenceOffset=-90

Table A.15: Referrable

Class	**ServiceInstanceToPortPrototypeMapping**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceMapping

▽

△

Class	**ServiceInstanceToPortPrototypeMapping**			
Note	This meta-class represents the ability to assign a transport layer dependent ServiceInstance to a Port Prototype. With this mapping it is possible to define how specific PortPrototypes are represented in the middleware in terms of service configuration. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInstanceToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
enablesLog Trace	Boolean	0..1	attr	This attribute enables/disables Log&Trace for the communication on the referenced Port of the referenced process. True: Log&Trace is enabled. False: Log&Trace is disabled.
portPrototype	PortPrototype	0..1	iref	Reference to a specific PortPrototype that represents the ServiceInstance. **Tags:**atp.Status=draft **InstanceRef implemented by:**PortPrototypeIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process in which the enclosing Service InstanceToPortPrototypeMapping is executed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft
processDesign	ProcessDesign	0..1	ref	Reference to the ProcessDesign in which the Executable that contains the SoftwareComponent and the referenced PortPrototype is executed. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	Reference to a ServiceInstance that is represented in the Software Component by the mapped group of Port Prototypes. **Tags:**atp.Status=draft

Table A.16: ServiceInstanceToPortPrototypeMapping

Class	<<atpVariation>> **SwDataDefProps**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the Data Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet ▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
	△ • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:**vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
additionalNative TypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:**xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:**xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:**xml.sequenceOffset=180
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:**xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:**xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly ▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
				△ • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:**xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:**xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:**xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memory AllocationKeywordPolicy of the referenced SwAddr Method. **Tags:**xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:**xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:**xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:**xml.sequenceOffset=90
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:**xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:**xml.sequenceOffset=230

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:**xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:**xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:**xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:**xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:**xml.sequenceOffset=290
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:**xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:**xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the ▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
				△ second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:**xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:**xml.sequenceOffset=355

Table A.17: SwDataDefProps

Class	**System**			
Package	M2::AUTOSARTemplates::SystemTemplate			
Note	The top level element of the System Description. **Tags:**atp.recommendedPackage=Systems			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
fibexElement	FibexElement	*	ref	Reference to ASAM FIBEX elements specifying Communication and Topology. All Fibex Elements used within a System Description shall be referenced from the System Element. atpVariation: In order to describe a product-line, all Fibex Elements can be optional. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
interpolation Routine MappingSet	InterpolationRoutine MappingSet	*	ref	This reference identifies the InterpolationRoutineMapping Sets that are relevant in the context of the enclosing System.
mapping	SystemMapping	*	aggr	Aggregation of all mapping aspects relevant in the System Description. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=mapping.shortName, mapping.variation Point.shortLabel vh.latestBindingTime=postBuild
pncVector Length	PositiveInteger	0..1	attr	Length of the partial networking request release information vector (in bytes).
pncVectorOffset	PositiveInteger	0..1	attr	Absolute offset (with respect to the NM-PDU) of the partial networking request release information vector that is defined in bytes as an index starting with 0.

▽

△

Class	**System**			
rootSoftware Composition	RootSwComposition Prototype	0..1	aggr	Aggregation of the root software composition, containing all software components in the System in a hierarchical structure. This element is not required when the System description is used for a network-only use-case. atpVariation: The RootSwCompositionPrototype can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=rootSoftwareComposition.shortName, root SoftwareComposition.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
systemVersion	RevisionLabelString	1	attr	Version number of the System Description.

Table A.18: System

B History of Constraints and Specification Items

Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

B.1 Constraint and Specification Item History of this document according to AUTOSAR Release yy-mm

...