



European Component Oriented Architecture (ECOA®) Collaboration Programme: Preliminary version of the ECOA Architecture Specification Part 6: Options

Dassault Ref No: DGT 2041491-A
Thales DMS Ref No: 69398920-035 --

Issue: 7

Prepared by
Dassault Aviation and Thales DMS

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Note: This specification is preliminary and is subject to further adjustments. Consequently, users are advised to exercise caution when relying on the information herein. No warranties are provided regarding the completeness or accuracy of the information in this preliminary version. The final version of the document will be released to reflect further improvements.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

This Page Intentionally Left Blank

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

This Page Intentionally Left Blank

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Contents

0	Introduction	7
1	Scope	7
2	Warning	7
3	Normative References	7
4	Definitions	9
5	Abbreviations	9
6	OPTION DYNAMIC TRIGGER	10
6.1	Mechanisms	10
6.1.1	Behaviour	10
6.1.2	Operations	12
6.2	Software Interface	13
6.3	Platform Requirements	13
6.4	Link with Meta-Model	13
7	OPTION UINT64	16
8	OPTION INT64	17
9	OPTION FAULT HANDLER	18
9.1	Mechanisms	18
9.1.1	Error Categorization	19
9.1.2	Error Propagation and Recovery Actions	19
9.1.2.1	Application Errors Propagation and Recovery Actions	19
9.1.2.2	Infrastructure Errors propagation and Recovery Actions	20
9.1.3	Operations and faults	21
9.2	Software Interface	25
9.2.1	Types	25
9.2.2	Component Interface	26
9.2.2.1	Error notification at fault handler level	26
9.2.3	Container Interface	26
9.2.3.1	Recovery Action	26
9.3	Platform Requirements	28
9.4	Link with Meta-Model	28
10	OPTION EXTERNAL INTERFACE	29
10.1	Mechanisms	29
10.2	Software Interface	30
10.2.1	ECOA Software side	30
10.2.2	Non-ECOA Software side	30

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

10.3	Platform Requirements	30
10.4	Link with Meta-Model	30
11	OPTION WARM START CONTEXT	33
11.1	Mechanisms	33
11.2	Software Interface	35
11.3	Platform Requirements	35
11.4	Link with Meta-Model	35
12	OPTION UTC TIME	36
12.1	Mechanisms	36
12.2	Software Interface	36
12.3	Platform Requirements	36
12.4	Link with Meta-Model	36
13	OPTION SUPERVISION	37
13.1	Introduction to Supervision	37
13.1.1	Supervision of components	37
13.1.2	Supervision of executables	38
13.2	Key principles of Supervision in ECOA	38
13.3	Recommendations for supervisor components	38
13.3.1	The need for supervisor components	38
13.3.2	Supervisor components and error management	39
13.3.3	Supervisor component design	39
13.3.4	Supervisor component instances	39
13.3.5	Supervisor components and composites	39
13.3.6	Deployment of supervisor components	40
13.4	Supervision SW Interface	40
13.4.1	Component Interface	40
13.4.2	Container Interface	40
13.5	Link with Meta-Model	40
14	OPTION PINFO WRITE	41
14.1	Mechanisms	41
14.2	Software Interface	41
14.3	Platform Requirements	41
14.4	Link with Meta-Model	41
15	OPTION AUTO START EXTERNAL TASK	43
15.1	Mechanisms	43
15.2	Software Interface	43

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

16	OPTION COMM PORTS	44
1.1.1	Example	44
16.1.1	Types of ports	46
16.1.3	Mapping of an external operation on communication ports	48
16.1.4	Operation IDs	48
16.1.6	Network Endianness	49
16.1.7	Conditional links	49
16.1.8	“Shortcut” connections	49
16.1.9	Independence between applications	49
17	OPTION CONDITIONAL LINKS (UNDER DEFINITION)	52
17.1	Mechanisms	52

Figures

Figure 1	Dynamic Trigger Behaviour	11
Figure 2	Propagation path of non-fatal application error	19
Figure 3	Propagation path of fatal application errors	20
Figure 4	Propagation path of infrastructure errors	20
Figure 5	Error propagation path	21
Figure 6	Event faults propagation behaviour	22
Figure 7	Request-Response faults propagation behaviour part 1	23
Figure 8	Request-Response faults propagation behaviour part 2	24
Figure 9	Versioned Data faults propagation behaviour	25
Figure 10	Non-ECOA Software using an ECOA External Interface	29
Figure 11	Management of Component Warm Start Context	34
Figure 12	- ECOA Component Lifecycle	37
Figure 13	- Example of communication ports	45

Tables

Table 1	User and Warm Start Component Context Volatility	34
----------------	---	-----------

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

0 Introduction

The Architecture Specification provides the specification for creating ECOA®-based software systems. It describes the standardised programming interfaces and data-model that allow developers to produce ECOA® components and construct ECOA®-based software systems. It uses terms defined in the Definitions (Architecture Specification Part 2). The details of the other documents comprising the rest of this Architecture Specification can be found in Section 3.

This document is Part 6 of the Architecture Specification, and describes the ECOA® Options.

The ECOA Standard is modular. It is composed of a Core and Options, Extensions and programming language Bindings. This document deals with Options, in order to ease the understanding.

1 Scope

This Architecture Specification specifies a uniform method for design, development and integration of complex real time software systems using a component oriented based approach.

2 Warning

This specification represents the output of a research programme. Compliance with this specification shall not in itself relieve any person from any legal obligations imposed upon them. Product development shall rely on the BNAE publications of the ECOA Standard.

3 Normative References

Architecture Specification Part 1	Dassault Ref No: DGT 2041078-A Thales DMS Ref No: 69398915-035 -- Issue 7 Architecture Specification Part 1 – Concepts
Architecture Specification Part 2	Dassault Ref No: DGT 2041081-A Thales DMS Ref No: 69398916-035 -- Issue 7 Architecture Specification Part 2 – Definitions
Architecture Specification Part 3	Dassault Ref No: DGT 2041082-A Thales DMS Ref No: 69398917-035 -- Issue 7 Architecture Specification Part 3 – Mechanisms
Architecture Specification Part 4	Dassault Ref No: DGT 2041083-A Thales DMS Ref No: 69398918-035 -- Issue 7 Architecture Specification Part 4 – Software Interface
Architecture Specification Part 5	Dassault Ref No: DGT 2041084-A Thales DMS Ref No: 69398919-035 -- Issue 7 Architecture Specification Part 5 – High Level Platform Requirements

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Architecture Specification
Part 6

Dassault Ref No: DGT 2041491-A
Thales DMS Ref No: 69398920-035 --
Issue 7

Architecture Specification Part 6 – Options

Architecture Specification
Part 7

Dassault Ref No: DGT 2041086-A
Thales DMS Ref No: 69398925-035 --
Issue 7

Architecture Specification Part 7 – Metamodel

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

4 Definitions

For the purpose of this standard, the definitions given in Architecture Specification Part 2 apply.

5 Abbreviations

API	Application Programming Interface
COTS	Commercial Off-The-Shelf
ECOAS	European Component Oriented Architecture. ECOAS® is a registered trademark.
IoC	Inversion-of-Control
IT	Information Technologies
OS	Operating System
QoS	Quality of Service
RTOS	Real-Time Operating System
PINFO	Persistent Information
UML	Unified Modelling Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

Without prejudice to the property rights relating to the ECOAS AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOAS AS6 Standard to the ECOAS AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

6 OPTION DYNAMIC TRIGGER

Option DYNAMIC TRIGGER enables the definition of non-standard concrete Components (called DynamicTriggerManager Components), which are dedicated to the management of non-periodic triggers.

This option is interesting for synchronizing several Components on the same triggering Event, depending on the same delay or appointment date. This behaviour is not basically covered by the internal Component Trigger mechanism available in ECOA AS7 Core perimeter:

- An internal Component Trigger cannot trigger several Components,
- The Event associated to an internal Trigger is always considered to be activating for the related Component,
- An internal Component Trigger cannot be set using an absolute time; it only accepts a delay.

Moreover, option DYNAMIC TRIGGER is required to ensure the portability of some ECOA AS6 Components on ECOA AS7 platforms. Thus, it is defined to both cover and extend ECOA AS6 services regarding the need of managing dynamic triggers.

When this option is not available, it may be necessary to update ECOA AS6 Components so that they might be executed in an ECOA AS7 platform:

- Addition of specific standard AS7 Components behaving the same way than AS6 DynamicTriggerInstance Modules,
- Modification of Components Assembly and Deployment.

6.1 Mechanisms

Option DYNAMIC TRIGGER makes the Component Kind DynamicTriggerManager available.

The purpose of the DynamicTriggerManager Component is to define a one-shot event generator internally to a Composite scope: DynamicTriggerManager Component will act as a sender of a valued event at a given moment specified at runtime, either as waiting period or as a wake-up absolute time.

6.1.1 Behaviour

A DynamicTriggerManager Component is fully generated by infrastructure (i.e. no manual code) to behave this way:

- A DynamicTriggerManager Component sends an Event (known as the `out` Event) at a given moment deduced from the receipt of an input Event (which is either known as the `in_delay` Event or as the `in_time` Event). The `in_delay` Event specifies the delay in the form of a relative amount of time after which the `out` Event shall be sent. The `in_time` Event specifies the delay in the form of an absolute expiry time at which the `out` Event shall be sent.
- A DynamicTriggerManager Component may also receive a `reset` Event, which will purge all timers having unexpired delays.
- It is possible for multiple Component Instances to:
 - Send `in_delay`, `in_time` and `reset` Events to the same DynamicTriggerManager Component.
 - Receive the same `out` Event.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- Multiple occurrences of `in_delay` and `in_time` Events may be processed such that a number of delays are waiting to expire. A `reset` Event is used to purge all currently active timers.
- The first parameter present in an `in_delay` or `in_time` Event must respectively be the relative amount of time or the absolute expiry time allowing to set the internal timer within the DynamicTriggerManager Component. Optional remaining user parameters of these input Events can be of any ECOA type or type derived from ECOA types.
- When a Component Instance uses an `in_time` Event to invoke a DynamicTriggerManager Component, it can calculate the expiry time parameter to be sent by summing the intended delay and the current **absolute system time** obtained by calling the `get_absolute_system_time` API. Otherwise, it can simply use an `in_delay` Event, which must directly contain the expected delay as first parameter.
- The `out` Event is generated with exactly the same parameters as the `in_delay` or `in_time` Event, except that the first parameter is omitted. The `out` Event is sent at the deduced expiry time, except if the expiry time is already in the past when the DynamicTriggerManager Component processes the `in_delay` or `in_time` Event. In such failure case, the `out` Event is sent immediately and the ECOA Infrastructure raises an Error to the Fault Management.
- The lifecycle of a DynamicTriggerManager is controlled the same way than any other Component.

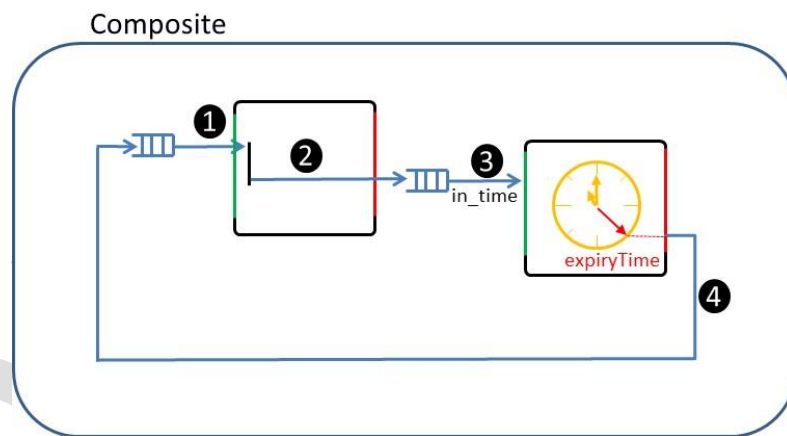


Figure 1 Dynamic Trigger Behaviour

Figure 1 shows, at **point 1**, an Operation being invoked on a Component Instance as a result of some other activity. During this execution, the Component Instance performs, at **point 2**, a Container Operation to request the DynamicTriggerManager Component Instance to send a Trigger Event after a given period (`in_time` Event) specified via the absolute `expiryTime` parameter. The `in_time` Event is queued in the DynamicTriggerManager Component Instance Queue, at **point 3**, otherwise if the queue is full, the event is discarded and an error raised to the Fault Management.

The queue is processed by the DynamicTriggerManager Component, and if the maximum number of concurrent delays has been reached, the delay is discarded and an error raised to the Fault Management, otherwise the delay is started. After the delay time has expired (i.e. when the current time becomes equal to or greater than the expiry time), the DynamicTriggerManager Component Instance will generate an `out` Event to the Receiver Component Instance Queue, shown at **point 4**.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

6.1.2 Operations

The DynamicTriggerManager Component can be considered as a Component whose operations are:

- EventReceived in_delay
 - On reception, the DynamicTriggerManager Component sets the trigger with an expiryTime deduced from the delay received in the “in_delay” Event
 - Parameters:
 - A delay (mandatory): expressed as **duration** and typed by ECOA:duration (seconds, nanoseconds)
 - p1, p2, etc.: optional user-defined parameters based on ECOA types
- EventReceived in_time
 - On reception, the DynamicTriggerManager Component sets the trigger for the expiryTime received in the “in” Event
 - Parameters:
 - An expiry time (mandatory): expressed as **Absolute System Time** and typed by ECOA:global_time (seconds, nanoseconds)
 - p1, p2, etc.: optional user-defined parameters based on ECOA types
- EventSent out
 - The Dynamic Trigger sends an “out” Event at the expiry time received in the related “in_delay” or “in_time” Event.
 - Parameters:
 - p1, p2, etc.: are identical (number, types and position identical to those of the “in_delay” or “in_time” Events)
- EventReceived reset
 - On reception, the DynamicTriggerManager Component cancels the trigger settings for all “in_delay” or “in_time” Events previously received and not yet expired (regardless of the Components that made those pending trigger requests).

The transmitted Events ‘in_delay’, ‘in_time’ and ‘out’ can have several optional user parameters (p1, p2, etc.):

- These parameters are sent unchanged by the DynamicTriggerManager Component using ‘out’ Event.
- The number of parameters and their types are strictly the same for “in_delay” and “in_time” Events. They are defined in the Component Type of the DynamicTriggerManager Component (see §6.4).
- The list of these parameters must be defined taking into account the needs of all Components connected to the DynamicTriggerManager Component. This list is necessarily common to all client Components.

Whatever they are mandatory or optional, all parameters present in the input and output Events of a DynamicTriggerManager Component must be explicitly defined its ComponentType.

This is different from ECOA AS6 modeling rules where mandatory parameters of a DynamicTriggerInstance are implicitly defined.

A DynamicTriggerManager Component may present only one of the two input Events “in_time” and “in_delay” if the second one is not useful for the Application architecture : this avoids having unconnected operations in the Assembly.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

ECOAS Model consistency rule: each Component connected to a DynamicTriggerManager Component shall declare the same optional user parameters in its own Events than those defined in the Component Type of the DynamicTriggerManager Component (respecting the same order).

6.2 Software Interface

DynamicTriggerManager Components implementation is implicit and managed by the infrastructure. So, there is no dedicated API for this option.

All inputs required for the infrastructure code generation are defined in the ECOAS model.

6.3 Platform Requirements

Id	Requirement
HLR.XX	When the ECOAS Platform offers option DYNAMIC_TRIGGER, a default value shall be defined and documented for the maximal number of possible concurrent pending timers managed by the same instance of a DynamicTriggerManager Component.

6.4 Link with Meta-Model

DYNAMIC_TRIGGER_MANAGER is a specific **ComponentKind** in the Component Type metamodel.

The model of a DynamicTriggerManager Component may require a specific additional element, which is compliant to the reference AS7 Metamodel (see ECOAS AS7 Part7):

- An optional metadata named **"maxPendingWakeups"** introduced at ComponentType level, which specifies the maximum number of pending triggering requests within the DynamicTriggerManager Component (both "in_delay" and "in_time" Events). When this metadata is not defined, a default value is applied according to the infrastructure specification.

Example:

- File: Timer.comp.xml

```
<componentType kind="DYNAMIC_TRIGGER_MANAGER">
  <meta name="maxPendingWakeups" type="int8" value="10"/>
  <operations>
    <eventReceived name="in_time"/>
    <parameter name="time" type="ECOAS:global_time"/>
    <parameter name="user_param1" type="int32"/>
    <eventReceived name="in_delay"/>
    <parameter name="delay" type="ECOAS:duration"/>
    <parameter name="user_param1" type="int32"/>
    <eventReceived name="reset"/>
    <eventSent name="out"/>
  </operations>
</componentType>
```

Without prejudice to the property rights relating to the ECOAS AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOAS AS6 Standard to the ECOAS AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

    <parameter name="user_param1" type="int32"/>
  </operations>

</componentType>

```

- File: MyComponent.comp.xml

```

<componentType kind="STANDARD">
  <operations>
    <eventReceived name="wake_up">
      <parameter name="user_param1" type="int32"/>
    </eventReceived>
    <eventSent name="reset_timer"/>
    <eventSent name="set_timer">
      <parameter name="when" type="EOCA:global_time"/>
      <parameter name="param1" type="int32"/>
    </eventSent>
  </operations>
</componentType>

```

- File: MyApp.assembly.xml

```

<instance name="CP1" componentType="MyComponent" implementation="C" />
<instance name="CP2" componentType="Timer" implementation="C" />

<links>
  <eventlink>
    <sender operation="set_timer" instance="CP1"/>
    <receiver operation="in_time" instance="CP2"/>
  </eventlink>
  <eventlink>
    <sender operation="out" instance="CP2"/>
    <receiver operation="wake_up" instance="CP1"/>
  </eventlink>
  <eventlink>
    <sender operation="reset_timer" instance="CP1"/>
    <receiver operation="reset" instance="CP2"/>
  </eventlink>
</links>

```

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Note that if there is no other use of a Timer Component in the Assembly schema, it is possible not to define interface “in_delay” in the ComponentType of Timer Component.

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

7 OPTION UINT64

This option allows the use of basic type ECOA:uint64.

Option entirely defined in ECOA AS7 Parts 1 to 5 and 7.

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

8 OPTION INT64

This option allows the use of basic type ECOA:int64.

Option entirely defined in ECOA AS7 Parts 1 to 5 and 7.

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

9 OPTION FAULT HANDLER

This option allows offering ECOA-standardized Fault Handling services at infrastructure level, which covers:

- The ability to define a Fault Handler Component which is triggered in case of error,
- The definition of one or several recovery actions the Fault Handler may apply when handling the error.

When a Component is declared as a Fault Handler, it means that this Component is responsible for applying a recovery procedure in case of error.

The list of possible standardized treatments to be executed in a recovery procedure depends on options and extensions actually defined in the platform. Particularly, option FAULT HANDLER can be advantageously combined with:

- Option WARM START CONTEXT (see section 11)
- Option UTC TIME (see section 12)
- MULTI_APP_ASSEMBLY and FAULT MANAGEMENT extensions (which offer complementary system level features).

By default (ECOA core perimeter), Fault Management is entirely managed by the platform, according to specifications supported by the platform supplier (i.e. out of the ECOA standard).

Option FAULT HANDLER is a solution to allow Application providers to have control over the definition of recovery procedures (except when the Fault Handler Component itself is faulty). There is another solution based on SUPERVISION option (see §13) which, although it has different features, also addresses fault management issues.

In both cases, the portability of applications is improved by ensuring a higher level of determinism of behaviors in case of failure.

Options FAULT HANDLER and SUPERVISION are not mutually exclusive, but there are rules to be respected when both are used (see hereafter).

9.1 Mechanisms

A Fault Handler Component is a concrete Component (i.e. not a Composite Component).

There is at most one Fault Handler Component in an Application.

A Fault Handler Component should be initialized and started before all other Application Components initialization that are not Trigger Components. There are exceptions when the Fault Handler Component is also a Supervisor Component (see section 13).

A Fault Handler Component offers a specific entry-point to process error notifications. This entry-point is triggered by the ECOA Infrastructure in case of an Application error (i.e. error detected by a concrete Component of the Application, apart from the Fault Handler Component itself) or in case of an Infrastructure error (i.e. error detected at Component Container level or at Executable level, or by the ECOA Infrastructure).

The ECOA Infrastructure associates a timestamp to any error that it raises to a Fault Handler Component.

When an error is raised by a Fault Handler Component, it is managed according to platform specifications (which depend whether ECOA extensions are available or not).

Specific case of an Application combining SUPERVISION and FAULT HANDLER options:

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- There is at most one Supervisor Component in an Application that implements a Fault Handler Component, and these two components shall be then one and the same: thus, the Supervisor Component is also the Fault Handler Component (i.e. attribute kind="SUPERVISOR" in Component Type and Option element "isFaultHandler" is set in Component Implementation).
- Recovery procedures are only applied using the Fault Handler specific entry-point.

9.1.1 Error Categorization

See Architecture Specification Part 4 §9.4.6.

9.1.2 Error Propagation and Recovery Actions

9.1.2.1 Application Errors Propagation and Recovery Actions

Figure 2 illustrates the propagation of a Non-fatal application error raised by a concrete Component, which is not a Fault Handler.

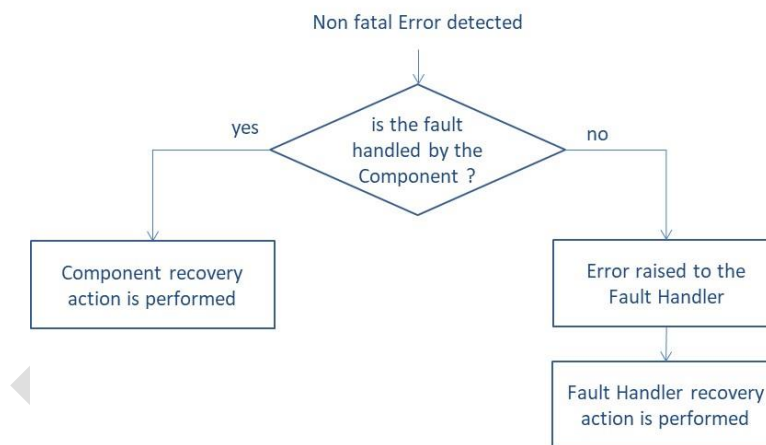
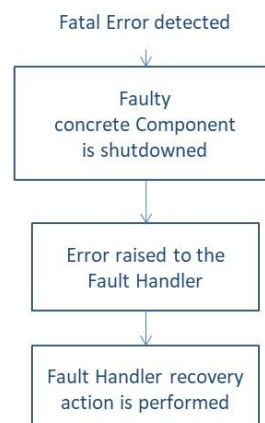


Figure 2 Propagation path of non-fatal application error

Figure 3 illustrates the propagation of a fatal error raised by a concrete Component, which is not a Fault Handler.



Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Figure 3 Propagation path of fatal application errors

9.1.2.2 Infrastructure Errors propagation and Recovery Actions

An Infrastructure error is a fault detected at Infrastructure level, i.e. at Component Container level or Executable level. Figure 4 illustrates Infrastructure error propagation:

- If the Infrastructure error involves the Fault Handler (itself or its related Executable), it is handled by the platform that applies a default recovery action.
 - Otherwise:
 - If the Infrastructure error is detected by a Component Container, it is raised to the Fault Handler
 - If the Infrastructure error comes from an Executable, it is directly detected by the Fault Handler.
- Then, the Fault Handler logs the fault and applies the corresponding recovery procedure.

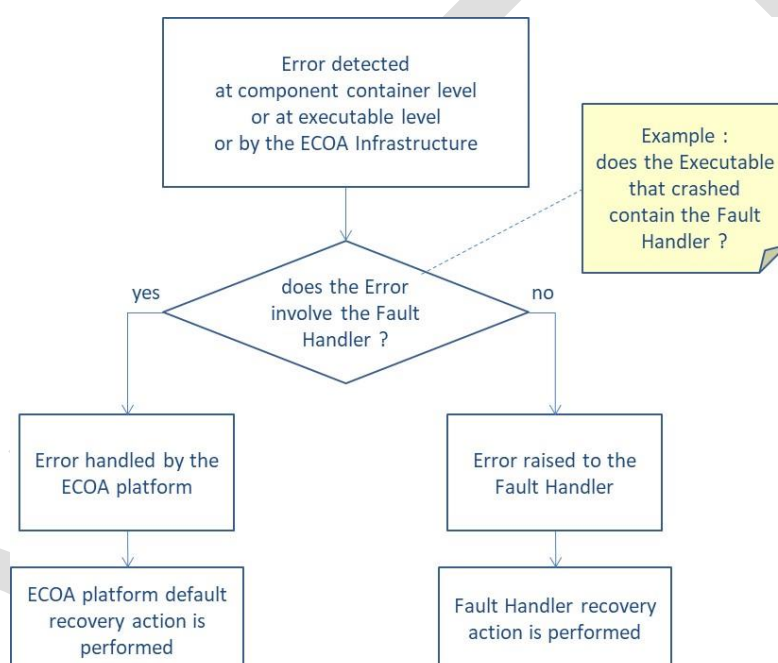


Figure 4 Propagation path of infrastructure errors

The parameterization of the Fault Handler recovery routines (i.e. the recovery actions to be applied) is the responsibility of the system architect and the system integrator. Recovery actions for Infrastructure errors may be applied at some levels, depending on the capabilities procured in the ECOA Platform:

- At Component level, the Fault Handler may:
 - Shutdown the faulty Component.
 - Warm/cold restart this Component
- At Composite Component level (restricted to "Leaf Composites", i.e. Composites containing only concrete Components), the Fault Handler may:
 - Shutdown the Composite containing the faulty Component
 - Warm/cold restart the Composite containing the faulty Component.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- At Executable level, the Fault Handler may:
 - Shutdown the Executable containing the faulty Component.
 - Warm/cold restart this Executable

Note: the warm restart recovery actions supported by the ECOA Platform should be consistent with that ECOA Platform ability to maintain the warm start context of Component Instances upon those recovery actions.

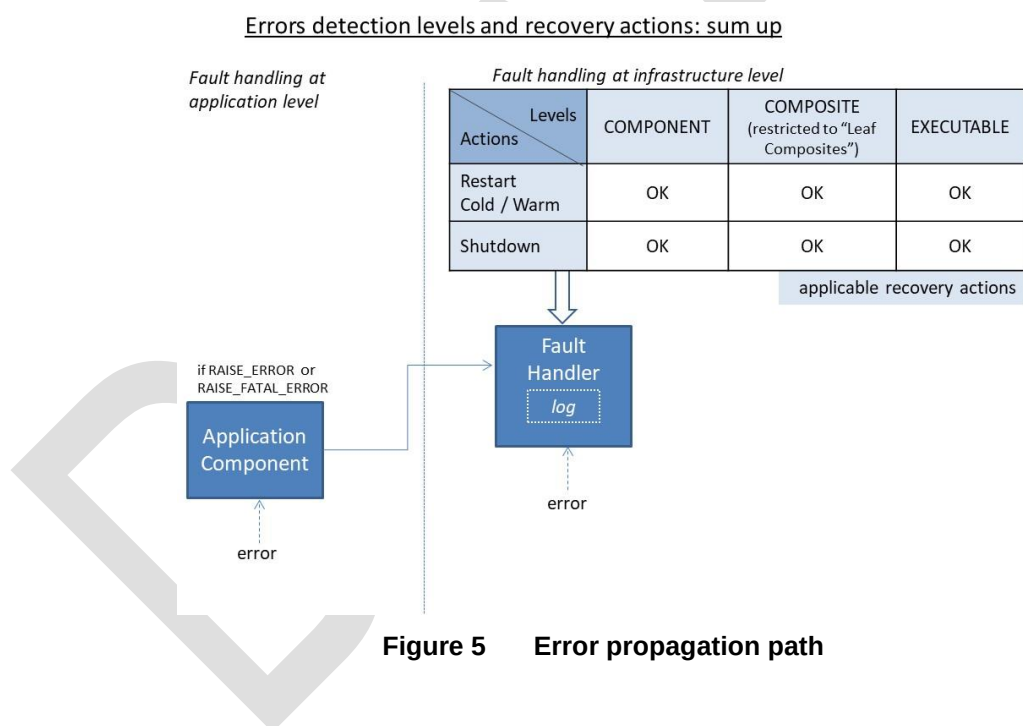
For an X-level recovery procedure:

Shutdown action consists of the Infrastructure putting all Component instances within X-perimeter in idle state (with the Trigger Instances being first). Then, the Infrastructure frees associated resources, without calling the shutdown entry points of these potentially faulty Component Instances. Therefore, this is a fast shutdown and not a graceful functional shutdown and is not part of the Component Lifecycle.

Restart action consists of the Infrastructure triggering a shutdown action over Component Instances within X-perimeter, then initialize and start lifecycle actions on these Component Instances (in the manner of a local fast startup).

In order to enable fault filtering, the Fault Handler has a context which can be used as memory to store the number of occurrences of a fault over a time frame. As an example, this could be used to enable a sporadic fault to be distinguished from a recurrent fault and applying a recovery action only if a given fault occurs more than X times in a given time frame.

Figure 5 summarizes the error propagation path and applicable recovery actions.



9.1.3 Operations and faults

ECOAS provides the component supplier with three types of operations for use in Application Components:

- Event
- Versioned Data
- Synchronous and Asynchronous Request/Response

Without prejudice to the property rights relating to the ECOAS AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOAS AS6 Standard to the ECOAS AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Each of these operations has specific behaviours regarding identified errors and infrastructure errors.

The following section provides Infrastructure errors that may held by the Fault Handler. Mandatory and optional Error codes returned to the Component are defined in {Standard No} Part 4.

Faults occuring outside a component area are managed by the platform.

Note: Figures in this section provide examples of infrastructure error propagation paths to the Fault Handler. Actual paths may depend on Platform implementation.

Event:

- Infrastructure errors handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE client container unable to send the event
 - OPERATION_NOT_AVAILABLE server cannot handle the event
 - UNKNOWN_OPERATION requested operation ID is invalid

Figure 6 illustrates these behaviours.

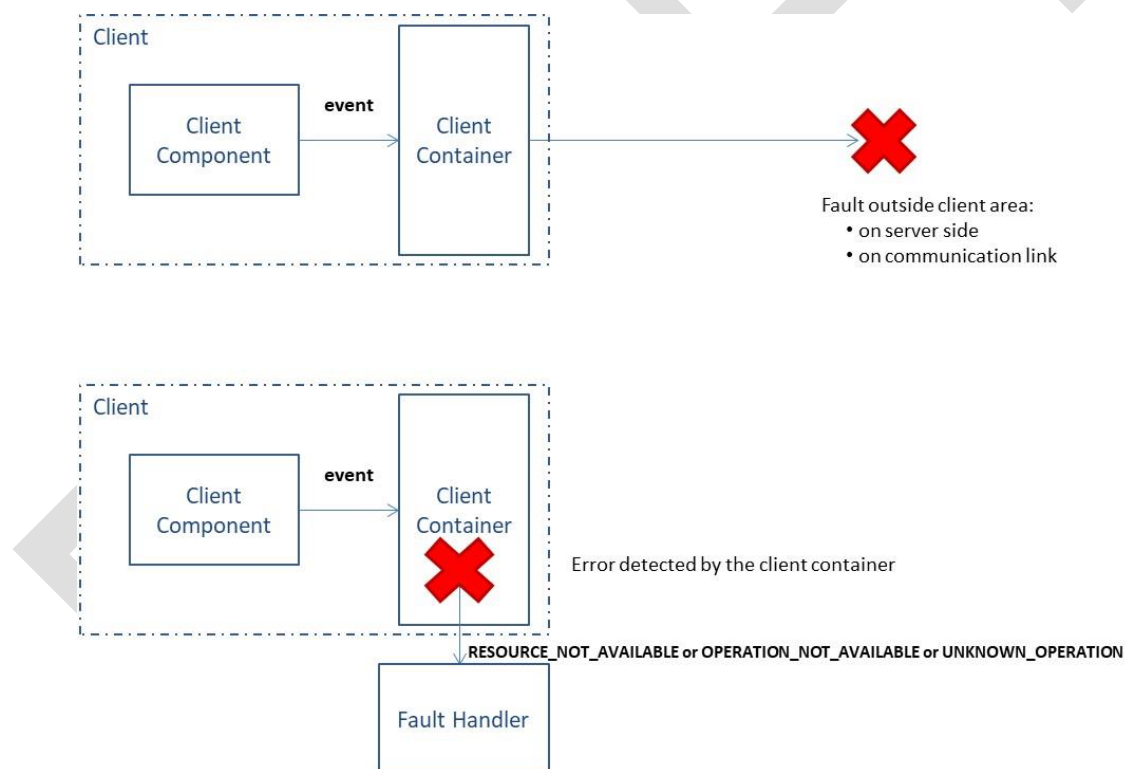


Figure 6 Event faults propagation behaviour

Request-Response:

- Request faults:
 - Infrastructure errors handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE client container unable to send the request
 - OPERATION_NOT_AVAILABLE server cannot handle the request

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- UNKNOWN_OPERATION requested operation ID is invalid
- Response faults:
 - Infrastructure errors handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE server container unable to send the response
 - OPERATION_NOT_AVAILABLE client cannot handle the response
 - OVERFLOW server container unable to retain the request (maxConcurrentRequests has been reached)

The Figure 7 and Figure 8 illustrate these behaviours.

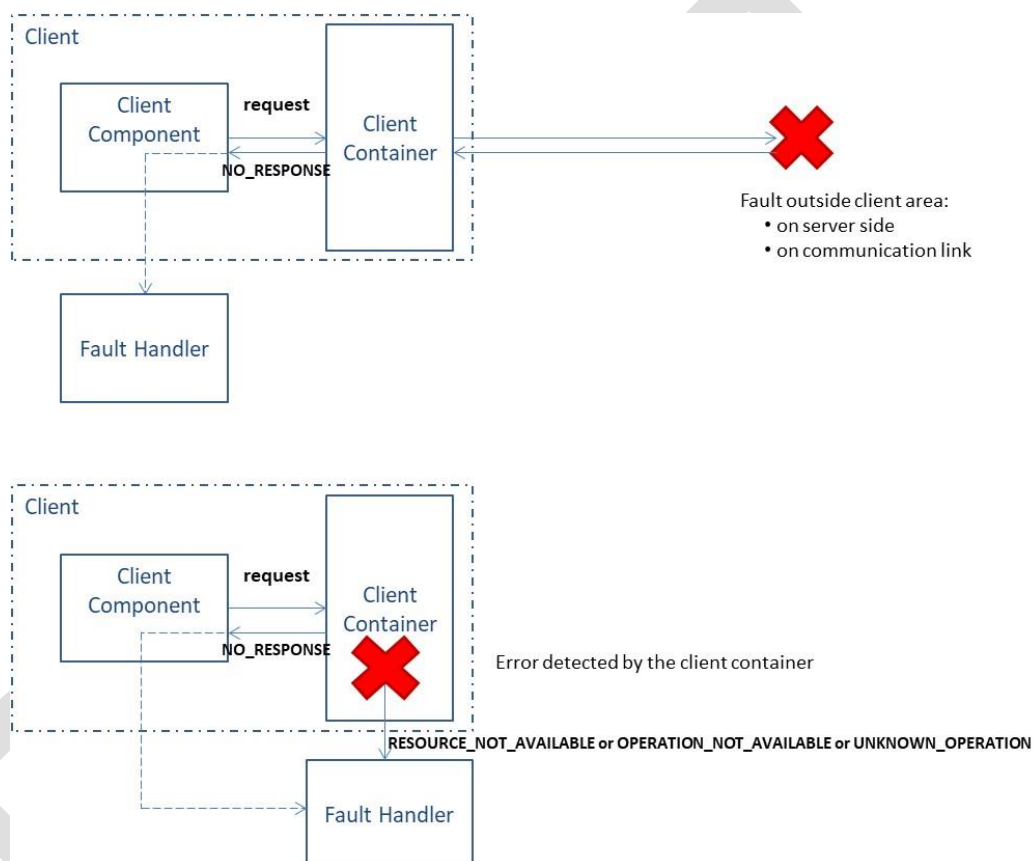


Figure 7 Request-Response faults propagation behaviour part 1

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

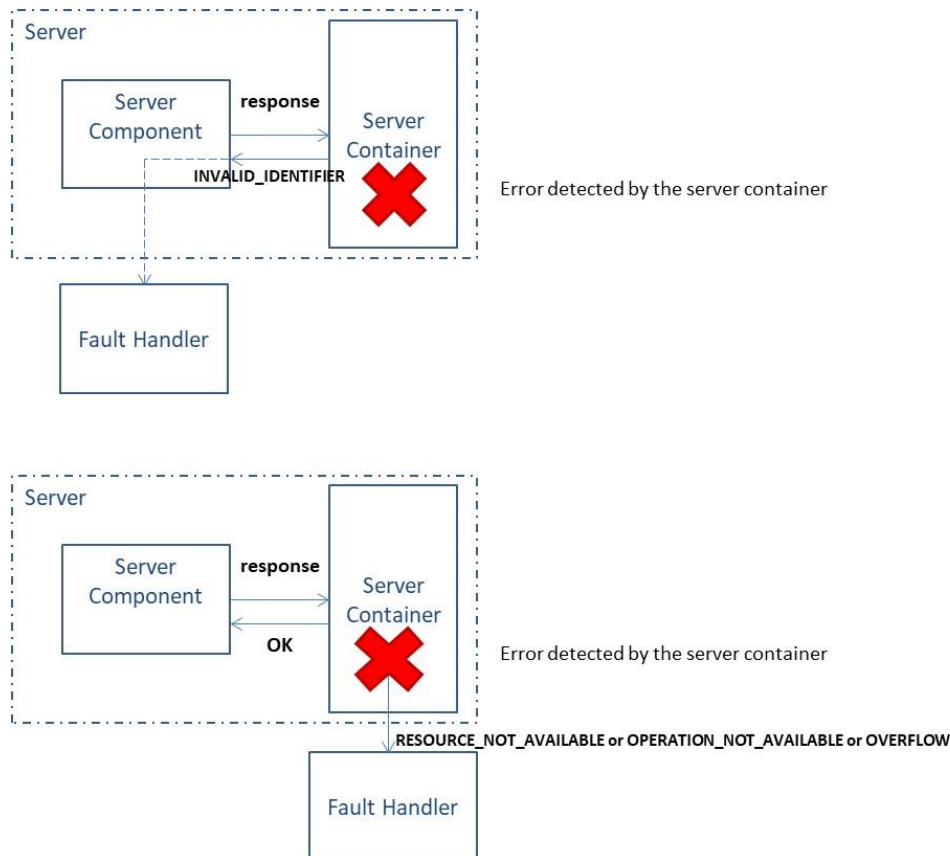


Figure 8 Request-Response faults propagation behaviour part 2

Note that in the case of a NO_RESPONSE error being detected by the client Container it is possible that two recovery actions are performed as a result of a single originating fault. This is due to the fact that the Infrastructure error is handled by the Fault Handler, but in addition, it is also possible that the Component raises an error to the Fault Handler.

Note also that when a fault is detected outside of the client area after a request call, the platform supplier has the possibility to choose between two implementations: he can choose to send NO_RESPONSE to the client though the ECOA Infrastructure, or do nothing and wait for the timeout (if specified) to happen on the client side (which causes the client Container to send a NO_RESPONSE to the client Module).

Versioned Data:

- Get_read_access:
 - Infrastructure error handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE client container unable to access the memory slot
- Release_read_access
 - Infrastructure error handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE client container unable to access the memory slot
- Get_write_access:

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- Infrastructure error handled by the Fault Handler:
 - RESOURCE_NOT_AVAILABLE client container unable to access the memory slot
- Publish_write_access:
 - Infrastructure error handled by the Fault Handler
 - RESOURCE_NOT_AVAILABLE client container unable to access the memory slot
- Cancel_write_access:
 - No Infrastructure error

The Figure 9 illustrates these behaviours.

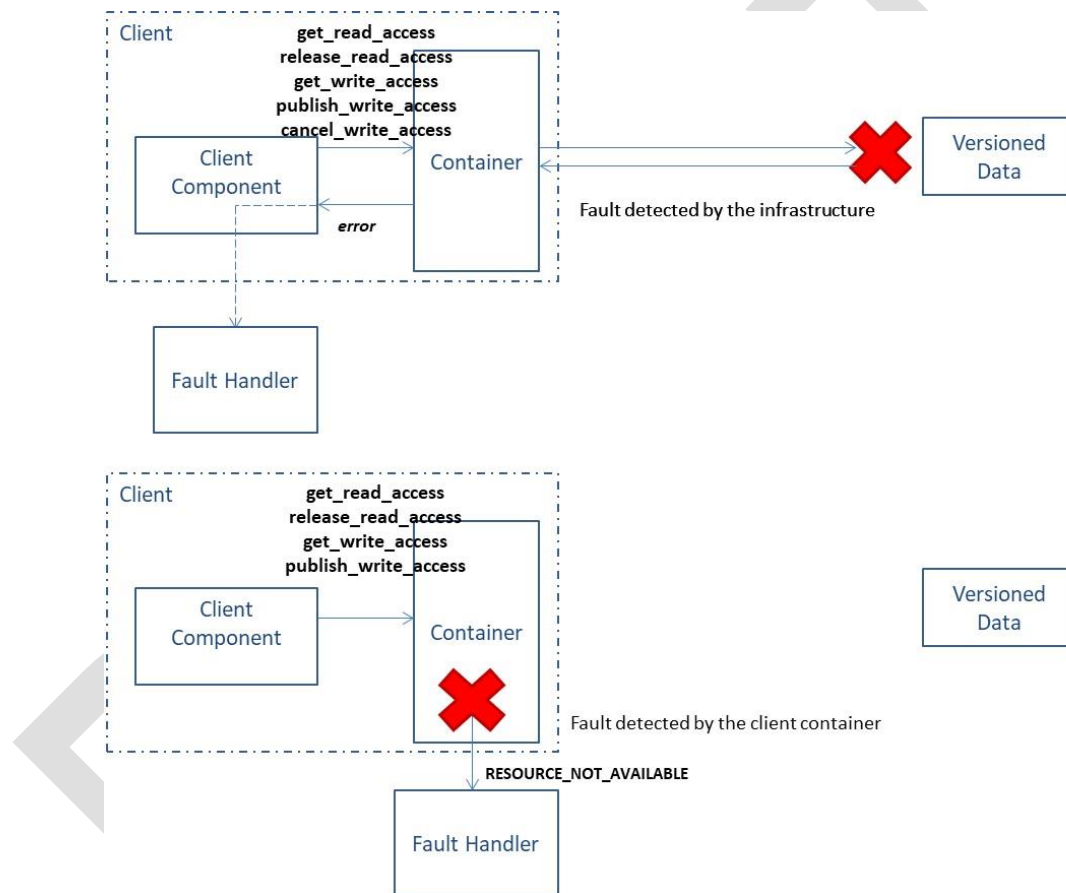


Figure 9 Versioned Data faults propagation behaviour

9.2 Software Interface

9.2.1 Types

See ECOA Architecture Specification Part 4 related to OPTION FAULT HANDLER.

In addition, the following type is defined: ECOA:recovery_action_type.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

The data type **ECOА:recovery_action_type** is an enumeration declared in the ECOА predefined library, which is used to specify the type of recovery action the Infrastructure will carry on the target asset (Composite Component, concrete Component, Task or Executable).

The enumeration values are:

- | | | |
|----------------|---|--|
| • SHUTDOWN | 0 | Shutdown the target asset |
| • COLD RESTART | 1 | Restart the target asset in cold mode |
| • WARM RESTART | 2 | Restart the target asset in warm mode (if available) |

9.2.2 Component Interface

See ECOА Architecture Specification Part 4 related to OPTION FAULT HANDLER.

9.2.2.1 Error notification at fault handler level

9.2.3 Container Interface

9.2.3.1 Recovery Action

After analysis of error notifications, the Fault Handler carries on recovery actions provided by the Container Interface API.

Recovery actions cover actions on Components (Composite or concrete), and Executables. There are two recovery actions – Shutdown and Restart. (See general description in §9.2.2.1).

Note: it is being assumed in the following that the ECOА Platform is able to maintain warm start context according to the Warm Restart recovery actions being specified below (see option WARM START CONTEXT §11).

- Shutdown concrete Component:
 - The Infrastructure puts the targeted Component Instance in idle state. The Infrastructure frees associated resources, without calling the shutdown entry point of the potentially faulty Component Instance.
- Restart concrete Component in cold or warm mode:
 - The Infrastructure performs the “Shutdown Component” recovery action (see above).
 - In case of cold start, the Infrastructure zeroes the ‘saved’ value of the Warm Start Context of the component. In case of warm start, the Infrastructure restores the ‘saved’ value of the Warm Start Context without modifying it.
 - The Infrastructure initializes the Component Instance (INITIALIZE entry-point).
 - The Infrastructure starts the Component Instance (START entry-point).
- Shutdown Composite Component (restricted to “Leaf Composites”):
 - The Infrastructure puts in idle state all Component Instances defined in the Composite. The Infrastructure frees associated resources, without calling the shutdown entry point of these potentially faulty Component Instances.
- Restart Composite Component in cold or warm mode (restricted to “Leaf Composites”):
 - The Infrastructure performs the “Shutdown Composite” recovery action (see above).
 - In case of cold start, the Infrastructure zeroes the ‘saved’ value of the Warm Start Context for all Components Instances defined in the Composite. In case of warm start, the Infrastructure restores the ‘saved’ value of the Warm Start Context without modifying it.

Without prejudice to the property rights relating to the ECOА AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOА AS6 Standard to the ECOА AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- The Infrastructure initializes all Component Instances defined in the Composite (INITIALIZE entry-point).
- The Infrastructure starts all Components Instances defined in the Composite (START entry-point).
- Shutdown the Executable:
 - The Infrastructure puts in idle state all concrete Component Instances deployed in the targeted Executable. The Infrastructure frees associated resources, without calling the shutdown entry point of these potentially faulty Component Instances
 - Depending on underlying capabilities, the infrastructure may unload the Executable from memory.
- Restart the Executable in cold or warm mode :
 - The Infrastructure performs the “Shutdown Executable” recovery action (see above).
 - Depending on underlying capabilities, the Infrastructure may unload the Executable from memory and reload it to restart from a clean technical context.
 - In case of cold start, the Infrastructure zeroes the ‘saved’ value of the Warm Start Context for all Components Instances deployed within the targeted Executable. In case of warm start, the Infrastructure restores the ‘saved’ value of the Warm Start Context without modifying it.
 - The Infrastructure initializes all Component Instances deployed within the targeted Executable (INITIALIZE entry-point).
 - The Infrastructure starts all Components Instances deployed within the targeted Executable (START entry-point).

The availability of recovery actions depends on the capabilities offered by the underlying platform.

Recovery actions are implemented by a single container operation.

Interface specifying elements:

Abstract API Name	Recovery_Action			
Use Case	Fault Handler Component launches a recovery action			
Level	OPTIONAL (i.e. MANDATORY when option FAULT HANDLER is available)			
Minimal variability patterns	#error_handler_impl_name#			
Mandatory parameters	Name	Abstract Type	IN/OUT	Role
	P1	#context#	IN/OUT	component context
	P2	ECO:A:recovery_action_type	IN	recovery action to be applied
	P3	ECO:A:asset_id	IN	asset ID on which the recovery action is applied
	P4	ECO:A:asset_type	IN	type of asset targeted by the recovery action (Composite Component, concrete Component, Task or Executable)
	P5	ECO:A:return_status	OUT	status on interface execution

The appropriate language binding will define the correct syntax for this component operation.

The following illustration provides an abstract definition of the interface to help binding elaboration:

```
ECO:A:return_status [#error_handler_impl_name#_container:]recovery_action([#context#],
ECO:A:recovery_action_type recovery_action, ECO:A:asset_id asset_id, ECO:A:asset_type asset_type);
```

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

The ECOA platform shall be able to manage at least the following return status codes:

- [ECOA:return_status:OK]
 - No error
- [ECOA:return_status:FAILURE]
 - Any error that is not managed by a dedicated status code

Some ECOA platforms may offer complementary return status codes to provide a more accurate analysis of failure cases. For example:

- [ECOA:return_status:OPERATION_NOT_AVAILABLE]
 - The recovery action is not implemented by the Infrastructure
 - The recovery action is not permitted for the targeted asset type
- [ECOA:return_status:INVALID_IDENTIFIER]
 - Operation called with an unknown asset ID
 - Operation called with an asset ID not consistent with the asset type
- [ECOA:return_status:OPERATION_ALREADY_PENDING]
 - Pending operation on the same asset already in progress

9.3 Platform Requirements

Id	Requirement
HLR.XX	The ECOA Platform may support ECOA Fault Handling at infrastructure level (option FAULT HANDLER).
HLR.XX	When the ECOA Platform offers option FAULT HANDLER, it shall support implementation of the ECOA Fault Handler as an Application Component.
HLR.XX	When the ECOA Platform offers option FAULT HANDLER, it shall support the detection of at least one application error or Infrastructure error.
HLR.XX	When the ECOA Platform offers option FAULT HANDLER, it shall support at least one recovery action.

9.4 Link with Meta-Model

In a Component Implementation model, option element **isFaultHandler** allows to indicate if the related component is a Fault Handler.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

10 OPTION EXTERNAL INTERFACE

Option EXTERNAL INTERFACE offers an alternative to External Components core mechanisms to enable communication with non-ECOASoftware. It is inherited from a mechanism defined in ECOAS6 for external communications.

So, it is mainly introduced as an ECOAS7 option to ensure the portability of legacy software based on ECOAS6 onto AS7 platforms.

It can also be used for new Driver Components, particularly when they have to exchange with a multi-threaded non-ECOASoftware.

10.1 Mechanisms

Option EXTERNAL INTERFACE creates a communication mechanism that operates as follows:

- A Driver Component launches non-ECOASoftware and declares an Event to receive outputs from this software,
- No other Driver Component is allowed to receive this Event,
- In the Assembly schema, the sender related to the eventLink connected to this Event is declared as an external instance using an External Interface,
- The generated container derived from the implementation of the receiving Driver Component offers a dedicated API allowing the non-ECOASoftware to post the Event (with or without parameters) in the related reception queue of the unique Driver Component Instance. The binding language to be applied for this API is deduced from this implementation model.

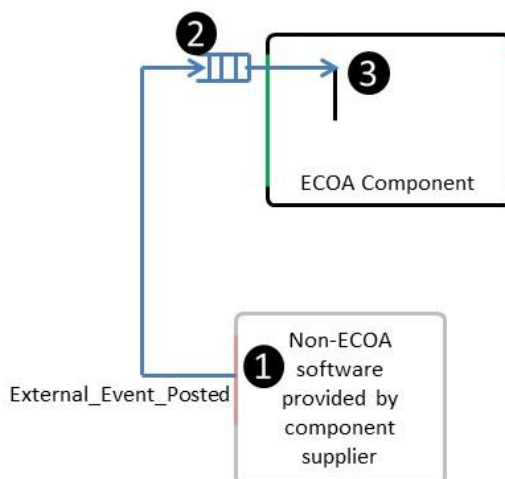


Figure 10 Non-ECOASoftware using an ECOA External Interface

Figure 10 shows an example of how non-ECOASoftware asynchronously interacts with one Operation of a Component Instance through a defined external interface (**point 1**).

When the ECOA external interface (implemented by the Component container) is invoked by the non-ECOASoftware at **point 1**, an event is placed into the Component Instance queue at **point 2**. The event will be processed as usual, and the associated operation will be invoked on the Component Instance at **point 3**. The Module Instance can then perform the appropriate processing for the received event.

Without prejudice to the property rights relating to the ECOAS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOAS6 Standard to the ECOAS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

10.2 Software Interface

External Interface option is associated to two different abstract API operations:

Category	Abstract API Name	Container Operation	Component Operation	Level	Section
Events API	Event_Received	No	Yes	MANDATORY	Part 4
External Interface	External_Event_Posted	Yes	No	OPTIONAL	10.2.2

10.2.1 ECOA Software side

ECOA concrete Components which are receivers of an External Event simply use the AS7 Core API for event reception (see Abstract API named « Event_Received” in Architecture Specification Part 4).

10.2.2 Non-ECOA Software side

An API is provided for non-ECOA Software, enabling it to post a message to ECOA concrete Components in the form of an Event operation:

Interface specifying elements:

Abstract API Name	External_Event_Posted			
Use Case	allows non-ECOA software to put an event in an ECOA component queue			
Level	OPTIONAL			
Minimal variability patterns	#component_impl_name#; #external_operation_name#			
Mandatory parameters	Name	Abstract Type	IN/OUT	Role
	P1	#event_parameters#	IN	input external data

The appropriate language binding will define the correct syntax for this component operation.

The following illustration provides an abstract definition of the interface to help binding elaboration:

```
void [#component_impl_name#_]#external_operation_name#(const #event_parameters#);
```

Such interfaces are provided by containers of ECOA Component Instances and can be directly called by non-ECOA Software.

This API only exists for ECOA platform which implements option EXTERNAL INTERFACE.

10.3 Platform Requirements

The notion of Driver Component is not relevant for the platform (See Architecture Specification Part 1).

10.4 Link with Meta-Model

The notion of Driver Component is not formalized in the metamodel (See Architecture Specification Part 1).

Nevertheless, Components which are linked to external operations are, by definition, Driver Components.

An External Interface is defined as a classical eventReceived operation in the ComponentType of an ECOA Driver Component.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

An eventReceived is recognized as an External Interface for the Component by analyzing the Assembly schema, where the related eventLink then includes the optional attribute **isExternalInterface** set to true in the definition of the external sender.

The example below shows an eventReceived operation “TheFeedback” that may be used to asynchronously notify the Driver Component of an event.

01-Components/cp_type2/cp_type2.comp.xml:

```
<componentType kind="STANDARD">
  <operations>

    <eventReceived name="TheFeedback">
      <input name="param1" type="uint32"/>
    </eventReceived>

  </operations>
</componentType>
```

01-Components/cp_type2/C/cp_type2.C.impl.xml:

```
<implementation>
  <language.c fullName="cp_type2">
</ implementation >
```

02-Assemblies

```
<instance name="MyComp" componentType="cp_type2" implementation="C"/>
```

In this example, “FeedbackLegacy” is an external operation connected to “MyComp” Component. As attribute “isExternalInterface” is set to true on this external operation, it means that “TheFeedback” is an External Interface for “MyComp” Component. The binding language to be applied to generate the external interface API is deduced from language specifications set in “MyComp” Component implementation model.

```
<eventLink>
  <sender instance="extern" operation="FeedbackLegacy"
isExternalInterface="true"/></sender>
  <receiver instance="MyComp" operation="TheFeedback"/>
</eventLink>
```

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

This will result in an external API being provided by the Container, which may have the following prototype definition:

```
void [#component_impl_name#_]FeedbackLegacy(const ECOA:uint32 param1);
```

This API is generated (in files `cp_type2_container.h` and `cp_type2_container.cpp`) with the purpose to be used in non-ECOA software, to post an event in the receiver ECOA Component Instance queue. From the ECOA Component point of view, the event posted this way is associated to “TheFeedback” event entry-point (based on the abstract API `Event_Received`).

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

11 OPTION WARM START CONTEXT

When a Component is not stateless, its source code should save the Component state using the Component Context, and should not use global variables. This enables a platform to instantiate a single component implementation multiple times and manage instance specific data contained in the Component Context.

As explained in Architecture Specification Part 3, in a Platform only implementing ECOA Core perimeter, the only available Component Context is the User Context, which is non-optional and always volatile.

Option WARM START CONTEXT offers concrete Components to have a second Component Context called Warm Start Context, which enables warm start recovery procedures that may be defined by the Fault Management (according to warm start scenarios available in the Platform, i.e. using Fault Handler or Supervisor Components, or even in situations possibly out of ECOA standard scope).

When this option is not available, in case of a component restart, only cold starting is possible: the Component is responsible for initializing its internal data through the Component INIT entry-point (which may rely on a PINFO to set initial values)

Option WARM START CONTEXT may be required to ensure compatibility with ECOA AS6 Components.

11.1 Mechanisms

Warm Start Context is instance specific optional data, defined and managed by the Component Implementation, to allow it:

- To maintain a private state between transitions of Component Lifecycle states,
- And to restore this private state where the transition is INIT.

The Component Implementation has access to a Container API so that it may request the Container to save the Warm Start Context in non-volatile storage such that it may be restored during a future Warm Restart. Warm Restart can be initiated by the Fault Handler if available. The Infrastructure shall perform the save operation atomically. If the save operation fails, no error is returned to the calling Component; however an error may be raised to the Fault Handler.

The ability to maintain the warm start context of Component Instances upon warm start recovery actions is a Platform procurement requirement. This is aimed at tailoring ECOA Platforms complexity to actual procurement needs. As previously stated, the warm restart recovery actions supported by an ECOA Platform should be consistent with that ECOA Platform ability to maintain the warm start context of Component Instances upon those recovery actions. In this section, it is being assumed that this consistency is fulfilled.

Example: if a Platform does not support warm restart recovery actions, the Platform does not need to offer option WARM START CONTEXT.

The calling Component is responsible for determining the validity of the Warm Start Context. This may be achieved by incorporating additional validity entries in the Warm Start Context itself.

For a Cold Start from power up or a Cold Restart initiated by a Fault Handler, the 'saved' value of the Component Warm Start Context is zeroed by the infrastructure.

In this way for a Warm Start initiated by a Fault Handler, the Component Warm Start Context is either restored from a previously zeroed version, or the most recently saved version.

Warm Start Context used by a Component Instance is zeroed/restored by the infrastructure prior to any activation of the Initialize Component Instance entry point.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Note: to know whether it initializes from a Warm Start or from Cold Start, a Component may test the value of a dedicated data in its Warm Start Context.

Note: the Component supplier shall initialize the User Context every time the Initialize Component Instance entry point has been called, and shall never assume the User Context has been maintained by the Infrastructure.

For restarts initiated by the physical platform, the Warm Start Component Context is non-volatile. Table 1 details the volatility of the User and Warm Start Module Context.

Table 1 User and Warm Start Component Context Volatility

User Context		Warm Start Context	
Warm	Cold	Warm	Cold
Volatile	Volatile	Non Volatile	Volatile

Figure 11 shows how a Component Warm Start Context is managed through these transitions. When Component state is UNAVAILABLE (see Architecture Specification Part 3), Component Context is unavailable too.

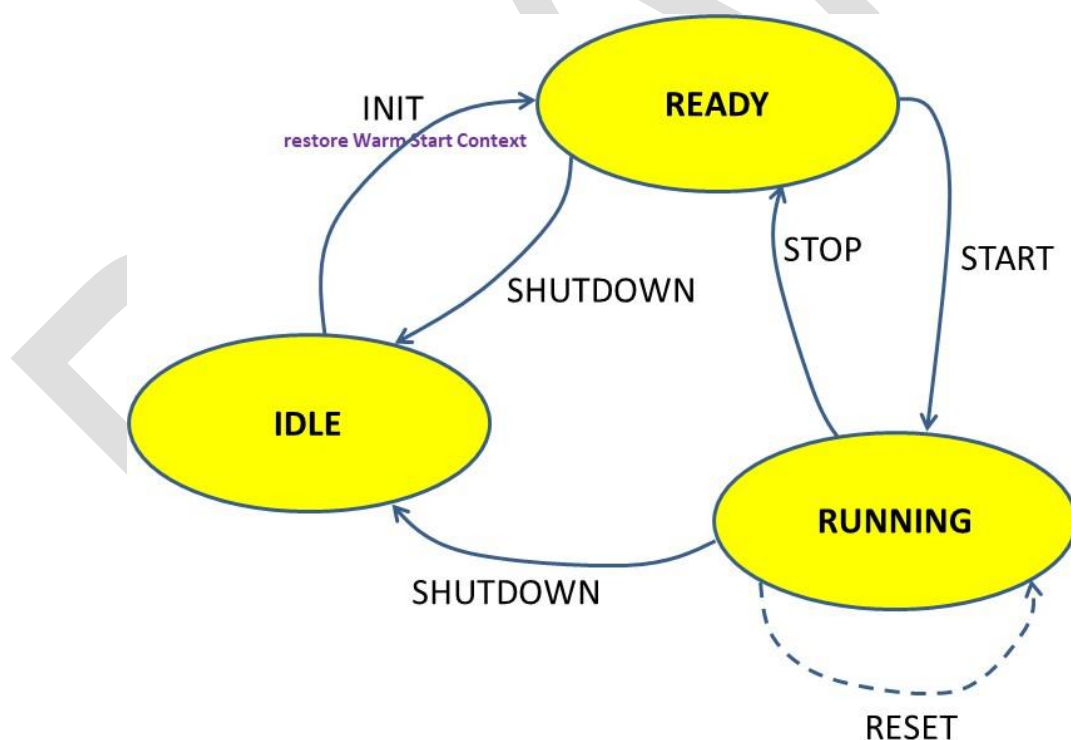


Figure 11 Management of Component Warm Start Context

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Warm Start Context is optional. It is only necessary for Component Instances maintaining an internal state between operation calls whose restoration is expected in some cases or restart procedure.

11.2 Software Interface

Architecture Specification Part 4 describes the detail of how Warm Start Context is represented in various languages and the language bindings specifies the detail of the implementation.

Related items in Part 4:

- Definition of warm start context content,
- Operation “Save_Warm_Start_Context”.

11.3 Platform Requirements

Id.	Requirement
	Implementation
HLR.XX	When option WARM START CONTEXT is available, the ECOA Platform shall be able to maintain the warm start context of Component Instances, depending on the possible warm start scenarios supported by the ECOA Platform.
HLR.XX	When the ECOA platform offers warm restart scenarios, option WARM START CONTEXT shall be available to maintain the warm start context of Component Instances upon that recovery action.
HLR. XX	The ECOA platform supplier shall define the possible scenarios of Component cold restart and warm restart.

11.4 Link with Meta-Model

When option WARM START CONTEXT is available:

- It is possible to set Option element “hasWarmStartContext” in Component Implementations, so that these Components might benefit from a warm start scenario,
- It is necessary to take into account warm start context size in the BinaryDescription of a Component Implementation (for relevant components).

Note that even if option WARM START CONTEXT is available, there still may be Component Implementations having Option Element “hasWarmStartContext” set to false.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

12 OPTION UTC TIME

Option UTC TIME gives access to Coordinated Universal Time (UTC), which is a reference absolute time that can be used to synchronize several actors in a system, or in a system of systems.

This global time may also be useful to date events (if other available times do not cover datation needs).

12.1 Mechanisms

UTC Time availability relies on platform hardware requirements to provide a very accurate and stable clock, which is regularly synchronized with a reference time source.

12.2 Software Interface

See Time Services API in ECOA AS7 Part 4:

- Get_UTC_Time
- Get_UTC_Time_Resolution

UTC Time can be used to date log messages, in accordance with format defined in ECOA AS7 Part 4:

```
<SYSTEM_TIME>:<IS_UTC>:<MESSAGE_LEVEL>:<COMPONENT_INSTANCE_NAME>:<MESSAGE_TEXT><EOL>
```

12.3 Platform Requirements

See ECOA AS7 Part 5.

12.4 Link with Meta-Model

Option UTC TIME is required to execute components having Option element “**needsUTCTime**” set in their Component Implementation.

Some components may moreover require information on time resolution: then, for them, Option element “**needsTimeResolution**” has to be set in Component Implementation. But this attribute is not specific to UTC Time : it applies to all ECOA time services which are available in the platform.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

13 OPTION SUPERVISION

13.1 Introduction to Supervision

This chapter complements the ECOA Part 1 document with the Supervision Option specific concepts.

Supervision in ECOA can be broken down in two parts:

- Supervision of components,
- Supervision of executables.

Note: See also option FAULT HANDLER. Options FAULT HANDLER and SUPERVISION are not mutually exclusive, but there are rules to be respected when both used (see §9).

13.1.1 Supervision of components

Part 1 document introduces the concept of Component Lifecycle, which is defined as the state of each component instance inside an application.

This state is one of:

- UNAVAILABLE
- IDLE
- READY
- RUNNING

Transitions are possible between some of these states according to the following diagram:

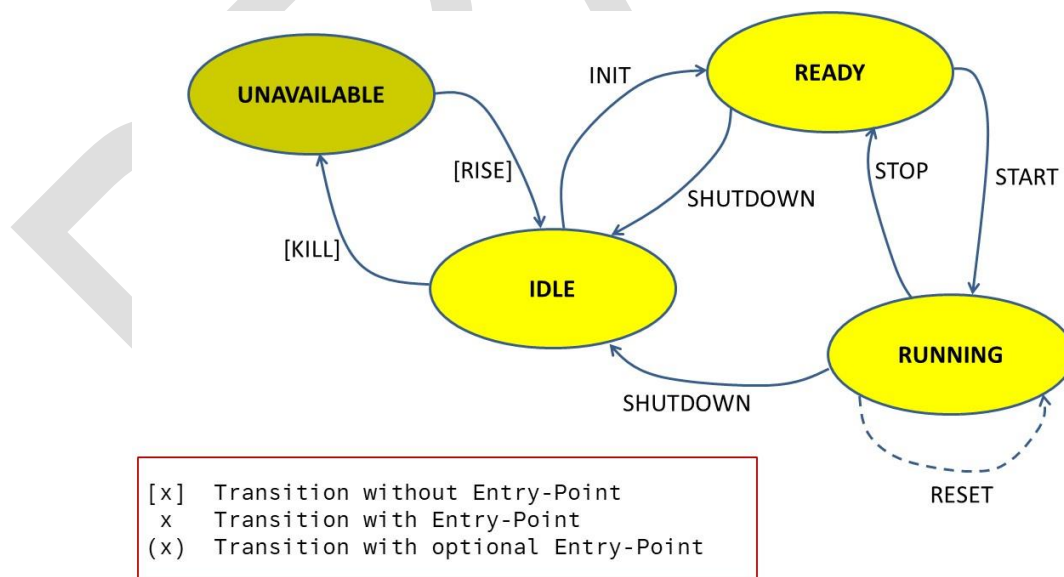


Figure 12 - ECOA Component Lifecycle

Without option Supervision, some properties are defined about these states (e.g. a component cannot be activated by an incoming operation unless it is in RUNNING state), but nothing is defined about how this Component Lifecycle is managed:

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- When the state changes
- Who can access the current state

Basically, considering ECOA Core perimeter, this is all platform-dependent. For example:

- A platform may systematically initialise all components, then start them,
- Or component states may be commanded manually through a control panel, or by some scripting (useful for test and integration),
- Or by another mechanism of the platform, not standardized within ECOA.

The option Supervision gives the possibility to control (observe) and to command (modify) the state of components, from the application itself, through special components called “Supervisors”.

13.1.2 Supervision of executables

Part 1 document introduces the concept of Executables, but nothing is specified regarding when they are started, stopped, restarted, etc.

Supervisors (introduced in the previous section) have the capability to control (observe) and to command (modify) the state of executables.

13.2 Key principles of Supervision in ECOA

Some components are identified, in their Component Type model, as special components of the kind SUPERVISOR (cf. [Part 7]).

A SUPERVISOR component has all the capabilities of a standard component: it may define operations, properties; it may be instantiated in an assembly.

There shall be at most one instance of SUPERVISOR component in an application.

Unlike other components except FAULT HANDLER component, a SUPERVISOR component type is the only type of component that will have the knowledge of the current Application's assembly and deployment.

A SUPERVISOR component can access the list of all component instances of the Application, access the state of all component instances (including itself), and control it (initialize, start, stop, shutdown cf. Component state diagram).

A SUPERVISOR component can access the list of all the Executables of the Application, access their state (started or not started), and start and stop them.

A SUPERVISOR component is notified of all component state changes, through an additional entry point that is executed by the infrastructure every time a component state changes (“on_state_change” API). This notification is independent of the supervisor's lifecycle state (i.e. it is run even if the supervisor component is IDLE or READY). This allows a supervisor component to initialize and start any component or executable. In particular, it allows a supervisor to initialize and start itself.

13.3 Recommendations for supervisor components

13.3.1 The need for supervisor components

A supervisor is not always needed in an ECOA application.

The ECOA platform can start all components automatically, using “fast” or “synchronized” value of the start_mode attribute in the deployment model (cf. PART 3 §7.7.1.1).

A supervisor is generally needed for one of the following reasons:

- To start/stop executables inside the application.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- To start components in a specific order when the application starts (including Trigger Components)
- To start/stop selected components, according to a logic of “modes” inside the application.
- To reinitialize and restart components after they shut down or on some other conditions.

Note: In the case the start_mode attribute is set to “none”, a supervisor component is necessary to start the components of the application.

13.3.2 Supervisor components and error management

A component can be both a SUPERVISOR component and a FAULT HANDLER component (see [FAULT HANDLER] option). The recommended approach for error management is:

- Components handle errors locally, as much as possible, in their own code (using return statuses). They perform their functions if they can, and stop doing so if they cannot. Through their operations (generally a data operation), they can publish information about their functional capabilities, which may be more or less degraded. If their functional capabilities reach zero, components may voluntarily stop, by calling the Raise_fatal_error API.
- The application has a Fault Handler component, that captures errors coming from the ECOA platform, and consolidates them and applies a recovery procedure.
- This component is also a supervisor component, so it is informed of the components' lifecycle (through its special API). It is also informed of errors coming from the ECOA platform because it is a Fault handler, and of functional modes elaborated by components (through normal ECOA operations defined in its component type). Based on this information, it can stop other components (if component A stops working, then also stop components B and C), and possibly, restart components according to a custom logic tailored on the components' functional dependencies.

13.3.3 Supervisor component design

The main role of a supervisor component is to **decide which components and executables shall be active or not at a given moment**, within an ECOA application.

A supervisor component should remain **simple and small**. It should focus on the management of the lifecycle of components of an assembly, and should not include additional functions except fault handling. It can do this using other information (such as functional modes, various events, etc.) provided by other components.

A supervisor component is usually **not designed to be reusable**. It is specific for a given assembly. It has a dependency on the list of component instances of the assembly. If it uses executables, it also has a dependency on the deployment.

13.3.4 Supervisor component instances

Although several supervisor components may be instantiated in an assembly, it is recommended to have **only one supervisor component instance** in a given assembly.

13.3.5 Supervisor components and composites

No supervisor component shall be instantiated inside a composite.

In other words, a supervisor component instance shall be part of the top-level assembly of an application.

Reminders:

- A composite component is a component made of other components.
- A composite component has no lifecycle state.
- A supervisor component has control over all the components of the application, including components inside composites.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Notes/Rationale:

- Defining supervisor components at different levels of the composite hierarchy does not seem adequate, because it would lead to complexities in determining the actual behaviour of the whole set of components (startup order, startup time, etc.).
- Unlike other components, supervisors are tightly linked to the application as a whole. They are dependent on the assembly and on the deployment of this assembly. They are not designed to be reusable across different applications.

13.3.6 Deployment of supervisor components

A supervisor component shall be deployed in the main executable of the application. It shall not be deployed in a sub-executable of the application.

It is recommended to deploy a supervisor component in a task with a high priority, so that it is not interrupted by other tasks.

13.4 Supervision SW Interface

This chapter complements ECOA Part 4 SW Interface document with the Supervision Option specific SW Interface.

13.4.1 Component Interface

The Component Interface specifies the interface to a component, which is used by the container to call component operations.

Refer to ECOA AS7 Part 4.

13.4.2 Container Interface

Refer to ECOA AS7 Part 4.

13.5 Link with Meta-Model

In the Component Type model, the attribute “kind” must be set to “SUPERVISOR”.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

14 OPTION PINFO WRITE

Option PINFO WRITE allows write access on PINFOs to be granted under strict conditions.

When this option is not available, Components can only access to PINFOs in read-only mode.

14.1 Mechanisms

A Component can access to a PINFO in write mode if the following conditions are satisfied:

- All conditions listed for read access are satisfied (see ECOA AS7 Part 3)
- The PINFO is not shared between several Component instances: a single Component instance is both the single writer and the single reader of this PINFO.
Indeed, the ECOA platform shall not allow PINFO reading and writing mechanisms to be a disguised exchange mechanism. An important part of real time properties of ECOA platforms rely on the mandatory use of Event, Versioned Data and Request Response mechanisms to achieve exchanges between Components.
- The current size of the PINFO is lower than the maximal authorized size.
The maximal size of a PINFO may be specified in the ECOA model. Otherwise, the default value defined by the platform provider is applied.
- The maximal size value may be a reference to a constant value.

Before runtime, the platform shall allocate appropriate memory resources for writable PINFOs.

At runtime, the platform shall control any access to a writable PINFO:

- Read and Write access on the same PINFO are mutually exclusive.
- The platform forbid writing operations out of allocated resources.

14.2 Software Interface

See ECOA AS7 part 4.

14.3 Platform Requirements

Id	Requirement
HLR.XX	When the ECOA Platform offers option PINFO WRITE, the maximal size of a PINFO shall be specified.

14.4 Link with Meta-Model

When option PINFO WRITE is available, attribute **writable** of pinfo items within a Component Type can be set to true.

When attribute writable of a pinfo item is set to true, it means that an instance of the related Component Type is allowed to access to the pinfo in writing mode.

The optional attribute **maxSize** (which is applicable for both read-only and writable pinfo items) allows to set the maximal authorized size (in bytes) of a pinfo. In case of a writable pinfo, it enables the platform to control the size of data to be written, and truncate them when necessary.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

ECOA Model consistency rules:

- An Application shall not contain more than one instance of a Component Type which defines a given writable pinfo.
No other Component instance shall refer to this writable pinfo.

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

15 OPTION AUTO START EXTERNAL TASK

Option AUTO START EXTERNAL TASK allows the external task of external components to start automatically.

When this option is not supported, external components must start their external task by calling the container API (cf. PART 4).

Note: This option exists mainly for historical reasons, and its use is discouraged. It does not bring any significant benefit, compared to the simpler solution of starting the external task by calling the container API.

There is no guarantee concerning the exact time when the external task is started, or the order between the start of the task and the initialization (or the start) of the component.

15.1 Mechanisms

When this option is supported, then, for an external component, if the option “autostartExternalThread” is set to “true” in the implementation model of this component, the ECOA platform shall start automatically the external task of this component.

The task is started when the executable containing the external component is started.

15.2 Software Interface

No impact.

15.3 Platform Requirements

Id	Requirement
HLR.XX	When the ECOA Platform offers option START EXTERNAL TASK, it shall take into account the option “autostartExternalThread” in implementation models.

15.4 Link with Meta-Model

The value “autostartExternalThread” in elements <option> of implementation models.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

16 OPTION COMM PORTS

The purpose of the option is to interface an ECOA application to other systems, using ECOA communication mechanisms (event, request-response, versioned data), without having to develop any manual code.

In particular, it allows offering ECOA-standardized communication between ECOA Applications, using exactly the same mechanisms that are defined for the communication between Components.

This includes:

- The ability to **define the interface of an application**, using the same description as the interface of a component (i.e. as a set of ECOA operations),
 - These operations are called *external operations*.
- The ability to **connect** component operations to external operations in the assembly model,
 - This connection is formalized through operation links, similarly to the internal assembly of a composite component, i.e. using a special instance named conventionally "extern" to represent the "outside world" of the application.
- The ability to **map** these operations on communication ports in the deployment model,
- The ability for the platform to **automatically manage** the application-level communications (sending and receiving messages and interpreting them as ECOA operations).

Said differently, the option COMM PORTS allows to view an application as a Composite that is able to communicate with the outside using ECOA mechanisms.

This option may be used:

- To communicate with other ECOA applications,
- To communicate with system or hardware entities that are not ECOA applications, using an arbitrary protocol, provided that the ECOA mechanisms have been mapped on to this protocol.

Option COMM PORTS does not specify any particular protocol. The protocol used may be completely custom and specific to an ECOA platform, or based on UDP and a description of the message contents, or based on a higher-level protocol such as HTTP, etc. (possible ECOA AS7 extensions).

An important principle of the COMM PORTS option is that the components **do not perceive any difference** between local communication (inside the application) and external communication (outside the application).

16.1 Mechanisms

1.1.1 Example

In this example, we have two applications, PING and PONG.

PING sends an event to PONG (through a communication port CH1), which sends it back to PING (through a communication port CH2).

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

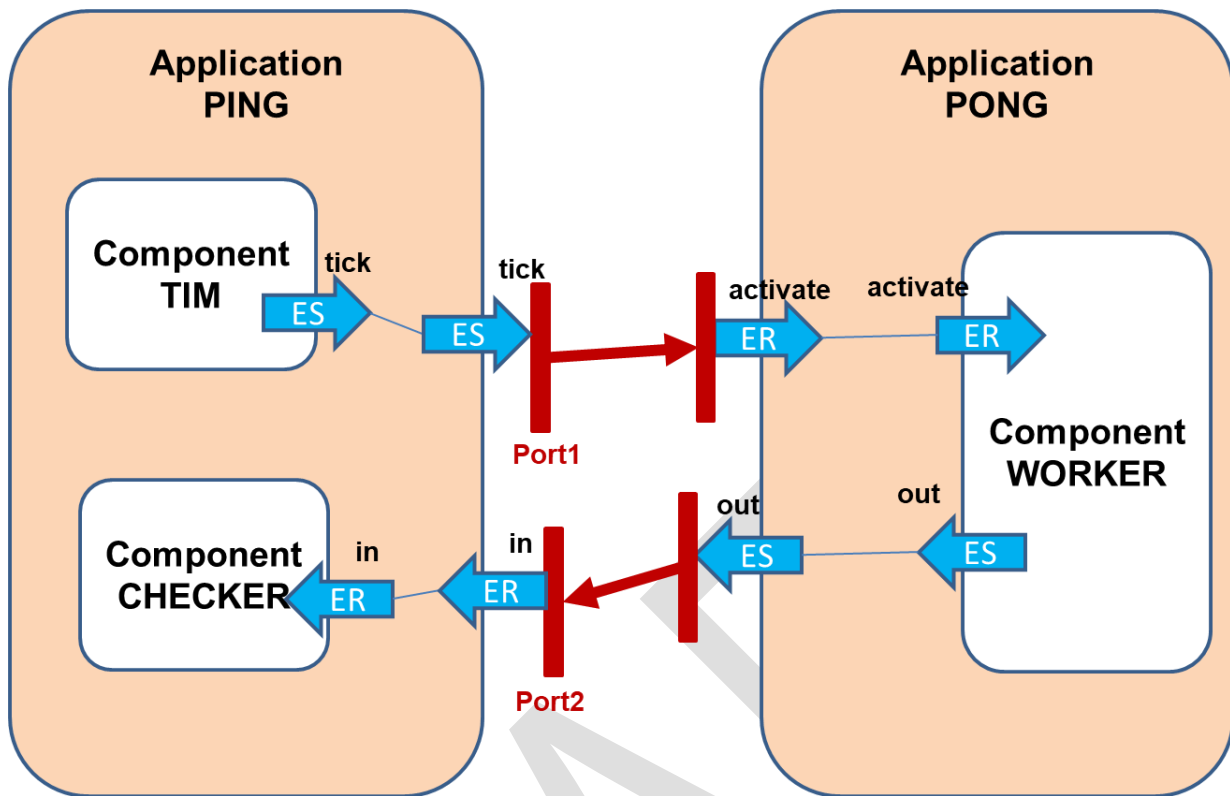


Figure 13 - Example of communication ports

In the following, we will focus on application PING, made of two components (TIM and CHECKER). First we have to create a new component type describing the interface of the application.

```
<!-- file: 01-Components/PING/PING.comp.xml -->
<componentType xmlns="http://www.ecoa.technology/ComponentType/3.0">
  <operations>
    <eventSent name="tick"/>
    <eventReceived name="in">
      <parameter name="p1" type="uint32" />
    </eventReceived>
  </operations>
</componentType>
```

The top-level assembly is (differences from an ordinary assembly are in bold):

```
<!-- file: 02-Assemblies/PING.assembly.xml -->
<assembly
  xmlns="http://www.ecoa.technology/Assembly/3.0"
  componentType="PING">

  <instance name="TIM" componentType="TIM" implementation="C"/>
  <instance name="CHECKER" componentType="CHECKER" implementation="C"/>

  <links>
    <eventLink>
      <sender instance="TIM" operation="tick" />
      <receiver instance="extern" operation="tick" />
      <receiver instance="CHECKER" operation="activate" />
    </eventLink>
  </links>
</assembly>
```

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

    </eventLink>
    <eventLink>
        <sender instance="extern" operation="in" />
        <receiver instance="CHECKER" operation="in" />
    </eventLink>
</links>
</assembly>

```

Note that this assembly is similar to the internal assembly of a composite: the same special instance name “**extern**” is used, to refer to “the outside world”.

The deployment is:

```

<application assembly="PING"
    xmlns="http://www.ecoa.technology/Deployment/3.0">
    ...
    <external_io>
        <inPort name="Port2">
            <operation name="in" id="0" />
        </inPort>
        <outPort name="Port1">
            <operation name="tick" id="2" />
        </outPort>
    </external_io>
</deployment>

```

This deployment has a section `<external_io>` containing elements `<inPort>` and `<outPort>`.

Each of these elements defines a communication port on which external operations can be received (`<inPort>`) or sent (`<outPort>`), or both (`<inOutPort>`). It has a logical name. Other properties, such as (for example) an IP address and a port number, shall be defined at platform-level configuration.

Each port contains `<operation>` elements, which are used to map an external operation on this port, with a given identifier to identify this operation in this port. In this example, we suppose that the protocol used requires to identify each operation by a number.

16.1.1 Types of ports

This option defines three kinds of communication ports:

- **Input-only** ports (`inPort` in deployment models): these ports are one-way, they can only receive information from outside the application.
- **Output-only** ports (`outPort` in deployment models): these ports are one-way, they can only send information to outside the application.
- **Input-Output** ports (`inOutPort` in deployment models): these ports are two-way, they can both send and receive information to/from outside the application.

Port type	Allowed operation types
Input-only	Event_received
	Data_read
Output-only	Event_sent

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

	Data_written
Input-Output	Event_received, Event_sent Data_read, Data_written Request_sent, Request_received

16.1.2 Adaptation of communication mechanisms

Globally, the three mechanisms defined in the ECOA standard for communication can be used in option COMM PORTS: event, request-response, versioned data.

However, between applications (contrary to inside an application), it is mandatory that all communication is done through a network-type mechanism.

Note: “Network-type mechanism” means that, from the application’s perspective, the mechanism looks like a network, i.e. explicitly sending and receiving pieces of information. For example, the well-known “socket” API. This does not necessary mean that a real network (such as Ethernet) is used. The network can be simulated, for example between containers or virtual machines running on the same physical machine.

16.1.2.1 Event mechanism

The ‘event’ mechanism is conceptually close to sending and receiving messages, so there is no difference between internal and external communication.

- *fifoSize* attributes are applicable.
- *Activating* attributes are applicable.

16.1.2.2 Request-Response mechanism

For a request-response operation mapped on a communication port:

- The request may be synchronous or asynchronous.
- The response may be immediate or deferred.
- *fifoSize* attributes are applicable.
- *Activating* attributes are applicable.
- *maxConcurrentRequests* attributes are applicable.
- Some constraints on multiplicity apply. See section “Multiple mapping of an external operation” below.

Note: The use of synchronous request-responses in communication ports is not recommended, as they imply a “blocking” behaviour, and thus, a higher coupling. Low coupling is suitable between applications. Even asynchronous request-responses imply a higher coupling than other mechanisms.

16.1.2.3 Versioned Data mechanism

For a data operation mapped on a communication port:

- A “dataWritten” external operation is always considered “write only”, i.e. the information flow is unidirectional (outgoing). In the component type defining the interface of the systems, all elements <dataWritten> must have the attribute: writeOnly=“true”, else the platform shall raise an error. Note that this forced “writeOnly” only allies to the “extern” pseudo-instance; it does not prevent real

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an ‘as is’ basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

component instances to have writeOnly set so "false" if it needs to read again the local copy of a data that is sent onto a communication port.

- The use of the "notification" mechanism is allowed, on the "reader" side of a data operation.
- *Activating* attributes are applicable (for the notification events).
- The 'uncontrolled' attribute is not significant. It is assumed to be 'false', because the information is physically copied in any case.
- The publish operation may be blocking and take some time to execute (depending on the size of the data), as data must be sent physically to the network.

16.1.3 Mapping of an external operation on communication ports

Regarding a given application and a given operation defined in the component type associated to this application:

- A purely outgoing operation (event or data) may be mapped on several ports of type "Output-only" or "Input-Output". The information will be sent through each port.
- A purely incoming operation (event or data) may be mapped on several ports of type "Input-only" or "Input-Output". The information will be expected from any port.
- A "request sent" external operation shall be mapped on exactly one Input-Output port.
- A "request received" external operation may be mapped on several pairs of Input-Output ports. In the assembly, the external operation shall be used in only one request-response link.
- If an external operation is not mapped to any port, it will never be sent nor received.

16.1.4 Operation IDs

Identifiers used in external ports for external operations can optionally be defined by the user, in the <external_io> section of the deployment model. An operation ID is given by the optional attribute "id" on elements "operation".

Example:

```
<operation name="in" id="101" />
```

For the ECOA metamodel, the value of an ID is an arbitrary string. Depending on the protocol, the ID may be, for example: a number (with constraints such as min/max), a name, a URI or URL, etc.

The protocol may impose additional constraints. For example, for "request received" operations, the ID of the request must be equal to the ID of the corresponding response. This may be needed by the platform to associate requests and responses.

The scope of unicity of the ID depends on the platform and the protocol: it may be unique in the port, in all ports sharing the same protocol, etc.

16.1.5 Protocol IDs

The protocol used by external operations can optionally be defined by the user, in the <external_io> section of the deployment model, for each port.

Example:

```
<inPort name="Port2" protocol_id="ELI_UDP">
```

For the ECOA metamodel, the value of "protocol_id" is an arbitrary string.

The list of available protocols is not defined in the standard, and may be platform-dependent.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

When ports of several applications are connected together, it is recommended to check that the same "protocol_id" is defined for all of them.

16.1.6 Network Endianness

If the application has external operations, the Endianness of external communications used can be specified in the deployment model, using the attribute 'endianness' set to 'BIG' or 'LITTLE'. By default, if no attribute is specified, then endianness preference is set to 'BIG'. This means that all data exchanged onto Network are processed under Big Endian format and then byte-swapped according to the destination platform endianness.

For optimal performance, it is recommended to select a network endianness that matches the endianness of the processor running the application.

An ECOA platform may support only endianness BIG, or only endianness LITTLE, or both.

An ECOA platform that supports BIG and LITTLE endianness may support or not support using different endiannesses in the same application.

The network endianness can be specified individually for each port.

Example:

```
<inPort name="Port2" endianness="BIG">
```

The network endianness can also be specified globally for all ports. This is the default endianness, which may be overridden by the port-specific endianness.

Example:

```
<application assembly="PING" endianness="BIG">
...
<external_io>
  <inPort name="Port1">                                <!-- BIG -->
  <inPort name="Port2" endianness="LITTLE"> <!--LITTLE -->
```

16.1.7 Conditional links

Conditional links are applicable to external operations in the same way as they are applicable to other operations.

16.1.8 "Shortcut" connections

In the top-level assembly, operation links connecting directly incoming and outgoing external operations together are not supported. An external operation cannot be connected directly to another external operation.

16.1.9 Independence between applications

The COMM PORT does not standardize any information exchange beyond the standard communication mechanisms between components (event, request-response, versioned data).

In particular, it does not specify anything about:

- The concept of "connection" between applications: is the other application connected? Is it "alive"?

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

- Supervision of components between applications: what are the other application's components? What is their state?
- The knowledge of the nature of what is at the other end of a communication channel. An application does not know if it communicates with another ECOA application, or something (software, hardware) that is not an application. It does not know if it shared the same ECOA platform or not.
- Same ECOA platform or different ECOA platforms.

All this is designed to preserve low coupling between applications.

16.2 Software Interface

Option [COMM PORTS] is transparent to application code.

The components do not know when they are communicating with other components, or when they are communicating with another application.

Therefore, there is no specific software interface for this option.

16.3 Platform Requirements

Id	Requirement
HLR.XX	When an ECOA Platform offers option COMM PORTS, the maximal size of the parameters, for each kind of operation, shall be indicated.
	When an ECOA Platform offers option COMM PORTS, it shall support at least one protocol. The list of protocols supported shall be indicated.
	When an ECOA Platform offers option COMM PORTS, the way to configure each communication port of an application shall be indicated (e.g. selection of the protocol, addresses, port numbers, operation_id, buffer sizes, etc.).
	When an ECOA Platform offers option COMM PORTS, it shall indicate what happens when an application receives a message with an unrecognized operation ID.
	When an ECOA Platform offers option COMM PORTS, if the endianness configuration for external operations is not supported, it shall fail at application build time, and indicate an error.

16.4 Link with Meta-Model

The following elements and attributes of the ECOA metamodel are significant only when option COMM PORTS is available (in xpath syntax).

In assembly models

- /assembly/@componentType
 - The 'componentType' references the ComponentType model (in 01-Components) that describes the interface of the whole application. This ComponentType model does not need to have an implementation.
- /assembly/links/*[@instance='extern']
 - The pseudo-instance 'extern' is allowed only if the 'componentType' attribute is set.
 - The operations referenced must exist in the ComponentType definition.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

In deployment models

- /application/external_io
 - This whole section is dedicated to the definition of communication ports, and the mapping of operations on communication ports.

DRAFT

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

17 OPTION CONDITIONAL LINKS (UNDER DEFINITION)

Option CONDITIONAL LINKS allows to **enable** or **disable** some connections defined in an assembly, dynamically during the execution of an application.

Option CONDITIONAL LINKS requires option SUPERVISION.

Some use cases for this option are:

- To define “modes” for an application. In each mode, the assembly is slightly different. The current changes dynamically at runtime. It is suitable to define a “complete” assembly (with all connections possible at any time), and for each mode, a subset of this complete assembly.
- To remove some conditional logic from the components, and put it in the assembly instead. The components are simplified and more reusable, because they are not aware of the conditions that define whether an information shall be used (or produced), or not.
- To have a single application that is equivalent to several distinct applications. Depending on a configuration that is known at startup time, the application may behave in one way or another, with differences in the way components are assembled.
- To maintain one assembly model instead of several nearly-identical assembly models.

17.1 Mechanisms

Conditional links are Operation Links, defined in an Assembly model, containing one (or more) <when> elements.

Example:

```
<eventLink >
  <sender instance="A" operation="X"/>
  <receiver instance="B" operation="X">
    <when instance="SuperComponent" variable="GUARD" value="0" />
  </receiver>
</eventLink>
```

A <when> element references:

- A component instance, which must be a SUPERVISOR component,
- The name of a supervision variable, which must be defined by this SUPERVISOR component,
- A value.

The value of a supervision variable is modifiable at run-time, at any moment, by the component that defines it.

The meaning of the <when> element is that the operation is active only when the supervision variable's value is strictly equal to the specified value.

Otherwise, it is inactive, and everything happens as if this connection was not defined in the assembly. More precisely:

- In case of an event operation, it is not received or sent,
- In case of a data operation, it is not read (no value will be returned in case of a read access) or written (the write access will succeed but the value will not be modified),
- In case of a request-response operation, the request will fail from the client's perspective, the server will not receive any request nor be able to send any response.

Every supervision variable always has a value. The initial value of a supervision variable is defined in the assembly model, through the element “variableInit”.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

The type of a supervision variable shall either be an int32, an uint32, or any simple or enumerated type based on these two types.

Example

File 01-Components/TESTER_SUPERVISOR/TESTER_SUPERVISOR.comp.xml

```
<componentType xmlns="http://www.ecoa.technology/ComponentType/3.0"
kind="SUPERVISOR">
  <variables>
    <variable type="int32" name="GUARD"></variable>
  </variables>
  <operations>
    <!-- ... -->
  </operations>
</componentType>
```

File 02-Assemblies/test.assembly.xml

```
<assembly xmlns="http://www.ecoa.technology/Assembly/3.0">
  <instance name="SuperComponent" componentType="TESTER_SUPERVISOR"
implementation="C">
    <variableInit name="GUARD" value="0" />
  </instance>
  <!--other instances... -->
  <links>
    <requestLink >
      <client operation="active_async_service_1"
instance="TestedComponent"/>
      <server operation="active_async_service"
instance="RunnerComponent">
        <when instance="SuperComponent" variable="GUARD"
value="0" />
      </server>
    </requestLink>
    <!-- ... -->
  </links>
</assembly>
```

17.2 Conditional Links SW Interface

This chapter complements ECOA Part 4 SW Interface document with the Conditional Links Option specific SW Interface.

17.2.1 Component Interface

N/A

17.2.2 Container Interface

"setter_variable" and "get_variable" functions to access supervision variables from supervisor components, refer to ECOA AS7 Part 4.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

17.3 Platform Requirements

Id	Requirement
HLR.XX	

17.4 Link with Meta-Model

The following elements and attributes of the ECOA metamodel are significant only when option CONDITIONAL LINKS is available (in xpath syntax).

In assembly models

- `/assembly/links/**/when`
 - The 'when' elements add logical conditions on a branch of an eventLink, dataLink or requestLink.
 - When several 'when' elements are defined, a logical "and" is performed between all conditions.

Without prejudice to the property rights relating to the ECOA AS6 Standard, the information in this document relating to the changes envisaged for the transition from the ECOA AS6 Standard to the ECOA AS7 Standard is the intellectual property of Dassault Aviation and Thales DMS France SAS. The information set out in this document is provided solely on an 'as is' basis and co-developers of this specification make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.