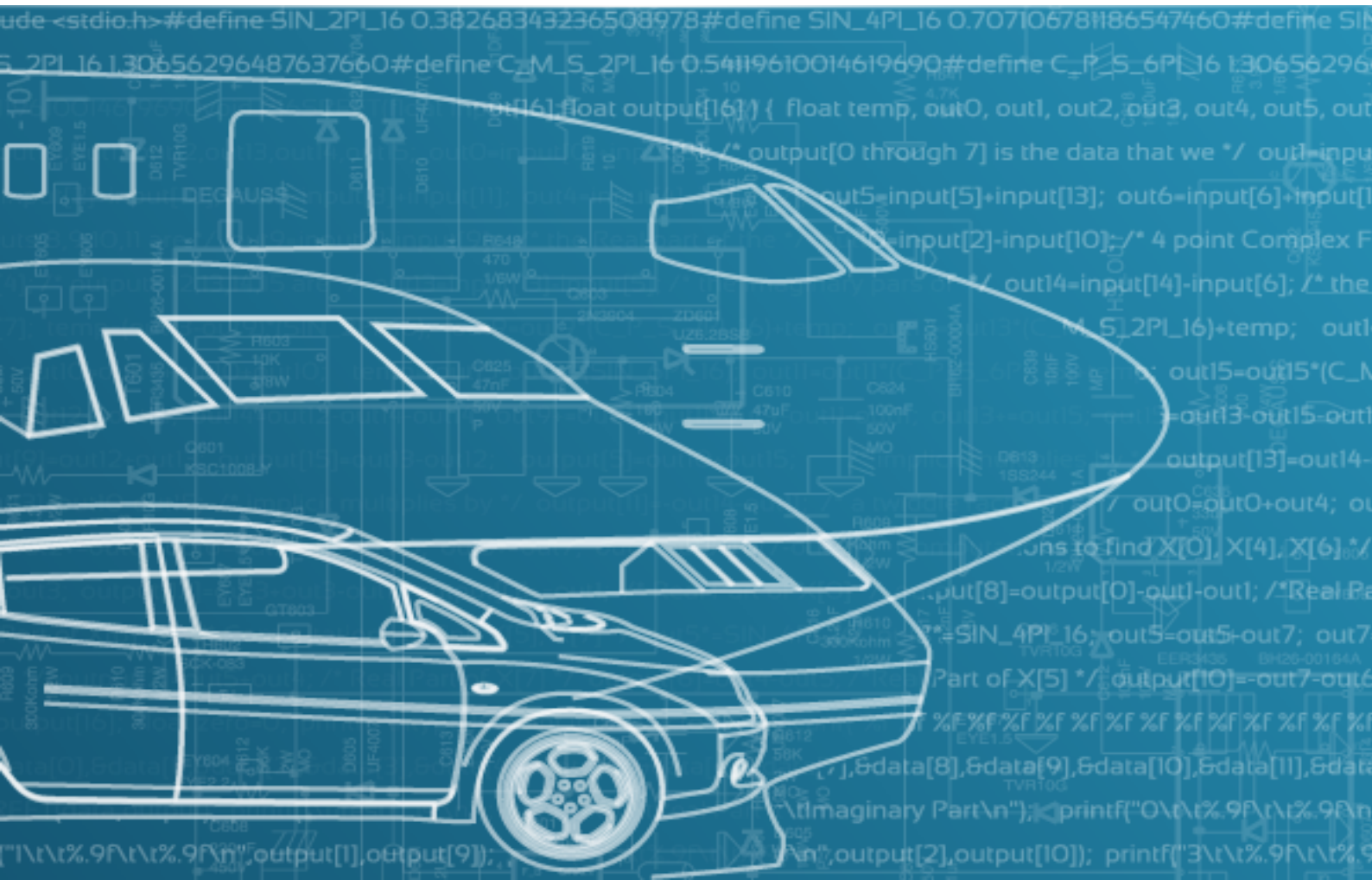
ASTERIOS

KRONO-SAFE

Safe design in real-time

Streams

Rev K19.5



HOW TO CONTACT US

Corporate headquarters	**KRONO-SAFE** Support 16 avenue Carnot, F - 91300 MASSY
Support contact	Customer Care Portal

VERSION

Document	Streams
Revision	K19.5
Date	May 16, 2024

COPYRIGHT

This work is **Copyright ©2012 - 2024 KRONO-SAFE**. All rights reserved. No part of this work may be reproduced in any form or by any means except as authorized by contract or other written permission. Because KRONO-SAFE is committed to continuous product improvement this manual is for informational purposes only and is subject to change without notice. While every attempt has been made to keep the information in this manual as accurate and as current as possible, KRONO-SAFE makes no warranty, expressed or implied, with regard to the information contained herein, including but not limited to the implied warranties of merchantability and fitness for any particular application. KRONO-SAFE assumes no responsibility for any errors that may appear within this document or for any other damages, direct or indirect, that may result from using this document or the products to which it relates.

Contents

This example introduces Streams and how they can be used to communicate between PsyC Tasks (Agents and Jobs). **Keywords:** `stream`, `pushto`, `popfrom`, `push`, `pop`, `count`

1 Content of the Application

The application built in this example simulates the merge of information to produce an output:

- the Agent `ag_reader_part0` in `process0.psy` sends piece by piece the first half of a message to be printed,
- the Agent `ag_reader_part1` in `process1.psy` sends piece by piece the second half of the message,
- the `input` Job `IO_displayer` in `IO.psy` gathers all the pieces and displays the complete message when all the pieces have been received.

2 A Short Introduction to Streams

The use of Streams in PsyC is detailed in the *PsyC Language Description* in the section *Communication Means*.

Streams are able to send data on demand from one Task to another according to the FIFO principle.

2.1 Visibility Principle

An Agent can pop a message if and only if this message has been made visible at a date preceding its current Earliest Start Date. This date from which items can be popped is named the *visibility horizon* of the Agent.

When an Agent pushes messages into a Stream, the messages are made visible at the current Deadline of the pusher. This date at which messages are made visible is named the *publication horizon* of the pusher.

Since Jobs are executed “on” a tick, their *visibility horizon* or their *publication horizon* is not defined the same way:

- The *publication horizon* of the `output` Jobs is the date of the tick “on” which they are executed.
- The *visibility horizon* of the `input` Jobs is the date of the tick “on” which they are executed. This is illustrated by the date at which the messages are popped by the Job `IO_displayer`: they are popped in the Job executed on the tick 52 of the Clock `ast_realtime_ms` and one of the popped message is produced by the Agent `ag_reader_part0` in an Elementary Action which Deadline is 52 (its *visibility date* is therefore 52).

2.2 Order in which Messages Are Popped

Messages can be popped from a Stream:

- by specifying the pusher which messages must be popped, for instance:

```
pop(output_fifo, taskid(ag_reader_part0))
```

- by popping indiscriminately the messages of all the pushers:

```
pop(output_fifo)
```

In the first case, it is enough to say that the FIFO principle is respected: the messages pushed first by the selected pusher are popped first.

In the second case, the following rules apply:

- Pop first *all* the visible messages of a pusher, respecting the order in which pushers are mentioned in the *authorized pushers list* (i.e. starting to pop the messages of the first pusher mentioned in this list).
- The visible messages of a pusher are popped respecting the FIFO principle.

The *authorized pushers list* is the list of Tasks declared in the `popfrom` block of the popping Task. In `IO.psy`, the Job `displayer` declares the following list:

```
popfrom output_fifo: ag_reader_part0, ag_reader_part1;
```

Therefore, when popping indiscriminately the messages, the messages of `ag_reader_part0` are popped before the messages of `ag_reader_part1`.

Note: The keyword `all` can replace the *authorized pushers list* in a `popfrom` block. This make the code more adaptable but less strongly specified.

In this case, if the order of the popped messages does matter in the Application, it is strongly discouraged to use the `pop` instruction without specifying the pusher. A default order between the pushers exists but it can be considered as a bad practice to rely on it. This default order is the lexicographical order on the name of the pushers (the name of the Jobs being prefixed by the name of their Worker).
