## Can Driver

A computer screen shot of a computer

Description automatically generated

### Giới thiệu và Tổng quan chức năng

Tài liệu này xác định chức năng, API và cấu hình của mô-đun CAN Driver trong AUTOSAR Basic Software (gọi là "mô-đun Can" trong tài liệu này).

Mô-đun Can là một phần của tầng thấp nhất, thực hiện việc truy cập phần cứng và cung cấp một API độc lập với phần cứng cho tầng trên.

Tầng trên duy nhất có quyền truy cập vào mô-đun Can là mô-đun CanIf (xem thêm SRS\_SPAL\_12092).

Mô-đun Can cung cấp dịch vụ để khởi tạo truyền và gọi hàm gọi lại của mô-đun CanIf để thông báo sự kiện, độc lập với phần cứng. Hơn nữa, nó cung cấp dịch vụ để kiểm soát hành vi và trạng thái của CAN controllers thuộc cùng một CAN Hardware Unit.

Nhiều CAN controllers có thể được kiểm soát bởi một mô-đun Can duy nhất miễn là chúng thuộc cùng một CAN Hardware Unit

Để có mô tả chi tiết hơn về CAN controller và CAN Hardware Unit, xem chương Viết tắt và biểu đồ trong [5].

### Từ viết tắt và viết tắt

|  |  |
| --- | --- |
| **Thuật ngữ** | **Mô tả** |
| CAN Controller | CAN Controller phục vụ chính xác một kênh vật lý. |
| CAN Hardware Unit | CAN Hardware Unit có thể bao gồm một hoặc nhiều CAN Controllers cùng loại và một hoặc nhiều vùng CAN RAM. CAN Hardware Unit nằm trên chip hoặc thiết bị bên ngoài. CAN Hardware Unit được đại diện bởi một CAN Driver. |
| CAN L-PDU | Data Link Layer Protocol Data Unit. Bao gồm Mã định danh (ID), Độ dài dữ liệu và Dữ liệu (SDU). (xem [18]) |
| CAN L-SDU | Data Link Layer Service Data Unit. Dữ liệu được vận chuyển bên trong L-PDU. (xem [18]) |
| DLC | Data Length Code (một phần của Message CAN mô tả độ dài SDU) |
| Hardware Object | Một CAN Hardware Object được định nghĩa là bộ đệm PDU bên trong CAN RAM của CAN Hardware Unit/ CAN Controller. Một CAN Hardware Object được định nghĩa là bộ đệm L-PDU bên trong RAM CAN của CAN Hardware Unit. |
| Hardware Receive Handle (HRH) | Hardware Receive Handle (HRH) được xác định và cung cấp bởi CAN Driver. Mỗi HRH thường chỉ đại diện cho một Hardware Object. HRH có thể được sử dụng để tối ưu hóa Software Filtering |
| Hardware Transmit Handle (HTH) | Hardware Transmit Handle (HTH) được xác định và cung cấp bởi CAN Driver. Mỗi HTH thường chỉ đại diện cho một hoặc nhiều Hardware Object được cấu hình như Hardware Transmit Buffer |
| Inner Priority Inversion | Việc truyền L-PDU có mức ưu tiên cao bị ngăn chặn bởi sự hiện diện của L-PDU có mức ưu tiên thấp đang chờ xử lý trong cùng một Hardware Object. |
| ISR | Interrupt Service Routine |
| L-PDU Handle | L-PDU handle được xác định và đặt bên trong lớp mô-đun CanIf. Thông thường, mỗi bộ điều khiển đại diện cho một L-PDU, là một cấu trúc cố định chứa thông tin dành cho quá trình xử lý Tx/Rx. |
| MCAL | Microcontroller Abstraction Layer |
| Outer Priority Inversion | Khoảng cách thời gian xảy ra giữa hai lần truyền L-PDU liên tiếp. Trong trường hợp này, L-PDU có mức ưu tiên thấp hơn từ nút khác có thể ngăn việc gửi L-PDU có mức ưu tiên cao hơn của chính nó. Ở đây, L-PDU có mức ưu tiên cao hơn không thể tham gia phân xử trong quá trình truy cập mạng vì L-PDU có mức ưu tiên thấp hơn đã thắng trong quá trình phân xử. |
| Physical Channel | Kênh vật lý đại diện cho giao diện từ CAN Controller đến CAN Network. Các kênh vật lý khác nhau của CAN Hardware Unit có thể truy cập các mạng khác nhau. |
| Priority | Mức độ ưu tiên của CAN L-PDU được thể hiện bằng CAN Identifier. Giá trị số của mã định danh càng thấp thì mức độ ưu tiên càng cao. |
| SFR | Special Function Register. Thanh ghi phần cứng điều khiển hành vi của bộ điều khiển. |
| SPAL | Standard Peripheral Abstraction Layer |
| ICOM | Intelligent Communication Controller |

#### Priority Inversion - Đảo ngược ưu tiên

A diagram of a data link

Description automatically generated

"Nếu chỉ có một bộ đệm truyền đơn được sử dụng, có thể xảy ra sự đảo ngược ưu tiên nội bộ. Do ưu tiên thấp, một thông điệp được lưu trữ trong bộ đệm phải chờ đợi cho đến khi 'lưu lượng trên bus ổn định'. Trong thời gian chờ đợi này, thông điệp này có thể ngăn cản một thông điệp có ưu tiên cao hơn được tạo ra bởi cùng một vi điều khiển từ việc được truyền qua bus."

A data link system

Description automatically generated with medium confidence

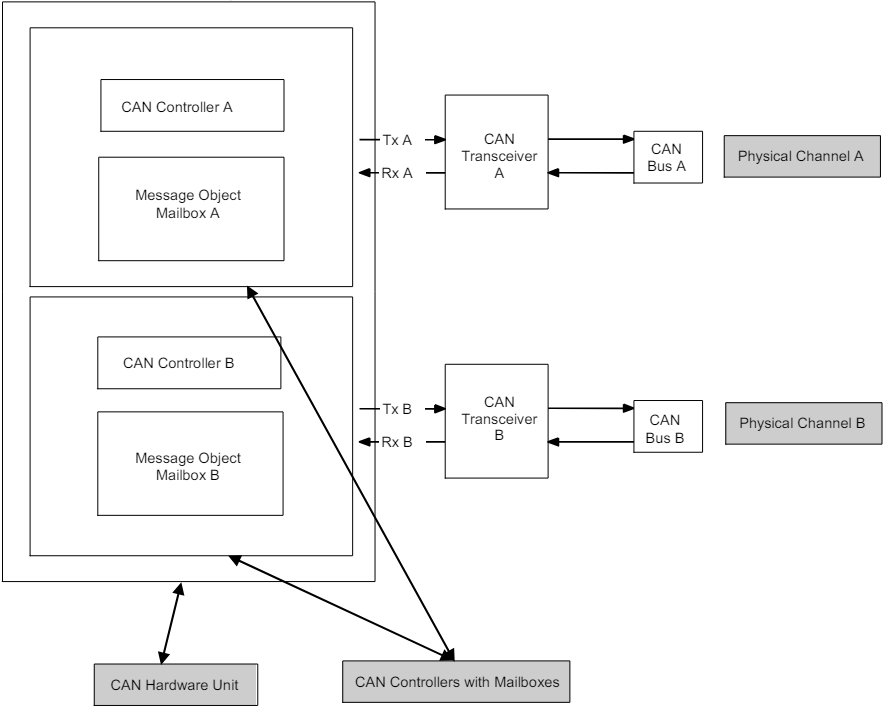
"Vấn đề của sự đảo ngược ưu tiên bên ngoài có thể xảy ra trong một số cài đặt CAN. Hãy giả sử rằng một nút CAN muốn truyền một gói tin của các thông điệp liên tiếp với ưu tiên cao, được lưu trữ trong các bộ đệm thông điệp khác nhau. Nếu khoảng thời gian giữa các thông điệp này trên mạng CAN dài hơn so với khoảng trống tối thiểu được xác định bởi tiêu chuẩn CAN, một nút thứ hai có thể bắt đầu truyền một thông điệp ưu tiên thấp hơn. Khoảng trống tối thiểu giữa các khung tin trên mạng CAN được xác định bởi trường Intermission, bao gồm 3 bit không quyết định. Một thông điệp, đợi trong quá trình truyền của một thông điệp khác, được bắt đầu trong khoảng thời gian Bus Idle, sớm nhất là trong bit tiếp theo sau trường Intermission. Ngoại lệ là một nút có thông điệp truyền đợi sẽ giải thích một bit động ở bit thứ ba của Intermission như một bit Start-of-Frame và bắt đầu truyền với bit nhận dạng đầu tiên mà không cần truyền trước một bit SOF. Thời gian xử lý nội bộ của một mô-đun CAN phải đủ ngắn để gửi các thông điệp liên tiếp với khoảng trống tối thiểu giữa các khung để tránh sự đảo ngược ưu tiên bên ngoài dưới tất cả các tình huống đã đề cập."

#### CAN Hardware Unit

CAN Hardware Unit kết hợp một hoặc nhiều CAN Controllers, có thể được đặt trên chip hoặc là các thiết bị độc lập bên ngoài cùng loại, với

Hardware Objects chung hoặc riêng lẻ.

Hình dưới đây thể hiện một CAN Hardware Unit bao gồm hai CAN controllers kết nối với hai Kênh Vật lý:



### Related documentation

### Constraints and assumptions

#### Limitations

Một CAN Controller luôn tương ứng với một kênh vật lý. Nó được phép kết nối các kênh vật lý ở phía bus. Mô-đun CanIf sẽ xử lý bất kể các CAN Controllers liên quan một cách riêng biệt.

Một số ít CAN Hardware Units hỗ trợ khả năng kết hợp nhiều CAN Controllers bằng cách sử dụng CAN RAM, để mở rộng số lượng đối tượng thông điệp cho một CAN Controller. Các bộ điều khiển CAN kết hợp này được xử lý như một bộ điều khiển bởi mô-đun Can.

Mô-đun Can không hỗ trợ các CAN remote frames.

[**SWS\_Can\_00237**] ⌈ Mô-đun Can không được phép truyền các thông điệp được kích hoạt bởi Remote Transmission Requests.⌋ (SRS\_Can\_01147)

[**SWS\_Can\_00236**] ⌈ Mô-đun Can phải khởi tạo CAN HW để bỏ qua bất kỳ Remote Transmission Requests.⌋ (SRS\_Can\_01147)

#### Ứng dụng trong các lĩnh vực xe hơi

Mô-đun Can có thể được sử dụng cho bất kỳ ứng dụng nào, khi giao thức CAN được sử dụng.

### Dependencies to other modules

#### Static Configuration

Các phần tử cấu hình mô tả trong chương 10 có thể được các mô-đun BSW khác tham chiếu đến cho cấu hình của chúng.

#### Driver Services

[**SWS\_Can\_00238**] ⌈ Nếu CAN Controller nằm trên chip, mô-đun Can không được phép sử dụng bất kỳ dịch vụ của các trình điều khiển khác.⌋ (SRS\_BSW\_00005)

[**SWS\_Can\_00239**] ⌈ Hàm Can\_Init phải khởi tạo tất cả các tài nguyên phần cứng trên chip được sử dụng bởi CAN Controller. Ngoại lệ duy nhất cho điều này là cấu hình chân I/O kỹ thuật số (của các chân được sử dụng bởi CAN), được thực hiện bởi trình điều khiển cổng.⌋ (SRS\_BSW\_00377)

[**SWS\_Can\_00240**] ⌈ Mô-đun Mcu (SPAL xem [8]) phải cấu hình các thiết lập của bộ register được 'chia sẻ' với các mô-đun khác.⌋ ()

Mẹo triển khai: Mô-đun Mcu phải được khởi tạo trước khi khởi tạo mô-đun Can.

[**SWS\_Can\_00242**] ⌈ Nếu CAN Controller sử dụng, mô-đun Can phải sử dụng các dịch vụ của các trình điều khiển MCAL khác (ví dụ: SPI).⌋ (SRS\_BSW\_00005)

Mẹo triển khai: Nếu mô-đun Can sử dụng các dịch vụ của các trình điều khiển MCAL khác (ví dụ: SPI), phải đảm bảo rằng các trình điều khiển này đang hoạt động trước khi khởi tạo mô-đun Can. Thứ tự khởi tạo của các trình điều khiển khác nhau được chỉ định một phần trong [7].

[**SWS\_Can\_00244**] ⌈ Mô-đun Can phải sử dụng các API đồng bộ của các trình điều khiển MCAL cơ sở và không được cung cấp các hàm gọi lại có thể được gọi bởi các trình điều khiển MCAL.⌋ ()

Do đó, loại kết nối giữa vi điều khiển và CAN HW Units chỉ ảnh hưởng đến triển khai và không ảnh hưởng đến API.

#### System Services

[**SWS\_Can\_00280**] ⌈ Trong các trường hợp phần cứng đặc biệt, mô-đun Can phải đánh giá sự kiện của phần cứng.⌋ ()

[**SWS\_Can\_00281**] ⌈ Mô-đun Can phải sử dụng OsCounter được cung cấp bởi dịch vụ hệ thống để phát hiện timeout trong trường hợp phần cứng không phản ứng trong thời gian mong đợi (sự cố phần cứng) để ngăn chặn vòng lặp vô hạn.⌋ ()

Mẹo triển khai: Thời gian chặn của hàm mô-đun Can đang chờ phản ứng của phần cứng phải ngắn hơn chu kỳ kích hoạt của hàm chính CAN (tức là Can\_MainFunction\_Read), vì các chức năng chính CAN không thể được sử dụng cho mục đích đó.

#### Can module Users

[**SWS\_Can\_00058**] ⌈ Mô-đun Can tương tác với các mô-đun khác (ví dụ: Tracer Lỗi Mặc định (DET), Quản lý Trạng thái Ecu (ECUM)) với mô-đun CanIf một cách trực tiếp. Tài liệu này không bao giờ xác định nguồn gốc thực sự của một yêu cầu hoặc điểm đến thực sự của một thông báo. Trình điều khiển chỉ nhìn thấy mô-đun CanIf như là nguồn gốc và điểm đến.⌋ (SRS\_SPAL\_12092)

#### File structure

[**SWS\_Can\_00436**] ⌈ Can\_GeneralTypes.h phải chứa tất cả các loại và hằng số được chia sẻ giữa các mô-đun CAN AUTOSAR Can, CanIf và CanTrcv.⌋ ()

### Requirements traceability

### Functional specification

Khi truyền L-PDU, mô-đun Can sẽ ghi L-PDU vào một bộ đệm thích hợp bên trong CAN Controller Hardware. [***Xem chương 7.5 để biết mô tả chi tiết về việc truyền L-PDU***](#_Truyền_L-PDU)***.***

Khi nhận L-PDU, mô-đun Can sẽ gọi hàm callback chỉ báo RX với các tham số là ID, Độ dài Dữ liệu và con trỏ đến L-SDU. [***Xem chương 7.6 để biết mô tả chi tiết về việc nhận L-PDU***](#_Nhận_L-PDU)***.***

Mô-đun Can cung cấp một giao diện hoạt động như một hàm xử lý định kỳ, và phải được gọi bởi mô-đun Basic Software Scheduler theo chu kỳ.

Hơn nữa, mô-đun Can cung cấp các dịch vụ để kiểm soát trạng thái của các CAN Controllers. Các sự kiện Bus-off và Wake-up được thông báo thông qua các hàm callback.

Mô-đun Can là một Basic Software Module truy cập các tài nguyên phần cứng. Do đó, nó được thiết kế để đáp ứng các yêu cầu đối với các Basic Software Modules được quy định trong AUTOSAR\_SRS\_SPAL (xem [3]).

[**SWS\_Can\_00033**] ⌈ Mô-đun Can phải triển khai các hàm dịch vụ ngắt cho tất cả các CAN Hardware Unit interrupts cần thiết. ⌋ (SRS\_BSW\_00164, SRS\_SPAL\_12129)

[**SWS\_Can\_00419**] ⌈ Mô-đun Can phải vô hiệu hóa tất cả các ngắt không được sử dụng trong CAN Controller. ⌋ ()

[**SWS\_Can\_00420**] ⌈ Mô-đun Can phải đặt lại cờ ngắt vào cuối hàm ISR (nếu không được tự động thực hiện bởi phần cứng). ⌋ ()

Gợi ý thực hiện: Mô-đun Can không được thiết lập cấu hình (ví dụ: ưu tiên) của bảng vectơ ngắt.

[**SWS\_Can\_00079**] ⌈ Mô-đun Can phải tuân thủ tất cả các hướng dẫn thiết kế và thực hiện được mô tả trong [2]. ⌋ (SRS\_BSW\_00007, SRS\_BSW\_00306, SRS\_BSW\_00308, SRS\_BSW\_00309, SRS\_BSW\_00330)

#### Driver scope

Một mô-đun Can cung cấp truy cập vào một CAN Hardware Unit có thể bao gồm nhiều CAN Controllers.

[**SWS\_Can\_00077**] ⌈ Đối với CAN Hardware Units của loại khác nhau, phải triển khai các mô-đun Can khác nhau. ⌋ (SRS\_BSW\_00347)

[**SWS\_Can\_00284**] ⌈ Trong trường hợp nhiều CAN Hardware Units (của cùng hoặc khác nhà sản xuất) được triển khai trong một ECU, tên hàm và biến toàn cục của các mô-đun Can phải được triển khai sao cho không có hai hàm cùng tên được tạo ra. ⌋ ()

Cách đặt tên tuân thủ như sau:

<Can module name>\_<vendorID>\_<Vendor specific API name><driver abbreviation>()

SRS\_BSW\_00347 quy định cách đặt tên.

[**SWS\_Can\_00385**] ⌈ Cách đặt tên phải được sử dụng chỉ trong trường hợp, nếu cần hỗ trợ nhiều loại CAN Controller khác nhau trên một ECU. ⌋ ()

[**SWS\_Can\_00386**] ⌈ Nếu chỉ có một loại bộ điều khiển được sử dụng, các quy ước đặt tên ban đầu mà không có bất kỳ phần mở rộng viết tắt trình điều khiển nào cũng đủ.⌋ ()

Xem [5] để biết mô tả cách xử lý nhiều mô-đun Can bởi mô-đun CanIf.

#### Driver State Machine

Mô-đun Can có hai trạng thái CAN\_UNINIT và CAN\_READY. Hình 7.1 Show the state machine.

[**SWS\_Can\_00103**] ⌈ Sau khi mở nguồn/đặt lại, mô-đun Can phải ở trạng thái CAN\_UNINIT. ⌋ (SRS\_BSW\_00406)

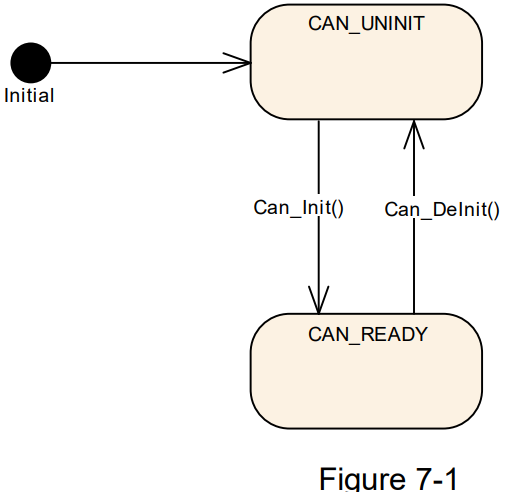


Figure7.1 Show the state machine.

[**SWS\_Can\_00246**] ⌈ Hàm **Can\_Init** phải thay đổi trạng thái của mô-đun thành CAN\_READY, sau khi khởi tạo tất cả các bộ điều khiển bên trong HW Unit.⌋ (SRS\_SPAL\_12057, SRS\_Can\_01041)

[**SWS\_Can\_00245**] ⌈ Hàm **Can\_Init** phải khởi tạo tất cả các CAN Controller theo cấu hình của chúng.⌋ (SRS\_SPAL\_12057, SRS\_Can\_01041)

Sau đó, mỗi CAN Controller phải được khởi động riêng lẻ bằng cách gọi hàm **Can\_SetControllerMode (CAN\_CS\_STARTED**)

Gợi ý thực hiện:

Các thiết lập thanh ghi phần cứng có ảnh hưởng đến tất cả các CAN Controller bên trong HW Unit chỉ có thể được thiết lập trong hàm Can\_Init.

Gợi ý thực hiện:

Mô-đun ECU State Manager phải gọi Can\_Init tối đa một lần trong quá trình chạy.

[**SWS\_Can\_91009**] ⌈ Hàm Can\_DeInit phải thay đổi trạng thái của mô-đun thành CAN\_UNINIT trước khi hủy khởi tạo tất cả các bộ điều khiển bên trong HW Unit⌋.⌋ (SRS\_Can\_01166)

Xem [SWS\_Can\_91010].

#### CAN Controller State Machine

Mỗi CAN Controlle có máy trạng thái phức tạp được triển khai trong phần cứng. Để đơn giản hóa, số lượng trạng thái được giảm xuống chỉ còn bốn trạng thái cơ bản sau đây trong mô tả này: UNINIT, STOPPED, STARTED và SLEEP.

Mọi truy cập phần cứng CAN được đóng gói bởi các hàm của mô-đun Can, nhưng mô-đun Can không ghi nhớ các thay đổi trạng thái.

Mô-đun Can cung cấp các dịch vụ **Can\_Init, Can\_SetBaudrate và Can\_SetControllerMode**. Các dịch vụ này thực hiện các thiết lập thanh ghi cần thiết gây ra sự thay đổi cần thiết của trạng thái bộ điều khiển phần cứng CAN.

Có hai cách để kích hoạt các thay đổi trạng thái bằng các sự kiện bên ngoài:

* Bus-off event
* HW wakeup event

Các sự kiện này được chỉ ra bằng một ngắt hoặc bằng một bit trạng thái được đặt trong hàm **Can\_MainFunction\_BusOff** hoặc **Can\_MainFunction\_Wakeup.**

Mô-đun Can thực hiện các thiết lập thanh ghi cần thiết để thực hiện hành vi yêu cầu (tức là không có khôi phục phần cứng trong trường hợp bus off).

Sau đó, nó thông báo cho mô-đun CanIf với hàm gọi lại tương ứng. Trạng thái phần mềm sau đó được thay đổi bên trong hàm gọi lại này.

Trong trường hợp lỗi phát triển được kích hoạt và có một chuyển đổi không được phép được yêu cầu bởi lớp trên, mô-đun Can phải tạo ra lỗi phát triển CAN\_E\_TRANSITION.

Mô-đun Can không kiểm tra trạng thái thực tế trước khi thực hiện Can\_Write hoặc gửi các hàm gọi lại.

##### CAN Controller State Description

Chương này mô tả hành vi phần cứng cần thiết cho các trạng thái bộ điều khiển khác nhau.

**CAN controller state UNINIT**

CAN Controller không được khởi tạo. Tất cả các thanh ghi thuộc Module CAN đều ở trạng thái thiết lập lại, các ngắt CAN bị vô hiệu hóa. CAN Controller không tham gia vào bus CAN.

**CAN controller state STOPPED**

Trong trạng thái này, CAN Controller đã được khởi tạo nhưng không tham gia vào bus. Ngoài ra, các khung lỗi và xác nhận không được gửi đi.

(Ví dụ: Đối với nhiều bộ điều khiển, việc vào chế độ 'khởi tạo' khiến cho bộ điều khiển bị dừng lại.)

**CAN controller state STARTED**

Bộ điều khiển đang ở chế độ hoạt động bình thường với đầy đủ chức năng, điều này có nghĩa là nó tham gia vào mạng. Đối với nhiều bộ điều khiển, việc rời khỏi chế độ 'khởi tạo' khiến cho bộ điều khiển được bắt đầu.

**CAN controller state SLEEP**

Các thiết lập phần cứng chỉ khác biệt so với trạng thái STOPPED đối với phần cứng CAN hỗ trợ chế độ ngủ (Chế độ wake-up qua bus CAN được hỗ trợ trực tiếp bởi phần cứng CAN).

[SWS\_Can\_00257] ⌈ Khi phần cứng CAN hỗ trợ chế độ ngủ và được kích hoạt để chuyển sang trạng thái SLEEP, mô-đun Can phải đặt bộ điều khiển vào trạng thái SLEEP từ đó phần cứng có thể được đánh thức qua Bus CAN. ⌋ (SRS\_SPAL\_12067)

[SWS\_Can\_00258] ⌈ Khi phần cứng CAN không hỗ trợ chế độ ngủ và được kích hoạt để chuyển sang trạng thái SLEEP, mô-đun Can phải mô phỏng một trạng thái SLEEP logic từ đó chỉ trở về khi được kích hoạt bởi phần mềm để chuyển sang trạng thái STOPPED. ⌋ ()

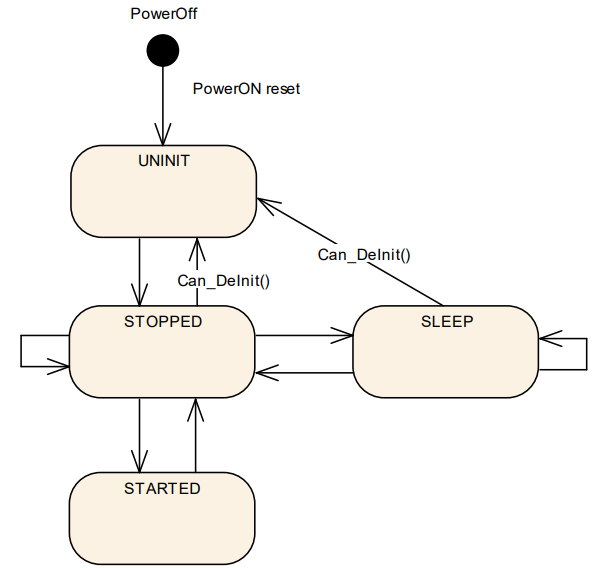
[SWS\_Can\_00404] ⌈ Phần cứng CAN phải duy trì ở trạng thái STOPPED trong khi trạng thái SLEEP logic đang hoạt động. ⌋ ()

##### CAN Controller State Transitions

Một chuyển đổi trạng thái được kích hoạt bởi phần mềm với hàm Can\_SetControllerMode với chuyển đổi cần thiết làm tham số. Một chuyển đổi trạng thái thành công được kích hoạt bởi phần mềm được thông báo bằng hàm gọi lại (CanIf\_ControllerModeIndication). Việc theo dõi xem trạng thái yêu cầu đã đạt được là phần của một mô-đun lớp trên và không phải là phần của mô-đun Can.

Một số chuyển đổi được kích hoạt bởi các sự kiện trên bus (hardware). Những chuyển đổi này gây ra một thông báo thông qua một hàm gọi lại (CanIf\_ControllerBusOff, EcuM\_CheckWakeup).

Hành vi cho các chuyển đổi không hợp lệ trong mã sản xuất không xác định. Hình 7-2 hiển thị tất cả các chuyển đổi trạng thái hợp lệ.



Hình 7-2: Shows all valid state transitions.

##### State transition caused by function Can\_Init

* UNINIT STOPPED (cho tất cả các bộ điều khiển trong HW Unit)
* Được kích hoạt bởi phần mềm thông qua gọi hàm Can\_Init.
* Thực hiện cấu hình cho tất cả các CAN Controllers bên trong HW Unit.

Tất cả các thanh ghi điều khiển được thiết lập theo cấu hình tĩnh.

[SWS\_Can\_00259] ⌈ Hàm Can\_Init phải đặt tất cả các CAN Controllers ở trạng thái STOPPED.⌋ ()

Khi hàm Can\_Init được gọi và mô-đun Can không ở trạng thái CAN\_UNINIT hoặc CAN Controllers không ở trạng thái UNINIT, nó phải tạo ra lỗi CAN\_E\_TRANSITION (So sánh với SWS\_Can\_00174 và SWS\_Can\_00408).

##### State transition caused by function Can\_SetBaudrate

* STOPPED STOPPED; SLEEP SLEEP; STARTED STARTED
* Được kích hoạt bởi phần mềm thông qua cuộc hàm Can\_SetBaudrate.
* Thay đổi cấu hình của CAN Controllers.

Các thanh ghi điều khiển của CAN Controllers được thiết lập theo cấu hình tĩnh.

[SWS\_Can\_00256] ⌈ Nếu việc gọi hàm Can\_SetBaudrate() gây ra một việc khởi tạo lại CAN Controllers và CAN Controllers không ở trạng thái STOPPED, nó phải trả về E\_NOT\_OK.⌋ ()

[SWS\_Can\_00260] ⌈ Nếu việc khởi tạo lại cần thiết, hàm Can\_SetBaudrate phải duy trì CAN Controllers ở trạng thái STOPPED.⌋ ()

[SWS\_Can\_00422] ⌈ Nếu việc khởi tạo lại cần thiết, hàm Can\_SetBaudrate phải đảm bảo rằng bất kỳ cài đặt nào sẽ không được thiết lập khi CAN Controllers tham gia vào mạng.⌋ ()

##### State transition caused by function Can\_SetControllerMode

Phần mềm có thể kích hoạt một chuyển đổi trạng thái của CAN Controllers với hàm Can\_SetControllerMode. Tùy thuộc vào phần cứng CAN, việc thay đổi cài đặt thanh ghi để chuyển sang một trạng thái mới của CAN Controllers có thể chỉ xảy ra sau một khoảng trễ. Mô-đun Can thông báo cho lớp trên (CanIf\_ControllerModeIndication) sau một chuyển đổi trạng thái thành công về trạng thái mới. Việc theo dõi xem trạng thái yêu cầu đã đạt được là phần của một mô-đun lớp trên và không phải là phần của mô-đun Can.

[SWS\_Can\_00370] ⌈ Hàm Can\_Mainfunction\_Mode phải kiểm tra 1 flag của thanh ghi trạng thái CAN cho đến khi cờ báo hiệu rằng thay đổi có hiệu lực và thông báo cho lớp trên với hàm CanIf\_ControllerModeIndication về một chuyển đổi trạng thái thành công với thông tin về CAN Controllers tương ứng với abstract CanIf ControllerId.⌋ ()

[SWS\_Can\_00398] ⌈ Hàm Can\_SetControllerMode phải sử dụng hệ thống dịch vụ GetCounterValue để theo dõi thời gian chờ để tránh chặn các hàm.⌋ ()

[SWS\_Can\_00372] ⌈ Trong trường hợp cờ báo hiệu rằng thay đổi không có hiệu lực và thời gian tối đa CanTimeoutDuration đã trôi qua, hàm Can\_SetControllerMode phải dừng và hàm Can\_Mainfunction\_Mode phải tiếp tục kiểm tra cờ báo hiệu.⌋ (SRS\_SPAL\_12077)

[SWS\_Can\_00373] ⌈ Hàm Can\_Mainfunction\_Mode phải gọi hàm CanIf\_ControllerModeIndication để thông báo cho lớp trên về một chuyển đổi trạng thái thành công của CAN Controllers tương ứng được chỉ định bởi abstract CanIf ControllerId, trong trường hợp chuyển đổi trạng thái được kích hoạt bởi hàm Can\_SetControllerMode.⌋ ()

**Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_STARTED)**

* STOPPED STARTED
* Được kích hoạt bởi phần mềm.

[SWS\_Can\_00261] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STARTED) phải thiết lập các thanh ghi phần cứng một cách sao cho CAN Controllers tham gia vào mạng.⌋ ()

[SWS\_Can\_00262] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STARTED) phải chờ một khoảng thời gian giới hạn cho đến khi CAN Controllers hoạt động hoàn toàn. So sánh với SWS\_Can\_00398.⌋ ()

Các yêu cầu truyền gửi được khởi tạo trước khi CAN Controllers hoạt động sẽ bị mất. Duy nhất chỉ có việc tiếp nhận các xác nhận truyền hoặc chỉ định nhận. Các thực thể gửi có thể nhận thấy một sự chậm trễ xác nhận và cần phải xử lý được điều đó.

[SWS\_Can\_00409] ⌈ Khi hàm Can\_SetControllerMode (CAN\_CS\_STARTED) được gọi và CAN Controllers không ở trạng thái STOPPED, nó phải phát hiện ra một chuyển đổi trạng thái không hợp lệ (So sánh với SWS\_Can\_00200).⌋ ()

**Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_STOPPED)**

* STARTED STOPPED
* SLEEP STOPPED
* Được kích hoạt bởi phần mềm

[SWS\_Can\_00263] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải thiết lập các bit bên trong phần cứng CAN sao cho CAN Controllers ngừng tham gia vào mạng.⌋ ()

[SWS\_Can\_00264] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải chờ một khoảng thời gian giới hạn cho đến khi CAN Controllers thực sự được tắt. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00267] ⌈ Nếu phần cứng CAN không hỗ trợ chế độ ngủ, việc chuyển từ SLEEP sang STOPPED phải trở lại từ trạng thái ngủ logic, nhưng không có hiệu lực đối với trạng thái của CAN Controllers (vì bộ điều khiển đã ở trạng thái dừng).⌋ ()

[SWS\_Can\_00268] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải chờ một khoảng thời gian giới hạn cho đến khi CAN Controllers ở trạng thái STOPPED. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00282] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải hủy bỏ các tin nhắn đang chờ. ⌋ ()

**Chuyển đổi trạng thái do hàm Can\_SetControllerMode (CAN\_CS\_SLEEP)**

* STOPPED SLEEP
* Được kích hoạt bởi phần mềm

[SWS\_Can\_00265] ⌈ Hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) phải đặt bộ điều khiển vào chế độ ngủ.⌋ ()

[SWS\_Can\_00266] ⌈ Nếu phần cứng CAN hỗ trợ chế độ ngủ, hàm Can\_SetControllerMode (CAN\_CS\_SLEEP) phải chờ một khoảng thời gian giới hạn cho đến khi CAN Controllers ở trạng thái NGỦ và đảm bảo rằng phần cứng CAN có thể đánh thức được. So sánh với SWS\_Can\_00398.⌋ ()

[SWS\_Can\_00290] ⌈ Nếu phần cứng CAN không hỗ trợ chế độ ngủ, hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) phải thiết lập bộ điều khiển CAN vào chế độ ngủ logic.⌋ ()

[SWS\_Can\_00405] ⌈ Chế độ ngủ logic này chỉ được rời khỏi khi hàm Can\_SetControllerMode (CAN\_CS\_STOPPED) được gọi.⌋ ()

[SWS\_Can\_00411] ⌈ Khi hàm Can\_SetControllerMode(CAN\_CS\_SLEEP) được gọi và bộ điều khiển CAN không ở trạng thái STOPPED hoặc SLEEP, nó phải phát hiện ra một chuyển đổi trạng thái không hợp lệ (So sánh với SWS\_Can\_00200).⌋ ()

##### State transition caused by Hardware Events

Chuyển đổi trạng thái do Hardware Wakeup (kích hoạt bởi sự kiện wake-up từ BUS CAN)

* SLEEP STOPPED
* Kích hoạt bởi các L-PDUs đến
* Mô-đun ECU Statemanagerđược thông báo bằng hàm EcuM\_CheckWakeup

Chuyển đổi trạng thái này chỉ xảy ra khi chế độ ngủ được hỗ trợ bởi phần cứng.

[SWS\_Can\_00270] ⌈ Khi có sự đánh thức phần cứng (được kích hoạt bởi một sự kiện đánh thức từ BUS CAN), bộ điều khiển CAN phải chuyển sang trạng thái DỪNG LẠI.⌋ ()

[SWS\_Can\_00271] ⌈ Khi có sự đánh thức phần cứng (được kích hoạt bởi một sự kiện wake-up từ BUS CAN), Mô-đun Can phải gọi hàm EcuM\_CheckWakeup hoặc trong ngữ cảnh Can\_MainFunction\_Wakeup.⌋ (SRS\_BSW\_00375, SRS\_SPAL\_12069, SRS\_Can\_01054)

[SWS\_Can\_00269] ⌈ Mô-đun Can không được tiếp tục xử lý L-PDU gây ra sự đánh thức.⌋ ()

[SWS\_Can\_00048] ⌈ Trong trường hợp có sự đánh thức BUS CAN trong quá trình chuyển tiếp ngủ, hàm Can\_SetControllerMode(CAN\_CS\_STOPPED) phải trả về E\_NOT\_OK.⌋ (SRS\_Can\_01122)

**State transition caused by Bus-Off (triggered by state change of CAN controller)**

[SWS\_Can\_00020] ⌈

* STARTED STOPPED
* Được kích hoạt bởi phần cứng nếu bộ điều khiển CAN đạt trạng thái bus-off
* Mô-đun CanIf được thông báo bằng hàm CanIf\_ControllerBusOff sau khi trạng thái STOPPED được đạt đến liên quan đến CAN Controllers tương ứng với abstract CanIf ControllerId..⌋ (SRS\_Can\_01055)

[SWS\_Can\_00272] ⌈ Sau khi phát hiện bus-off, CAN Controllers phải chuyển sang trạng thái STOPPED và mô-đun Can phải đảm bảo rằng CAN Controllers không còn tham gia vào mạng nữa. ⌋ (SRS\_Can\_01060)

[SWS\_Can\_00273] ⌈ Sau khi phát hiện bus-off, mô-đun Can phải hủy các tin nhắn đang chờ xử lý. ⌋ (SRS\_Can\_01060)

[SWS\_Can\_00274] ⌈ Mô-đun Can phải tắt hoặc ngăn chặn việc phục hồi bus-off tự động.⌋ (SRS\_Can\_01060)

##### State transition caused by function Can\_DeInit

* STOPPED UNINIT; SLEEP UNINIT (cho tất cả các bộ điều khiển trong HW unit)
* Được kích hoạt bởi phần mềm bằng cách gọi hàm Can\_DeInit
* Chuẩn bị tất cả các CAN Controllers trong HW Unit để được cấu hình lại.

[SWS\_Can\_91010] ⌈ Hàm Can\_DeInit phải đặt tất cả các CAN Controllers vào trạng thái UNINIT⌋ (SRS\_Can\_01166, SRS\_Can\_01053)

Khi hàm Can\_DeInit được gọi và mô-đun Can không ở trạng thái CAN\_READY hoặc bất kỳ CAN Controllers nào ở trạng thái STARTED, nó phải phát ra lỗi CAN\_E\_TRANSITION (Tham khảo [SWS\_Can\_91011] và [SWS\_Can\_91012]).

#### Can module/Controller Initialization

Mô-đun ECU State Manager phải khởi tạo mô-đun Can trong giai đoạn khởi động bằng cách gọi hàm Can\_Init trước khi sử dụng bất kỳ hàm nào khác của mô-đun Can.

[SWS\_Can\_00250] ⌈ Hàm Can\_Init phải khởi tạo:

Biến tĩnh, bao gồm cờ,

Cài đặt chung cho toàn bộ CAN HW Unit

Cài đặt cụ thể của CAN Controllers cho mỗi CAN Controllers.⌋ (SRS\_BSW\_00101)

[SWS\_Can\_00053] ⌈ Can\_Init không được thay đổi các thanh ghi của các CAN Controller Hardware mà không được sử dụng. ⌋ (SRS\_SPAL\_12125)

Mô-đun Can phải áp dụng các quy tắc sau đây về việc khởi tạo các thanh ghi của CAN Controllers:

* [SWS\_Can\_00407] ⌈ Nếu phần cứng chỉ cho phép sử dụng thanh ghi một lần, mô-đun Can thực hiện chức năng đó chịu trách nhiệm khởi tạo thanh ghi.
* Nếu thanh ghi có thể ảnh hưởng đến nhiều mô-đun phần cứng và nếu nó là một thanh ghi I/O thì nó phải được khởi tạo bởi trình điều khiển PORT.
* Nếu thanh ghi có thể ảnh hưởng đến nhiều mô-đun phần cứng và nếu nó không phải là một thanh ghi I/O thì nó phải được khởi tạo bởi trình điều khiển MCU.
* Các thanh ghi chỉ có thể ghi một lần và cần phải được khởi tạo trực tiếp sau khi thiết lập lại phải được khởi tạo bởi mã khởi đầu.
* Tất cả các thanh ghi khác phải được khởi tạo bởi mã khởi đầu.⌋ (SRS\_SPAL\_12461)

[SWS\_Can\_00056] ⌈ Các yếu tố cấu hình sau khi xây dựng được đánh dấu là 'nhiều' ('M' hoặc 'x') trong chương 10 có thể được chọn bằng cách chuyển con trỏ 'Config' vào hàm khởi tạo của mô-đun. ⌋ ()

[SWS\_Can\_00062] ⌈ Nếu Can\_SetBaudrate xác định rằng việc thay đổi cấu hình nhắm mục tiêu đòi hỏi một việc khởi tạo lại và CAN Controllers ở trạng thái STOPPED, hàm Can\_SetBaudrate phải khởi tạo lại CAN Controllers và các cài đặt cụ thể của bộ điều khiển.⌋ (SRS\_Can\_01139, SRS\_Can\_01042)

Nếu việc khởi tạo lại cần thiết, CAN Controllers phải được chuyển sang trạng thái STOPPED trước khi Can\_SetBaudrate() có thể được thực hiện và cấu hình baud rate mới có thể được áp dụng.

[SWS\_Can\_00255] ⌈ Hàm Can\_SetBaudrate chỉ ảnh hưởng đến các khu vực thanh ghi chứa cấu hình cụ thể cho một bộ điều khiển CAN duy nhất. ⌋ ()

[SWS\_Can\_00021] ⌈ Cấu hình bộ điều khiển được mong muốn có thể được chọn với tham số Config. ⌋ (SRS\_BSW\_00344, SRS\_BSW\_00404, SRS\_BSW\_00405, SRS\_SPAL\_12263, SRS\_SPAL\_12265)

[SWS\_Can\_00291] ⌈ Config là một con trỏ vào mảng của cấu trúc dữ liệu cụ thể của việc triển khai được lưu trữ trong ROM. Các bộ cấu hình bộ điều khiển khác nhau được đặt dưới dạng các cấu trúc dữ liệu trong ROM.⌋ (SRS\_BSW\_00438)

Các giá trị có thể có cho Cấu hình được cung cấp bởi mô tả cấu hình (xem chương 10).

Cấu hình mô-đun Can xác định cài đặt CAN HW Unit và tham chiếu đến các bộ cấu hình CAN Controllers mặc định.

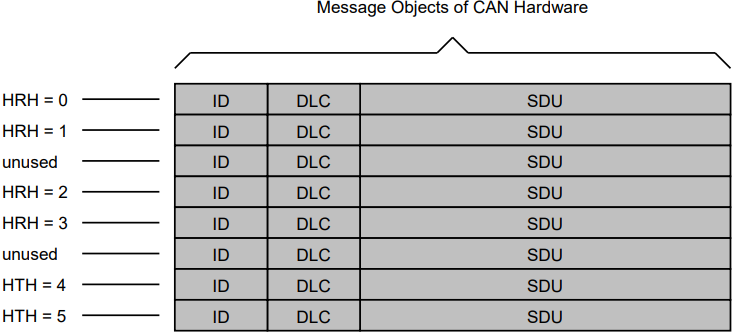
#### Truyền L-PDU

Trong quá trình truyền L-PDU, mô-đun Can chuyển đổi nội dung ID và Độ dài Dữ liệu của L-PDU sang một định dạng cụ thể của phần cứng (nếu cần) và kích hoạt việc truyền.

[SWS\_Can\_00059] ⌈ Bản đồ dữ liệu bởi CAN sang bộ nhớ được xác định theo cách mà byte dữ liệu CAN được gửi ra đầu tiên là phần tử mảng 0, byte dữ liệu CAN được gửi ra cuối cùng là phần tử mảng 7 hoặc 63 trong trường hợp của CAN FD.⌋ (SRS\_SPAL\_12063)

[SWS\_Can\_00427] ⌈ Nếu bản trình bày bên trong bộ đệm phần cứng CAN khác với định nghĩa AUTOSAR, mô-đun Can phải cung cấp một Bộ đệm SDU được điều chỉnh cho các lớp trên cùng.⌋ ()

[SWS\_Can\_00100] ⌈ Một số đối tượng phần cứng TX với các HTH duy nhất có thể được cấu hình. Mô-đun CanIf cung cấp HTH như tham số của yêu cầu TX. Xem Hình 7-3 cho một cấu hình có thể.⌋ (SRS\_Can\_01135)



Hình 7-3: Ví dụ về việc gán HTHs và HRHs cho các Hardware Objects. Việc đánh số HTHs và HRHs là cụ thể cho từng triển khai. Việc đánh số được chọn chỉ là một ví dụ.

[SWS\_Can\_00276] ⌈ Hàm Can\_Write sẽ lưu trữ swPduHandle được đưa vào trong tham số PduInfo cho đến khi mô-đun Can gọi hàm CanIf\_TxConfirmation cho yêu cầu này, trong đó swPduHandle được đưa ra như là tham số.⌋ ()

Tính năng của SWS\_Can\_00276 được sử dụng để giảm thời gian tìm kiếm trong triển khai của mô-đun CanIf.

[SWS\_Can\_00016] ⌈ Mô-đun Can sẽ gọi CanIf\_TxConfirmation để chỉ ra việc truyền thành công. Nó có thể được gọi bởi dịch vụ ngắt TX của nguồn tài nguyên phần cứng tương ứng hoặc bên trong hàm Can\_MainFunction\_Write trong trường hợp chế độ đánh giá.⌋

##### Ưu tiên chéo (Priority Inversion)

Việc truyền đa kênh là cần thiết để ngăn chặn ưu tiên chéo (xem chương 2.1).

[SWS\_Can\_00277] ⌈ Mô-đun Can sẽ cho phép chức năng "Truyền đa kênh" có thể cấu hình tĩnh (BẬT | TẮT) tại thời điểm biên dịch trước.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00401] ⌈ Nhiều đối tượng phần cứng truyền (được xác định bởi "CanHwObjectCount") sẽ được gán bởi một HTH để đại diện cho một thực thể truyền đến lớp trên cùng.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00402] ⌈ Mô-đun Can sẽ hỗ trợ các cơ chế truyền đa kênh cho các thiết bị nơi mà:

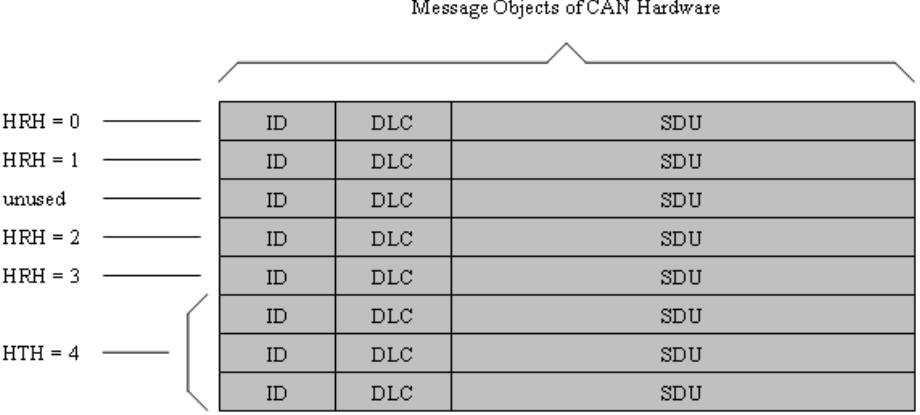
* Nhiều đối tượng phần cứng truyền, được nhóm lại thành một thực thể truyền có thể được điền vào qua cùng một bộ đăng ký, và vi điều khiển lưu trữ L-PDU vào một bộ đệm trống tự động,

hoặc

* Phần cứng cung cấp bộ registers hoặc chức năng để xác định một đối tượng phần cứng truyền trống trong một thực thể truyền.⌋ (SRS\_Can\_01134)

[SWS\_Can\_00403] ⌈ Mô-đun Can sẽ hỗ trợ truyền đa kênh cho các thiết bị, mà gửi các L-PDU theo thứ tự ưu tiên của L-PDU.⌋ (SRS\_Can\_01134)

Lưu ý: Tránh mô phỏng phần mềm của việc xử lý ưu tiên, vì chi phí cơ hội sẽ hủy đi lợi ích của việc truyền đa kênh.



Hình 7-4: Ví dụ về việc gán HTHs và HRHs cho các Đối tượng Phần cứng với truyền thông đa luồng. Các số thứ tự của HTHs và HRHs là cụ thể cho việc triển khai. Các số thứ tự được chọn chỉ là một ví dụ.

##### Tính nhất quán của dữ liệu truyền

[SWS\_Can\_00011] ⌈ Module Can phải sao chép dữ liệu trực tiếp từ bộ đệm của tầng trên. Trách nhiệm của tầng trên là duy trì sự nhất quán của bộ đệm cho đến khi cuộc gọi hàm trả về (Can\_Write).⌋ (SRS\_SPAL\_12075, SRS\_Can\_01059)

#### Nhận L-PDU

[SWS\_Can\_00279] ⌈ Khi nhận được L-PDU, module Can sẽ gọi hàm RX indication CanIf\_RxIndication với ID, HoH, abstract CanIf ControllerId trong tham số Mailbox, và chiều dài dữ liệu và con trỏ đến bộ đệm L-SDU trong tham số PduInfoPtr.⌋ (SRS\_Can\_01045)

[SWS\_Can\_00423] ⌈ Trong trường hợp của một khung CAN mở rộng, module Can sẽ chuyển đổi ID thành một định dạng chuẩn vì Tầng trên (CANIF) không biết rằng khung CAN nhận được là một khung CAN tiêu chuẩn hay mở rộng. Trong trường hợp LÀ một khung CAN mở rộng, MSB của ID trong một khung CAN nhận được cần được đặt là '1' để đánh dấu khung CAN nhận được là Mở rộng.⌋ ()

[SWS\_Can\_00396] ⌈ RXTemporary-interrupt service routine của tài nguyên phần cứng tương ứng hoặc hàm Can\_MainFunction\_Read trong trường hợp chế độ đánh giá bằng cách bỏ phiếu sẽ gọi hàm gọi lại CanIf\_RxIndication.⌋ (SRS\_Can\_01045)

[SWS\_Can\_00060] ⌈ Ánh xạ dữ liệu bằng CAN vào bộ nhớ được xác định theo cách rằng byte dữ liệu CAN nhận được đầu tiên là phần tử mảng 0, byte dữ liệu CAN nhận được cuối cùng là phần tử mảng 7 hoặc 63 trong trường hợp CAN FD. Nếu bảng trình bày bên trong bộ đệm phần cứng CAN khác với định nghĩa AUTOSAR, module Can phải cung cấp một bộ đệm SDU được điều chỉnh cho các tầng trên.⌋ (SRS\_SPAL\_12063)

[SWS\_Can\_00501] ⌈ CanDrv sẽ chỉ ra liệu thông điệp nhận được là một khung CAN truyền thống hay một khung CAN FD như mô tả trong Can\_IdType.⌋ (SRS\_Can\_01162)

##### Tính nhất quán của dữ liệu nhận được

Để ngăn mất mát thông điệp nhận được, một số bộ điều khiển hỗ trợ một FIFO được xây dựng từ một tập hợp các đối tượng phần cứng, trong khi trên các bộ điều khiển khác, có thể cấu hình một đối tượng phần cứng khác với các thuộc tính tương tự mà hoạt động như một bộ đệm bóng bóng và bước vào khi đối tượng chính bận.

[SWS\_Can\_00489] ⌈ Trình điều khiển CAN phải hỗ trợ các bộ điều khiển mà triển khai một FIFO phần cứng. Kích thước của FIFO được cấu hình thông qua "CanHwObjectCount".⌋ ()

[SWS\_Can\_00490] ⌈ Các bộ điều khiển không hỗ trợ FIFO phần cứng thường cung cấp khả năng triển khai cơ chế bộ nhớ đệm bóng bóng, nơi các đối tượng phần cứng bổ sung tiếp quản khi đối tượng phần cứng chính bận rộn. Số lượng đối tượng phần cứng được cấu hình thông qua "CanHwObjectCount".⌋ ()

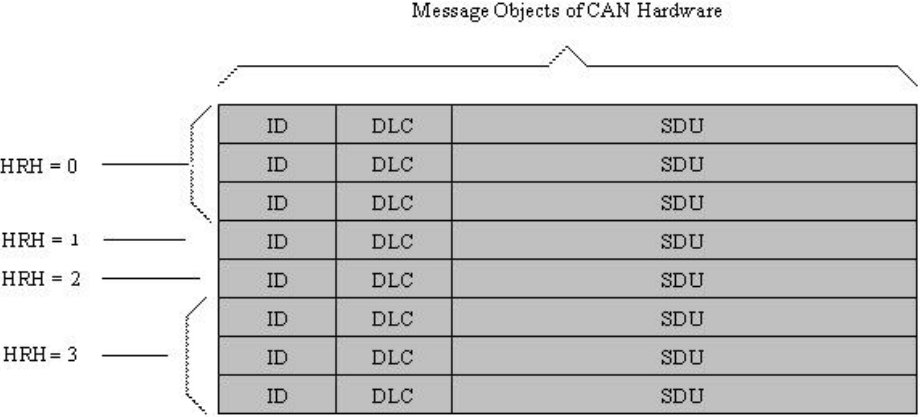


Figure 7-5: Ví dụ về việc gán các HRH giống nhau cho nhiều Đối tượng Phần cứng. Cách đánh số được chọn chỉ là một ví dụ.

[SWS\_Can\_00299] ⌈ Module Can sẽ sao chép L-SDU vào bộ đệm bóng sau khi nhận, nếu bộ đệm RX không thể được bảo vệ (khóa) bởi Phần cứng CAN chống ghi đè bởi một thông điệp mới được nhận.⌋ ()

[SWS\_Can\_00300] ⌈ Module Can sẽ sao chép L-SDU vào bộ đệm bóng, nếu Phần cứng CAN không có sẵn toàn cầu.⌋ ()

Quá trình RX hoàn chỉnh (bao gồm sao chép vào tầng đích, ví dụ: COM) được thực hiện trong ngữ cảnh của ngắt RX hoặc trong ngữ cảnh của Can\_MainFunction\_Read.

[SWS\_Can\_00012] ⌈ Module Can sẽ đảm bảo rằng cả các ISR lẫn hàm Can\_MainFunction\_Read không thể bị gián đoạn bởi chính nó. Bộ đệm Phần cứng CAN (hoặc bóng) luôn nhất quán, vì nó được ghi và đọc theo chuỗi trong một hàm duy nhất không bị gián đoạn bởi chính nó.⌋ (SRS\_Can\_01059)

Nếu Phần cứng CAN không thể được cấu hình để khóa đối tượng phần cứng RX sau khi nhận (tính năng phần cứng), có thể xảy ra tình trạng bộ đệm phần cứng bị ghi đè bởi một thông điệp mới đến. Trong trường hợp này, bộ điều khiển CAN phát hiện một sự kiện "ghi đè", nếu được hỗ trợ bởi phần cứng.

Nếu Phần cứng CAN có thể được cấu hình để khóa đối tượng phần cứng RX sau khi nhận, có thể xảy ra tình trạng thông điệp mới đến không thể được lưu vào bộ đệm phần cứng. Trong trường hợp này, bộ điều khiển CAN phát hiện một sự kiện "tràn", nếu được hỗ trợ bởi phần cứng.

[SWS\_Can\_00395] ⌈ Module Can sẽ tạo ra lỗi thời gian chạy CAN\_E\_DATALOST trong trường hợp phát hiện sự kiện "ghi đè" hoặc "tràn".⌋ ()

Gợi ý Triển khai:

Nhà thiết kế hệ thống phải đảm bảo rằng thời gian chạy cho việc nhận thông điệp (điều khiển ngắt hoặc đánh giá bằng cách bỏ phiếu) tương ứng với việc nhận nhanh nhất có thể trong hệ thống.

#### Khái niệm Wakeup

Module Can xử lý các sự thức dậy có thể được phát hiện bởi bản thân bộ điều khiển Can và không thông qua bộ chuyển mạch Can. Có hai kịch bản có thể xảy ra: thức dậy bằng ngắt và thức dậy bằng cách đánh giá bằng cách bỏ phiếu.

Đối với thức dậy bằng ngắt, một ISR của Module Can được gọi khi phần cứng phát hiện thức dậy.

[SWS\_Can\_00364] ⌈ Nếu ISR cho các sự kiện thức dậy được gọi, nó sẽ gọi EcuM\_CheckWakeup lần lượt. Tham số được truyền cho EcuM\_CheckWakeup sẽ là ID của nguồn thức dậy được tham chiếu bởi tham số cấu hình CanWakeupSourceRef.⌋ (SRS\_BSW\_00375, SRS\_SPAL\_12069, SRS\_Can\_01054)

Bộ điều khiển Trạng thái ECU sau đó sẽ thiết lập MCU và gọi lại Module Can thông qua Giao diện Can, dẫn đến việc gọi Can\_CheckWakeup.

Khi các sự kiện thức dậy được phát hiện bằng cách đánh giá bằng cách bỏ phiếu, Bộ điều khiển Trạng thái ECU sẽ gọi Can\_CheckWakeup theo chu kỳ qua Giao diện Can như trước đây. Trong cả hai trường hợp, Can\_CheckWakeup sẽ kiểm tra xem có sự thức dậy được phát hiện bởi một bộ điều khiển Can và trả về kết quả. Bộ điều khiển CAN sau đó sẽ thông báo về sự kiện thức dậy cho Bộ điều khiển Trạng thái ECU qua EcuM\_SetWakeupEvent.

Xác thực thức dậy để ngăn chặn các sự kiện thức dậy sai lầm sẽ được thực hiện bởi Bộ điều khiển Trạng thái ECU và Giao diện Can sau đó và không có sự giúp đỡ nào từ Module Can.

Để biết mô tả tổng quan về các cơ chế thức dậy và biểu đồ chuỗi thức dậy, xin tham khảo Chi tiết về Quản lý Trạng thái ECU [7].

#### Khái niệm Thông báo

Module Can chỉ cung cấp một giao diện thông báo được kích hoạt bởi sự kiện đến cho module CanIf. Mỗi thông báo được đại diện bởi một hàm gọi lại.

[SWS\_Can\_00099] ⌈ Các sự kiện phần cứng có thể được phát hiện bằng một ngắt hoặc bằng cách bỏ phiếu các cờ trạng thái của các đối tượng phần cứng. Các khả năng cấu hình liên quan đến việc bỏ phiếu là phụ thuộc vào phần cứng (tức là các sự kiện nào có thể được bỏ phiếu, các sự kiện nào cần được bỏ phiếu), và không bị hạn chế bởi tiêu chuẩn này.⌋ (SRS\_Can\_01132)

[SWS\_Can\_00007] ⌈ Phải có khả năng cấu hình trình điều khiển sao cho không có ngắt nào được sử dụng (toàn bộ bỏ phiếu). ⌋ (SRS\_Can\_01062)

Cấu hình của những gì được bỏ phiếu và không được bỏ phiếu bởi Module Can là nội bộ của trình điều khiển và không hiển thị bên ngoài module. Việc bỏ phiếu được thực hiện bên trong các chức năng chính CAN (Can\_MainFunction\_xxx). Các sự kiện được bỏ phiếu cũng được thông báo qua hàm gọi lại phù hợp. Sau đó, ngữ cảnh gọi không phải là ISR mà là hàm chính của CAN. Việc triển khai tất cả các hàm gọi lại phải được thực hiện như ngữ cảnh gọi là ISR.

Để biết thêm chi tiết, xem mô tả các chức năng chính CAN như Can\_MainFunction\_Read, Can\_MainFunction\_Write, Can\_MainFunction\_BusOff và Can\_MainFunction\_Wakeup.

#### Vấn đề Reentrancy

Một hàm phải thỏa mãn các điều kiện sau để có thể tái nhập:

- Nó sử dụng tất cả các biến chia sẻ theo cách nguyên tử, trừ khi mỗi biến được phân bổ cho một phiên bản cụ thể của hàm.

- Nó không gọi các hàm không tái nhập.

- Nó không sử dụng phần cứng theo cách không nguyên tử.

Các yêu cầu truyền gửi đơn giản được chuyển tiếp bởi mô-đun CanIf bên trong hàm CanIf\_Transmit.

Hàm CanIf\_Transmit là tái nhập. Do đó, hàm Can\_Write cần được triển khai an toàn cho luồng (ví dụ bằng cách sử dụng mutex):

Các cuộc gọi tiếp theo (phân biệt) sẽ trả về với CAN\_BUSY khi việc ghi không thể thực hiện tái nhập. (ví dụ: ghi vào các Xử lý TX phần cứng khác nhau được phép, ghi vào cùng Xử lý TX không được phép)

Trong trường hợp CAN\_BUSY, mô-đun CanIf sẽ xếp hàng yêu cầu đó. (hành vi giống như nếu tất cả các đối tượng phần cứng đều bận)

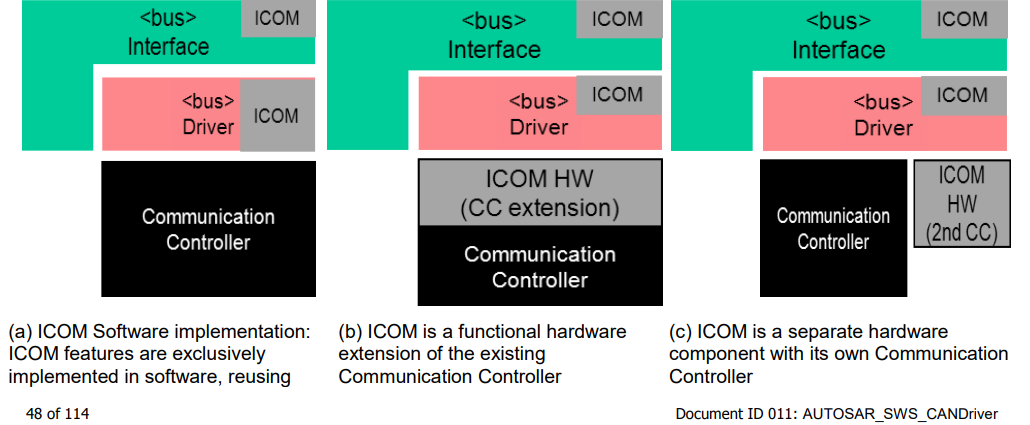
Các hàm Can\_EnableCanInterrupts và Can\_DisableCanInterrupts có thể được gọi trong các hàm tái nhập. Do đó, các hàm này cũng cần được triển khai tái nhập.

Tất cả các dịch vụ khác không cần được triển khai như các hàm tái nhập.

Các hàm chính CAN (ví dụ: Can\_MainFunction\_Read) không được phép bị gián đoạn bởi chính chúng. Do đó, các hàm chính CAN này không phải là hàm tái nhập.

#### Mạng ảo

Tối ưu hóa hiệu suất năng lượng đang trở nên ngày càng quan trọng trong tất cả các lĩnh vực ô tô vì tiêu thụ năng lượng có tác động trực tiếp đến tiêu thụ nhiên liệu, lượng khí CO2 phát thải và phạm vi của các xe hybrid hoặc toàn bộ điện. Khái niệm Mạng Ảo có tiềm năng cao để giảm năng lượng ở cấp độ ECU.



Dưới đây là bản dịch từ tiếng Anh sang tiếng Việt của đoạn văn trên, đảm bảo ngữ pháp, đầy đủ nội dung và logic:

Bộ điều khiển truyền thông hiện có

Hình 7-6: Các phương án thực hiện ICOM khả thi

Việc sử dụng các Bộ điều khiển Truyền thông Thông minh (ICOM) hỗ trợ các tính năng đó (không bắt buộc phải có phần cứng cụ thể). Nếu một số hoặc tất cả chức năng của một ECU tạm thời không cần thiết, ví dụ dựa trên trạng thái của xe, ECU có thể chuyển sang chế độ "Mạng giả". Trong chế độ này, MCU và/hoặc các ngoại vi sẽ chuyển sang chế độ tiêu thụ năng lượng thấp. Chỉ có ICOM và các bộ thu/phát kết nối với nó vẫn hoạt động. ICOM sẽ tạo ra một sự kiện đánh thức, gây ra bởi ví dụ như một tin nhắn bus được nhận, khi ECU cần khởi động lại hoạt động. Tùy thuộc vào cách thực hiện ICOM, ID tin nhắn và nội dung tin nhắn nhận được có thể được đánh giá và lọc hoàn toàn trong phần cứng, hoặc yêu cầu một cơ chế gọi lại trong phần mềm.

Như được thể hiện trong Hình 7-6, việc thực hiện ECU có thể được chia thành ba phương án khả thi - Phương án (a) là phương án phần mềm, không có phần cứng cụ thể để hỗ trợ Mạng giả. Phương án (b) mô tả một phần mở rộng phần cứng chức năng của bộ điều khiển truyền thông và phương án (c) cho thấy một biến thể phần cứng với bộ điều khiển truyền thông thứ hai mở rộng để xử lý đánh thức. Tất cả các biến thể và cách thực hiện ICOM đều phải được hỗ trợ bởi Mạng giả. Tùy thuộc vào cách thực hiện phần cứng, ICOM cũng có khả năng gửi tin nhắn. Bằng cách sử dụng ICOM để tiếp tục gửi, ví dụ như các tin nhắn trạng thái, các nút khác phụ thuộc vào tin nhắn đó sẽ không bị ảnh hưởng bởi một ECU ở chế độ Mạng giả. Hơn nữa, Mạng giả nhằm mục đích giảm thời gian phản hồi đánh thức, tức là khoảng thời gian giữa một sự kiện đánh thức và hành vi hợp lệ của một ECU. Bằng cách sử dụng ICOM để lưu các tin nhắn liên quan trong khi chế độ Mạng giả được kích hoạt, ứng dụng sẽ có quyền truy cập vào các giá trị tín hiệu hợp lệ cuối cùng ngay sau khi khởi động lại hoạt động. Do đó, ECU có thể phản hồi ngay lập tức với yêu cầu của người dùng sau khi đánh thức và không phải chờ đợi cho đến khi tin nhắn tương ứng được nhận lại.

Đây là một đoạn mô tả các yêu cầu phần mềm liên quan đến việc xử lý chế độ "Pretended Networking" trong driver CAN của AUTOSAR. Tôi sẽ dịch đoạn này sang tiếng Việt:

##### Hỗ trợ xử lý chế độ Pretended Networking [SWS\_Can\_00497]

⌈Trình điều khiển CAN phải vô hiệu hóa chế độ Pretended Networking sau khi khởi tạo bộ điều khiển CAN.⌋

() Kích hoạt chế độ Pretended Networking:

[SWS\_Can\_00462] ⌈Chế độ Pretended Networking sẽ được kích hoạt bằng cách gọi Can\_SetIcomConfiguration() với một ID cấu hình không được đặt là 0.⌋

[SWS\_Can\_00464] ⌈CanDrv chịu trách nhiệm thực hiện cấu hình lại bộ điều khiển CAN (bao gồm ICOM) theo các tham số CanIcomConfig cho cấu hình được chọn (CanIcomConfigId).⌋

[SWS\_Can\_00467] ⌈Nếu kích hoạt thành công, CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error được đặt thành CAN\_OK.⌋

Trong đoạn này, các yêu cầu phần mềm được liệt kê như sau:

1. Sau khi khởi tạo bộ điều khiển CAN, trình điều khiển CAN phải vô hiệu hóa chế độ Pretended Networking.

2. Để kích hoạt chế độ Pretended Networking, phải gọi hàm Can\_SetIcomConfiguration() với một ID cấu hình không phải 0.

3. CanDrv có trách nhiệm cấu hình lại bộ điều khiển CAN (bao gồm ICOM) theo các tham số cấu hình CanIcomConfig cho cấu hình được chọn (CanIcomConfigId).

4. Nếu kích hoạt chế độ Pretended Networking thành công, hàm CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành CAN\_OK.

Các yêu cầu này xác định cách thức kích hoạt và xử lý chế độ Pretended Networking trong trình điều khiển CAN của AUTOSAR.

Đây là phần tiếp theo của các yêu cầu phần mềm liên quan đến xử lý chế độ Pretended Networking trong AUTOSAR. Tôi sẽ dịch phần này sang tiếng Việt:

Nếu kích hoạt thành công thì CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành ICOM\_SWITCH\_E\_OK tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng. Nếu kích hoạt không thành công, CanIf\_CurrentIcomConfiguration sẽ được gọi với tham số Error đặt thành ICOM\_SWITCH\_E\_FAILED tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng.⌋

[SWS\_Can\_00468] ⌈Nếu Pretended Networking được kích hoạt, CanDrv sẽ gọi CanIf\_RxIndication() nếu và chỉ nếu tin nhắn nhận được khớp với các điều kiện đánh thức của CanIcomConfig (xem CanIcomWakeupCauses).⌋

[SWS\_Can\_00470] ⌈Nếu Pretended Networking được kích hoạt, CanDrv sẽ từ chối yêu cầu Can\_Write() với giá trị trả về CAN\_BUSY.⌋

[SWS\_Can\_00498] ⌈Trình điều khiển CAN sẽ vô hiệu hóa chế độ Pretended Networking trước khi bộ điều khiển CAN được khởi động bằng SetControllerMode(CAN\_CS\_STARTED)⌋

Vô hiệu hóa chế độ Pretended Networking:

[SWS\_Can\_00471] ⌈Chế độ Pretended Networking sẽ được vô hiệu hóa (tức là CanDrv sẽ hoạt động giống như khi nó được cấu hình mà không hỗ trợ Pretended Networking) bằng cách gọi Can\_SetIcomConfiguration() với ID cấu hình = 0.⌋

[SWS\_Can\_00472] ⌈Nếu Pretended Networking bị vô hiệu hóa, CanDrv sẽ xử lý các tin nhắn bình thường như đã cấu hình trong cấu hình thông thường.⌋

[SWS\_Can\_00474] ⌈Trình điều khiển CAN sẽ thông báo cho CanIf về một sự chuyển đổi cấu hình bằng cách gọi CanIf\_CurrentIcomConfiguration tham chiếu đến bộ điều khiển CAN tương ứng với CanIf ControllerId trừu tượng. Tham số lỗi được đặt thành ICOM\_SWITCH\_E\_OK nếu vô hiệu hóa thành công và ICOM\_SWITCH\_E\_FAILED nếu không.⌋

[SWS\_Can\_00499] ⌈Trình điều khiển CAN sẽ vô hiệu hóa chế độ Pretended Networking trước khi bộ điều khiển CAN bị dừng bằng SetControllerMode(CAN\_CS\_STOPPED).⌋

##### Hỗ trợ gửi và nhận tin nhắn tự động

[SWS\_Can\_00477] ⌈Việc gửi tin nhắn tự động trong chế độ Pretended Networking chỉ được hỗ trợ nếu có phần cứng ICOM bổ sung. Một tham số cấu hình xác định liệu có hỗ trợ phần cứng hay không (Tham khảo CanIcomVariant).⌋

[SWS\_Can\_00478] ⌈Nếu ICOM được thực hiện bằng phần mềm, bộ điều khiển sẽ không gửi tin nhắn trong chế độ Pretended Networking.⌋

[SWS\_Can\_00479] ⌈CanDriver sẽ chuyển tiếp tất cả các tin nhắn nhận được trong Chế độ Pretended Networking đến CanIf.⌋

Phần này mô tả các yêu cầu về cách xử lý kích hoạt/vô hiệu hóa chế độ Pretended Networking, xử lý tin nhắn nhận/gửi trong chế độ này, và các điều kiện hỗ trợ gửi/nhận tin nhắn tự động khi ở chế độ Pretended Networking.

Đây là phần mô tả về phân loại lỗi trong mô-đun CAN của AUTOSAR. Tôi sẽ dịch nó sang tiếng Việt:

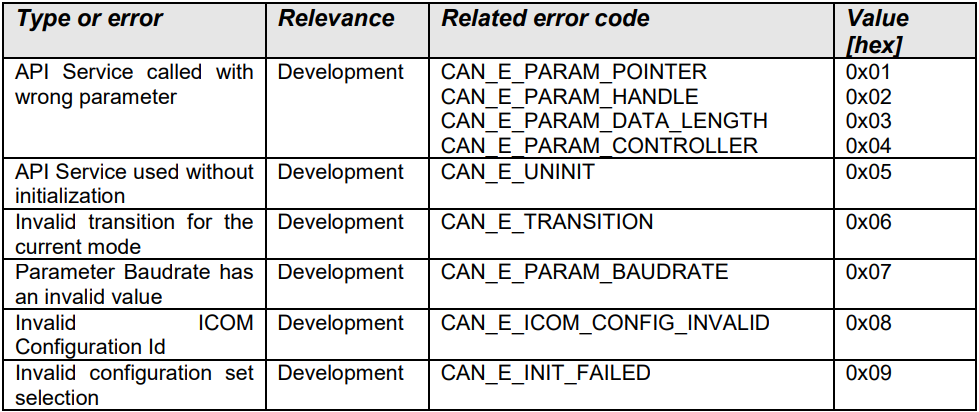
#### Phân loại lỗi

[SWS\_Can\_00104] ⌈Mô-đun CAN phải có khả năng phát hiện các lỗi và ngoại lệ sau đây tùy thuộc vào cấu hình của nó (mặc định/sản xuất)⌋ (SRS\_BSW\_00337, SRS\_BSW\_00385, SRS\_BSW\_00331)

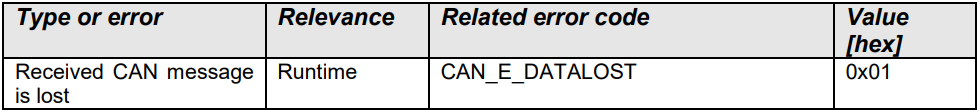
##### Lỗi trong quá trình phát triển

Phần này nêu ra rằng mô-đun CAN phải có khả năng phát hiện các lỗi và ngoại lệ khác nhau trong quá trình vận hành, tùy thuộc vào cấu hình của nó (cấu hình mặc định hay cấu hình sản xuất). Các yêu cầu phần mềm liên quan đến phân loại lỗi sẽ được liệt kê trong các phần tiếp theo.

Phần 7.11.1 sẽ đề cập đến các lỗi phát sinh trong quá trình phát triển phần mềm CAN, như lỗi cấu hình, lỗi tham số đầu vào không hợp lệ, v.v. Đây là dạng lỗi cần được phát hiện và xử lý trong giai đoạn phát triển để đảm bảo chất lượng phần mềm.



##### Runtime Errors



Đoạn văn bản này tiếp tục mô tả các loại lỗi khác nhau trong mô-đun CAN của AUTOSAR. Tôi sẽ dịch nó sang tiếng Việt:

[SWS\_Can\_00026] ⌈Mô-đun CAN phải báo các lỗi gây ra bởi việc sử dụng không đúng cách API của mô-đun CAN. Điều này bao gồm kiểm tra tham số API và lỗi tuần tự gọi hàm.⌋ (SRS\_BSW\_00337, SRS\_BSW\_00323, SRS\_SPAL\_00157)

[SWS\_Can\_00091] ⌈Sau khi trả về DET, hàm của mô-đun CAN gây ra lỗi phát triển sẽ trả về ngay lập tức.⌋ (SRS\_SPAL\_12448)

[SWS\_Can\_00089] ⌈Môi trường của mô-đun CAN chỉ báo các lỗi phát triển trong giá trị trả về của một hàm của mô-đun CAN khi DET được bật và hàm cung cấp giá trị trả về. Giá trị trả về là E\_NOT\_OK.⌋ (SRS\_BSW\_00369, SRS\_BSW\_00386, SRS\_SPAL\_12448)

##### Lỗi tạm thời

Không có lỗi tạm thời.

##### Lỗi sản xuất

Mô-đun CAN không gọi Trình quản lý Sự kiện Chẩn đoán, vì không có mã lỗi sản xuất được xác định cho mô-đun CAN.

##### Giá trị trả về

CAN\_BUSY được báo cáo thông qua giá trị trả về của hàm Can\_Write. Mô-đun CanIf phản ứng theo các sơ đồ trình tự được quy định cho mô-đun CanIf.

E\_NOT\_OK được báo cáo thông qua giá trị trả về trong trường hợp có sự đánh thức trong quá trình chuyển sang chế độ ngủ.

Các sự kiện Busoff và Đánh thức được chuyển tiếp thông qua các hàm gọi lại thông báo.

#### Hỗ trợ CAN FD

Vì lý do hiệu suất, một số bộ điều khiển CAN cho phép sử dụng tính năng Flexible Data-Rate gọi là CAN FD (xem "CAN with Flexible Data-Rate" specification). Được chỉ định trong giai đoạn tranh chấp, có thể chuyển sang tốc độ baud cao hơn trong khi truyền dữ liệu và CRC. Tốc độ baud thứ hai này phải được cấu hình bằng cách mở rộng CanControllerBaudrateConfig với CanControllerFdBaudrateConfig. Nếu một tốc độ baud đang hoạt động có cấu hình CAN FD (xem CanControllerFdBaudrateConfig), tính năng CAN FD sẽ được kích hoạt cho bộ điều khiển này. Tốc độ baud thứ hai đã chỉ định cần thiết để hỗ trợ việc nhận khung CAN FD với chuyển đổi tốc độ bit (BRS). Việc tốc độ baud thứ hai có được sử dụng để truyền hay không phụ thuộc vào tham số cấu hình CanControllerTxBitRateSwitch (xem CanControllerFdBaudrateConfig).

Tuy nhiên, có thể có trường hợp cần phải truyền các tin nhắn CAN 2.0 truyền thống trong mạng hỗ trợ tin nhắn CAN-FD, ví dụ để hỗ trợ đánh thức chọn lọc CAN. Trong những trường hợp này, cần hỗ trợ truyền xen kẽ các tin nhắn CAN truyền thống với tin nhắn CAN-FD. Điều này có thể đạt được ở cấp độ khung bằng cách sử dụng hai bit có trọng số lớn nhất của CanId (xem Can\_IdType, SWS\_Can\_00416) được truyền trong Can\_Write để chỉ định loại khung nào sẽ được sử dụng.

CAN FD cũng hỗ trợ dung lượng tải trọng mở rộng cho phép truyền lên đến 64 byte. Tính năng này cũng phụ thuộc vào cấu hình CAN FD (xem CanControllerFdBaudrateConfig). Do đó, nếu bộ điều khiển CAN ở chế độ CAN FD (có CanControllerFdBaudrateConfig hợp lệ) và cờ CAN FD được đặt trong CanId được truyền đến Can\_Write(), CanDrv hỗ trợ truyền PDU với độ dài lên đến 64 byte. Nếu có yêu cầu truyền một khung CAN FD và bộ điều khiển CAN không ở chế độ CAN FD (không có CanControllerFdBaudrateConfig), khung sẽ được gửi như khung CAN truyền thống miễn là độ dài PDU <= 8 byte.

Đoạn văn này mô tả cách xử lý các loại lỗi phát triển, lỗi tạm thời và lỗi sản xuất trong mô-đun CAN. Nó cũng giải thích về khả năng hỗ trợ CAN FD (Flexible Data-Rate) của CAN, cho phép truyền khung dữ liệu lớn hơn với tốc độ baud cao hơn.

### API specification

Tiền tố của tên hàm có thể được thay đổi trong một triển khai với nhiều mô-đun CAN như được mô tả trong [SWS\_Can\_00284].

#### Imported types

Trong chương này, tất cả các kiểu dữ liệu được nhập từ các mô-đun sau đây được liệt kê:

[SWS\_Can\_00222] ⌈

|  |  |  |
| --- | --- | --- |
| ***Module*** | ***Header File*** | ***Imported Type*** |
| Can\_GeneralTypes | Can\_GeneralTypes.h | Can\_ControllerStateType |
| Can\_GeneralTypes.h | Can\_ErrorStateType |
| Can\_GeneralTypes.h | Can\_HwHandleType |
| Can\_GeneralTypes.h | Can\_HwType |
| Can\_GeneralTypes.h | Can\_IdType |
| Can\_GeneralTypes.h | Can\_PduType |
| ComStack\_Types | ComStackTypes.h | IcomConfigIdType |
| ComStackTypes.h | IcomSwitch\_ErrorType |
| ComStackTypes.h | PduIdType |
| ComStackTypes.h | PduInfoType |
| EcuM | EcuM.h | EcuM\_WakeupSourceType |
| Icu | Icu.h | Icu\_ChannelType |
| Os | Os.h | StatusType |
| Os.h | TickRefType |
| Rte\_Os\_Type.h | CounterType |
| Std\_Types | StandardTypes.h | Std\_ReturnType |
| StandardTypes.h | Std\_VersionInfoType |

#### Type definitions

##### Can\_ConfigType

**[SWS\_Can\_00413]**

|  |  |  |
| --- | --- | --- |
| ***Name:*** | Can\_ConfigType | |
| ***Type:*** | Structure | |
| ***Range:*** | Implementation specific. |  |
| ***Description:*** | Đây là loại của cấu trúc dữ liệu bên ngoài chứa dữ liệu khởi tạo tổng thể cho CAN driver và các cài đặt SFR ảnh hưởng đến tất cả các bộ điều khiển. Hơn nữa, nó chứa các con trỏ đến các cấu trúc cấu hình bộ điều khiển. Nội dung của cấu trúc dữ liệu khởi tạo phụ thuộc vào phần cứng CAN cụ thể. | |
| ***Available via:*** | Can.h | |

##### Can\_PduType

**[SWS\_Can\_00415]**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_PduType | | |
| ***Type:*** | Structure | | |
| ***Element:*** | PduIdType | swPduHandle | -- |
| uint8 | length | -- |
| Can\_IdType | id | -- |
| uint8\* | sdu | -- |
| ***Description:*** | This type unites PduId (swPduHandle), SduLength (length), SduData (sdu), and CanId (id) for any CAN L-SDU. | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

##### Can\_IdType

**[SWS\_Can\_00416]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_IdType | | |
| ***Type:*** | uint32 | | |
| ***Range:*** | Standard32Bit | -- | 0..0x400007FF |
| Extended32Bit | -- | 0..0xDFFFFFFF |
| ***Description:*** | Represents the Identifier of an L-PDU. The two most significant bits specify the frame type:  00 CAN message with Standard CAN ID   1. CAN FD frame with Standard CAN ID   10 CAN message with Extended CAN ID  11CAN FD frame with Extended CAN ID | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

##### Can\_HwHandleType

**[SWS\_Can\_00429]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_HwHandleType | | |
| ***Type:*** | uint8, uint16 | | |
| ***Range:*** | Standard | -- | 0..0x0FF |
| Extended | -- | 0..0xFFFF |
| ***Description:*** | Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects use extended range. | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

##### Can\_HwType

**[SWS\_CAN\_00496]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_HwType | | |
| ***Type:*** | Structure | | |
| ***Element:*** | Can\_IdType | CanId | Standard/Extended CAN ID of CAN L- PDU |
| Can\_HwHandleType | Hoh | ID of the corresponding Hardware Object Range |
| uint8 | ControllerId | ControllerId provided by CanIf clearly  identify the corresponding controller |
| ***Description:*** | This type defines a data structure which clearly provides an Hardware Object Handle including its corresponding CAN Controller and therefore CanDrv as well as the specific CanId. | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

##### Extension to Std\_ReturnType

**[SWS\_Can\_00039]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Range:*** | CAN\_BUSY | 0x02 | transmit request could not be processed because no transmit object was available |
| ***Description:*** | Overlayed return value of Std\_ReturnType for CAN driver API Can\_Write() | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

⌋ (SRS\_BSW\_00331)

##### Can\_ErrorStateType

**[SWS\_Can\_91003]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_ErrorStateType | | |
| ***Type:*** | Enumeration | | |
| ***Range:*** | CAN\_ERRORSTATE\_ACTIVE | -- | The CAN controller takes fully part in communication. |
| CAN\_ERRORSTATE\_PASSIVE | -- | The CAN controller takes part in communication, but does not send active error frames. |
| CAN\_ERRORSTATE\_BUSOFF | -- | The CAN controller does not take part in communication. |
| ***Description:*** | Error states of a CAN controller. | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

##### Can\_ControllerStateType

**[SWS\_Can\_91013]** ⌈

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name:*** | Can\_ControllerStateType | | |
| ***Type:*** | Enumeration | | |
| ***Range:*** | CAN\_CS\_UNINIT | 0x00 | CAN controller state UNINIT. |
| CAN\_CS\_STARTED | 0x01 | CAN controller state STARTED. |
| CAN\_CS\_STOPPED | 0x02 | CAN controller state STOPPED. |
| CAN\_CS\_SLEEP | 0x03 | CAN controller state SLEEP. |
| ***Description:*** | States that are used by the several ControllerMode functions. | | |
| ***Available via:*** | Can\_GeneralTypes.h | | |

#### Function definitions

This is a list of functions provided for upper layer modules.

##### Services affecting the complete hardware unit

###### Can\_Init

**[SWS\_Can\_00223]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_Init | |
| ***Syntax:*** | void Can\_Init( const Can\_ConfigType\* Config) | |
| ***Service ID[hex]:*** | 0x00 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant | |
| ***Parameters (in):*** | Config | Pointer to driver configuration. |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | This function initializes the module. | |
| ***Available via:*** | Can.h | |

⌋ (SRS\_BSW\_00358, SRS\_BSW\_00414)

Symbolic names of the available configuration sets are provided by the configuration description of the Can module. See chapter [10](#_bookmark109) about configuration description.

**[SWS\_Can\_00174] ⌈** If development error detection for the Can module is enabled: The function Can\_Init shall raise the error CAN\_E\_TRANSITION if the driver is not in state CAN\_UNINIT.⌋ ()

**[SWS\_Can\_00408] ⌈** If development error detection for the Can module is enabled: The function Can\_Init shall raise the error CAN\_E\_TRANSITION if the CAN controllers are not in state UNINIT.⌋ ()

###### Can\_GetVersionInfo

**[SWS\_Can\_00224]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_GetVersionInfo | |
| ***Syntax:*** | void Can\_GetVersionInfo( Std\_VersionInfoType\* versioninfo) | |
| ***Service ID[hex]:*** | 0x07 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | None | |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | versioninfo | Pointer to where to store the version information of this module. |
| ***Return value:*** | None | |
| ***Description:*** | This function returns the version information of this module. | |
| ***Available via:*** | Can.h | |

**[SWS\_Can\_00177] ⌈** If development error detection for the Can module is enabled: The function Can\_GetVersionInfo shall raise the error CAN\_E\_PARAM\_POINTER if the parameter versionInfo is a null pointer.⌋ ()

###### Can\_DeInit

**[SWS\_Can\_91002]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_DeInit |
| ***Syntax:*** | void Can\_DeInit( void) |
| ***Service ID[hex]:*** | 0x10 |
| ***Sync/Async:*** | Synchronous |
| ***Reentrancy:*** | Non Reentrant |
| ***Parameters (in):*** | None |
| ***Parameters (inout):*** | None |
| ***Parameters (out):*** | None |
| ***Return value:*** | None |
| ***Description:*** | This function de-initializes the module. |
| ***Available via:*** | Can.h |

**⌋ (SRS\_Can\_01166, SRS\_BSW\_00336)**

**Note:** General behavior and constraints on de-initialization functions are specified by [SWS\_BSW\_00152], [SWS\_BSW\_00072], [SWS\_BSW\_00232], [SWS\_BSW\_00233]

**Caveat:** Caller of the Can\_DeInit function has to be sure no CAN controller is in the state STARTED

**[SWS\_Can\_91011] ⌈** If development error detection for the Can module is enabled: The function Can\_DeInit shall raise the error CAN\_E\_TRANSITION if the driver is not in state CAN\_READY.⌋ (SRS\_BSW\_00369)

**[SWS\_Can\_91012] ⌈** If development error detection for the Can module is enabled: The function Can\_DeInit shall raise the error CAN\_E\_TRANSITION if any of the CAN controllers is in state STARTED.⌋ (SRS\_BSW\_00369)

##### Services affecting one single CAN Controller

###### Can\_SetBaudrate

**[SWS\_CAN\_00491]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_SetBaudrate | |
| ***Syntax:*** | Std\_ReturnType Can\_SetBaudrate( uint8 Controller, uint16 BaudRateConfigID ) | |
| ***Service ID[hex]:*** | 0x0f | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant for different Controllers. Non reentrant for the same Controller. | |
| ***Parameters (in):*** | Controller | CAN controller, whose baud rate shall be set |
| BaudRateConfigID | references a baud rate configuration by ID (see CanControllerBaudRateConfigID) |
| ***Parameters***  ***(inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | Std\_ReturnType | E\_OK: Service request accepted, setting of (new) baud rate started  E\_NOT\_OK: Service request not accepted |
| ***Description:*** | This service shall set the baud rate configuration of the CAN controller. Depending  on necessary baud rate modifications the controller might have to reset. | |
| ***Available via:*** | Can.h | |

There might be several baud rate configurations available. The function Can\_SetBaudrate can be used to switch between different configurations. Depending on the old and new baud rate configuration only a subset of parameters may be changed during runtime and a re-initialization of the CAN Controller might be avoidable.

If the call of Can\_SetBaudrate will cause a re-initialization of the CAN Controller the CAN controller must be in state STOPPED when this function is called (see SWS\_Can\_00256 and SWS\_Can\_00260).

The CAN controller is in state STOPPED after (re-)initialization (see SWS\_Can\_00259).

[SWS\_Can\_00492] ⌈ If development error detection for the Can module is enabled: The function Can\_SetBaudrate shall raise the error CAN\_E\_UNINIT and return E\_NOT\_OK if the driver is not yet initialized.⌋ () [SWS\_Can\_00493] ⌈ If development error detection for the Can module is enabled: The function Can\_SetBaudrate shall raise the error CAN\_E\_PARAM\_BAUDRATE and return E\_NOT\_OK if the parameter BaudRateConfigID has an invalid value.⌋ () [SWS\_Can\_00494] ⌈ If development error detection for the Can module is enabled the function Can\_SetBaudrate shall raise the error CAN\_E\_PARAM\_CONTROLLER and return E\_NOT\_OK if the parameter Controller is out of range.⌋ () [SWS\_Can\_00500] ⌈ If the requested baud rate change can not performed without a re-initialization of the CAN Controller E\_NO\_OK shall be returned.⌋ ()

###### Can\_SetControllerMode

**[SWS\_Can\_00230]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_SetControllerMode | |
| ***Syntax:*** | Std\_ReturnType Can\_SetControllerMode( uint8 Controller, Can\_ControllerStateType Transition) | |
| ***Service ID[hex]:*** | 0x03 | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Non Reentrant | |
| ***Parameters (in):*** | Controller | CAN controller for which the status shall be changed |
| Transition | Transition value to request new CAN controller state |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | Std\_ReturnType | E\_OK: request accepted  E\_NOT\_OK: request not accepted, a development error occurred |
| ***Description:*** | This function performs software triggered state transitions of the CAN controller State machine. | |
| ***Available via:*** | Can.h | |

[**SWS\_Can\_00017**] ⌈ The function Can\_SetControllerMode shall perform software triggered state transitions of the CAN controller State machine. See also [SRS\_SPAL\_12169]⌋ (SRS\_SPAL\_12169, SRS\_Can\_01053)

[**SWS\_Can\_00384**] ⌈ Each time the CAN controller state machine is triggered with the state transition value CAN\_CS\_STARTED, the function Can\_SetControllerMode shall re-initialize the CAN controller with the same controller configuration set previously used by functions Can\_SetBaudrate or Can\_Init.⌋ () Refer to SWS\_Can\_00048 for the case of a wakeup event from CAN bus occurred during sleep transition.

[SWS\_Can\_00294] ⌈ The function Can\_SetControllerMode shall disable the wake-up interrupt, while checking the wake-up status. ⌋ ()

[SWS\_Can\_00196] ⌈ The function Can\_SetControllerMode shall enable interrupts that are needed in the new state. ⌋ ()

[SWS\_Can\_00425] ⌈ Enabling of CAN interrupts shall not be executed, when CAN interrupts have been disabled by function Can\_DisableControllerInterrupts.⌋ ()

[SWS\_Can\_00197] ⌈ The function Can\_SetControllerMode shall disable interrupts that are not allowed in the new state. ⌋ ()

[SWS\_Can\_00426] ⌈ Disabling of CAN interrupts shall not be executed, when CAN interrupts have been disabled by function Can\_DisableControllerInterrupts.⌋ ()

[SWS\_Can\_00198] ⌈ If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can\_SetControllerMode shall raise development error CAN\_E\_UNINIT and return E\_NOT\_OK.⌋ ()

[SWS\_Can\_00199] ⌈ If development error detection for the Can module is enabled: if the parameter Controller is out of range, the function Can\_SetControllerMode shall raise development error CAN\_E\_PARAM\_CONTROLLER and return E\_NOT\_OK.⌋ ()

[SWS\_Can\_00200] ⌈ If development error detection for the Can module is enabled: if an invalid transition has been requested, the function Can\_SetControllerMode shall raise the error CAN\_E\_TRANSITION and return E\_NOT\_OK.⌋ ()

###### Can\_DisableControllerInterrupts

**[SWS\_Can\_00231]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_DisableControllerInterrupts | |
| ***Syntax:*** | void Can\_DisableControllerInterrupts( uint8 Controller) | |
| ***Service ID[hex]:*** | 0x04 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | Controller | CAN controller for which interrupts shall be disabled. |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | This function disables all interrupts for this CAN controller. | |
| ***Available via:*** | Can.h | |

**⌋ (SRS\_BSW\_00312)**

**[SWS\_Can\_00049] ⌈** The function Can\_DisableControllerInterrupts shall access the CAN controller registers to disable all interrupts for that CAN controller only, if interrupts for that CAN Controller are enabled. ⌋ (SRS\_Can\_01043)

**[SWS\_Can\_00202] ⌈** When Can\_DisableControllerInterrupts has been called several times, Can\_EnableControllerInterrupts must be called as many times before the interrupts are re-enabled.⌋ ()

Implementation note:

The function Can\_DisableControllerInterrupts can increase a counter on every execution that indicates how many Can\_EnableControllerInterrupts need to be called before the interrupts will be enabled (incremental disable).

**[SWS\_Can\_00204] ⌈** The Can module shall track all individual enabling and disabling of interrupts in other functions (i.e. Can\_SetControllerMode) , so that the correct interrupt enable state can be restored.⌋ ()

Implementation example:

* in ‘interrupts enabled mode’: For each interrupt state change does not only modify the interrupt enable bit, but also a software flag.
* in ‘interrupts disabled mode’: only the software flag is modified.
* Can\_DisableControllerInterrupts and Can\_EnableControllerInterrupts do not modify the software flags.
* Can\_EnableControllerInterrupts reads the software flags to re-enable the correct interrupts.

**[SWS\_Can\_00205] ⌈** If development error detection for the Can module is enabled: The function Can\_DisableControllerInterrupts shall raise the error CAN\_E\_UNINIT if the driver not yet initialized.⌋ ()

**[SWS\_Can\_00206] ⌈** If development error detection for the Can module is enabled: The function Can\_DisableControllerInterrupts shall raise the error CAN\_E\_PARAM\_CONTROLLER if the parameter Controller is out of range.⌋ ()

###### Can\_EnableControllerInterrupts

**[SWS\_Can\_00232]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_EnableControllerInterrupts | |
| ***Syntax:*** | void Can\_EnableControllerInterrupts( uint8 Controller) | |
| ***Service ID[hex]:*** | 0x05 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant | |
| ***Parameters (in):*** | Controller | CAN controller for which interrupts shall be re-enabled |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | None | |
| ***Description:*** | This function enables all allowed interrupts. | |
| ***Available via:*** | Can.h | |

**⌋ (SRS\_BSW\_00312)**

[SWS\_Can\_00050] ⌈ The function Can\_EnableControllerInterrupts shall enable all interrupts that must be enabled according the current software status.⌋ (SRS\_Can\_01043) SWS\_Can\_00202 applies to this function. [SWS\_Can\_00208] ⌈ The function Can\_EnableControllerInterrupts shall perform no action when Can\_DisableControllerInterrupts has not been called before.⌋ () See also implementation example for Can\_DisableControllerInterrupts. [SWS\_Can\_00209] ⌈ If development error detection for the Can module is enabled: The function Can\_EnableControllerInterrupts shall raise the error CAN\_E\_UNINIT if the driver not yet initialized.⌋ () [SWS\_Can\_00210] ⌈ If development error detection for the Can module is enabled: The function Can\_EnableControllerInterrupts shall raise the error CAN\_E\_PARAM\_CONTROLLER if the parameter Controller is out of range.⌋ ()

###### Can\_CheckWakeup

**[SWS\_Can\_00360]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_CheckWakeup | |
| ***Syntax:*** | Std\_ReturnType Can\_CheckWakeup( uint8 Controller) | |
| ***Service ID[hex]:*** | 0x0b | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant | |
| ***Parameters (in):*** | Controller | Controller to be checked for a wakeup. |
| ***Parameters***  ***(inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | Std\_ReturnType | E\_OK: API call has been accepted  E\_NOT\_OK: API call has not been accepted |
| ***Description:*** | This function checks if a wakeup has occurred for the given controller. | |
| ***Available via:*** | Can.h | |

[**SWS\_Can\_00361**] ⌈ The function Can\_CheckWakeup shall check if the requested CAN controller has detected a wakeup. If a wakeup event was successfully detected, reporting shall be done to EcuM via API EcuM\_SetWakeupEvent.⌋ ()

[SWS\_Can\_00485] ⌈ The function Can\_CheckWakeup shall be pre compile time configurable On/Off by the configuration parameter: CanWakeupFunctionalityAPI ⌋ ()

[SWS\_Can\_00362] ⌈ If development error detection for the Can module is enabled: The function Can\_CheckWakeup shall raise the error CAN\_E\_UNINIT if the driver is not yet initialized.⌋ ()

[SWS\_Can\_00363] ⌈ If development error detection for the Can module is enabled: The function Can\_CheckWakeup shall raise the error CAN\_E\_PARAM\_CONTROLLER if the parameter Controller is out of range.⌋ ()

###### Can\_GetControllerErrorState

**[SWS\_Can\_91004]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_GetControllerErrorState | |
| ***Syntax:*** | Std\_ReturnType Can\_GetControllerErrorState( uint8 ControllerId, Can\_ErrorStateType\* ErrorStatePtr) | |
| ***Service ID[hex]:*** | 0x11 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant for the same ControllerId | |
| ***Parameters (in):*** | ControllerId | Abstracted CanIf ControllerId which is assigned to a CAN controller, which is requested for ErrorState. |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | ErrorStatePtr | Pointer to a memory location, where the error state of the CAN  controller will be stored. |
| ***Return value:*** | Std\_ReturnType | E\_OK: Error state request has been accepted. E\_NOT\_OK: Error state request has not been accepted. |
| ***Description:*** | This service obtains the error state of the CAN controller. | |
| ***Available via:*** | Can.h | |

**[SWS\_Can\_91005] ⌈** If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can\_GetControllerErrorState shall raise development error CAN\_E\_UNINIT and return E\_NOT\_OK.⌋ (SRS\_BSW\_00406, SRS\_BSW\_00416)

**[SWS\_Can\_91006] ⌈** If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function Can\_GetControllerErrorState shall raise development error CAN\_E\_PARAM\_CONTROLLER and return E\_NOT\_OK.⌋ (SRS\_BSW\_00323)

[**SWS\_Can\_91007] ⌈** If development error detection for the Can module is enabled: if the parameter ErrorStatePtr is a null pointer, the function Can\_GetControllerErrorState shall raise development error CAN\_E\_PARAM\_POINTER and return E\_NOT\_OK.⌋ (SRS\_BSW\_00323)

**[SWS\_Can\_91008] ⌈** When the API Can\_GetControllerErrorState() is called with Can Controller and shall return the error status to upper layer.⌋ (SRS\_CAN\_01167) Controller Id as input parameter then Can driver shall read the error state register of

###### **Can**\_GetControllerMode

**[SWS\_Can\_91014]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_GetControllerMode | |
| ***Syntax:*** | Std\_ReturnType Can\_GetControllerMode( uint8 Controller, Can\_ControllerStateType\* ControllerModePtr  ) | |
| ***Service ID[hex]:*** | 0x12 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant | |
| ***Parameters (in):*** | Controller | CAN controller for which the status shall be requested. |
| ***Parameters***  ***(inout):*** | None | |
| ***Parameters (out):*** | ControllerModePtr | Pointer to a memory location, where the current mode of the CAN controller will be stored. |
| ***Return value:*** | Std\_ReturnType | E\_OK: Controller mode request has been accepted. E\_NOT\_OK: Controller mode request has not been  accepted. |
| ***Description:*** | This service reports about the current status of the requested CAN controller. | |
| ***Available via:*** | Can.h | |

[SWS\_Can\_91015] ⌈ The service Can\_GetControllerMode shall return the mode of the requested CAN controller. ⌋

[SWS\_Can\_91016] ⌈ If development error detection for the Can module is enabled: The function Can\_GetControllerMode shall raise the error CAN\_E\_UNINIT and return E\_NOT\_OK if the driver is not yet initialized.⌋ ( SRS\_BSW\_00406,SRS\_BSW\_00416)

[SWS\_Can\_91017] ⌈ If parameter Controller of Can\_GetControllerMode() has an invalid value, the CanDrv shall report development error code CAN\_E\_PARAM\_CONTROLLER to the Det\_ReportError service of the DET. ⌋ (SRS\_BSW\_00323)

[SWS\_Can\_91018] ⌈ If parameter ControllerModePtr of Can\_GetControllerMode() has an null pointer, the CanDrv shall report development error code

###### Can\_GetControllerRxErrorCounter

**[SWS\_Can\_00511]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_GetControllerRxErrorCounter | |
| ***Syntax:*** | Std\_ReturnType Can\_GetControllerRxErrorCounter( uint8 ControllerId, uint8\* RxErrorCounterPtr) | |
| ***Service ID[hex]:*** | 0x30 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant for the same ControllerId | |
| ***Parameters (in):*** | ControllerId | CAN controller, whose current Rx error counter shall be acquired. |
| ***Parameters***  ***(inout):*** | None | |
| ***Parameters (out):*** | RxErrorCounterPtr | Pointer to a memory location, where the current Rx error counter of the CAN controller will be stored. |
| ***Return value:*** | Std\_ReturnType | E\_OK: Rx error counter available.  E\_NOT\_OK: Wrong ControllerId, or Rx error counter not available. |
| ***Description:*** | Returns the Rx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E\_NOT\_OK would be returned.  Please note that the value of the counter might not be correct at the moment the API returns it, because the Rx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state. | |
| ***Available via:*** | Can.h | |

[SWS\_Can\_00512] ⌈ If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can\_GetControllerRxErrorCounter shall raise development error CAN\_E\_UNINIT and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00406) [SWS\_Can\_00513] ⌈ If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function Can\_GetControllerRxErrorCounter shall raise development error CAN\_E\_PARAM\_CONTROLLER and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00323) [SWS\_Can\_00514] ⌈ If development error detection for the Can module is enabled: if the parameter RxErrorCounterPtr is a null pointer, the function Can\_GetControllerRxErrorCounter shall raise development error CAN\_E\_PARAM\_POINTER and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00323) [SWS\_Can\_00515] ⌈ When the API Can\_GetControllerRxErrorCounter is called with Controller Id as input parameter then Can driver shall read the Rx error counter

###### Can\_GetControllerTxErrorCounter

**[SWS\_Can\_00516]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_GetControllerTxErrorCounter | |
| ***Syntax:*** | Std\_ReturnType Can\_GetControllerTxErrorCounter( uint8 ControllerId,  uint8\* TxErrorCounterPtr  ) | |
| ***Service ID[hex]:*** | 0x31 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Non Reentrant for the same ControllerId | |
| ***Parameters (in):*** | ControllerId | CAN controller, whose current Tx error counter shall be acquired. |
| ***Parameters***  ***(inout):*** | None | |
| ***Parameters (out):*** | TxErrorCounterPtr | Pointer to a memory location, where the current Tx error counter of the CAN controller will be stored. |
| ***Return value:*** | Std\_ReturnType | E\_OK: Tx error counter available.  E\_NOT\_OK: Wrong ControllerId, or Tx error counter not available. |
| ***Description:*** | Returns the Tx error counter for a CAN controller. This value might not be available for all CAN controllers, in which case E\_NOT\_OK would be returned.  Please note that the value of the counter might not be correct at the moment the API returns it, because the Tx counter is handled asynchronously in hardware. Applications should not trust this value for any assumption about the current bus state. | |
| ***Available via:*** | Can.h | |

[SWS\_Can\_00517] ⌈ If development error detection for the Can module is enabled: if the module is not yet initialized, the function Can\_GetControllerTxErrorCounter shall raise development error CAN\_E\_UNINIT and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00406)

[SWS\_Can\_00518] ⌈ If development error detection for the Can module is enabled: if the parameter ControllerId is out of range, the function Can\_GetControllerTxErrorCounter shall raise development error CAN\_E\_PARAM\_CONTROLLER and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00323)

[SWS\_Can\_00519] ⌈ If development error detection for the Can module is enabled: if the parameter TxErrorCounterPtr is a null pointer, the function Can\_GetControllerTxErrorCounter shall raise development error CAN\_E\_PARAM\_POINTER and return E\_NOT\_OK. ⌋ (SRS\_BSW\_00323)

[SWS\_Can\_00520] ⌈ When the API Can\_GetControllerTxErrorCounter is called with Controller Id as input parameter then Can driver shall read the Tx error counter register of Can Controller and shall return the Tx error count to upper layer. ⌋ (SRS\_Can\_01170)

##### Services affecting a Hardware Handle

###### Can\_Write

**[SWS\_Can\_00233]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_Write | |
| ***Syntax:*** | Std\_ReturnType Can\_Write( Can\_HwHandleType Hth, const Can\_PduType\* PduInfo  ) | |
| ***Service ID[hex]:*** | 0x06 | |
| ***Sync/Async:*** | Synchronous | |
| ***Reentrancy:*** | Reentrant (thread-safe) | |
| ***Parameters (in):*** | Hth | information which HW-transmit handle shall be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside one hardware unit. |
| PduInfo | Pointer to SDU user memory, Data Length and Identifier. |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | Std\_ReturnType | E\_OK: Write command has been accepted E\_NOT\_OK: development error occurred  CAN\_BUSY: No TX hardware buffer available or pre-emptive call of Can\_Write that can't be implemented re-entrant (see Can\_ReturnType) |
| ***Description:*** | This function is called by CanIf to pass a CAN message to CanDrv for transmission. | |
| ***Available via:*** | Can.h | |

**⌋ (SRS\_BSW\_00312)**

The function Can\_Write first checks if the hardware transmit object that is identified by the HTH is free and if another Can\_Write is ongoing for the same HTH.

**[SWS\_Can\_00212] ⌈** The function Can\_Write shall perform following actions if the hardware transmit object is free:

* + - * + The mutex for that HTH is set to ‘signaled’
        + The ID, Data Length and SDU are put in a format appropriate for the hardware (if necessary) and copied in the appropriate hardware registers/buffers.
        + All necessary control operations to initiate the transmit are done
        + The mutex for that HTH is released
        + The function returns with E\_OK⌋ (SRS\_Can\_01049)

**[SWS\_Can\_00213] ⌈** The function Can\_Write shall perform no actions if the hardware transmit object is busy with another transmit request for an L-PDU:

1. The transmission of the other L-PDU shall not be cancelled and the function Can\_Write is left without any actions.
2. The function Can\_Write shall return CAN\_BUSY.⌋ (SRS\_Can\_01049).

**[SWS\_Can\_00214] ⌈** The function Can\_Write shall return CAN\_BUSY if a preemptive call of Can\_Write has been issued, that could not be handled reentrant (i.e. a call with the same HTH).⌋ (SRS\_BSW\_00312, SRS\_Can\_01049)

**[SWS\_Can\_00275] ⌈** The function Can\_Write shall be non-blocking.⌋ ()

**[SWS\_Can\_00216] ⌈** If development error detection for the Can module is enabled:

E\_NOT\_OK if the driver is not yet initialized.⌋ () The function Can\_Write shall raise the error CAN\_E\_UNINIT and shall return

**[SWS\_Can\_00217] ⌈** If development error detection for the Can module is enabled: The function Can\_Write shall raise the error CAN\_E\_PARAM\_HANDLE and shall return E\_NOT\_OK if the parameter Hth is not a configured Hardware Transmit Handle.⌋ ()

**[SWS\_Can\_00218] ⌈** The function Can\_Write shall return E\_NOT\_OK and if development error detection for the CAN module is enabled shall raise the error CAN\_E\_PARAM\_DATA\_LENGTH:

* If the length is more than 64 byte.
* If the length is more than 8 byte and the CAN controller is not in CAN FD mode (no CanControllerFdBaudrateConfig).
* If the length is more than 8 byte and the CAN controller is in CAN FD mode (valid CanControllerFdBaudrateConfig), but the CAN FD flag in Can\_PduType->id is not set (refer Can\_IdType).⌋ ( SRS\_Can\_01005)

[**SWS\_Can\_00219**] ⌈ If development error detection for CanDrv is enabled: Can\_Write() shall raise CAN\_E\_PARAM\_POINTER and shall return E\_NOT\_OK if the parameter PduInfo is a null pointer.⌋ ()

**[SWS\_Can\_00503] ⌈** Can\_Write() shall accept a null pointer as SDU (Can\_PduType.Can\_SduPtrType = NULL) if the trigger transmit API is enabled for this hardware object (CanTriggerTransmitEnable = TRUE).⌋ ()

**[SWS\_Can\_00504] ⌈** If the trigger transmit API is enabled for the hardware object, Can\_Write() shall interpret a null pointer as SDU (Can\_PduType.Can\_SduPtrType = NULL) as request for using the trigger transmit interface. If so and the hardware object is free, Can\_Write() shall call CanIf\_TriggerTransmit() with the maximum size of the message buffer to acquire the PDU’s data.⌋ ()

Note: Using the message buffer size allows for late changes of the PDU size, e.g. if a container PDU receives another contained PDU between the call to Can\_Write() and the call of CanIf\_TriggerTransmit().

**[SWS\_Can\_00505] ⌈** If development error detection for CanDrv is enabled: Can\_Write() shall raise CAN\_E\_PARAM\_POINTER and shall return E\_NOT\_OK if the trigger transmit API is disabled for this hardware object (CanTriggerTransmitEnable = FALSE) and the SDU pointer inside PduInfo is a null pointer.⌋ ()

**[SWS\_Can\_00506] ⌈** Can\_Write() shall return E\_NOT\_OK if the trigger transmit API (CanIf\_TriggerTransmit()) returns E\_NOT\_OK.⌋ (SRS\_BSW\_00449, SRS\_BSW\_00357, SRS\_BSW\_00369, SRS\_Can\_01130)

**[SWS\_Can\_00486] ⌈** The CAN Frame has to be sent according to the two most significant bits of Can\_PduType->id. The CAN FD frame bit is only evaluated if CAN Controller is in CAN FD mode (valid CanControllerFdBaudrateConfig).⌋ ()

**[SWS\_Can\_00502] ⌈** If PduInfo->SduLength does not match possible DLC values CanDrv shall use the next higher valid DLC for transmission with initialization of unused bytes to the value of the corresponding CanFdPaddingValue (see ECUC\_Can\_00485).⌋ ( SRS\_Can\_01160)

#### Call-back notifications

This chapter lists all functions provided by the Can module to lower layer modules.

The lower layer module of Can module is the SPI module. The SPI module, which is part of the MCAL, may used to exchange data between the microcontroller and an external CAN controller.

The Can module does not provide callback functions. Only synchronous MCAL API may used to access external CAN controllers.

##### Call-out function

The AUTOSAR CAN module supports optional L-PDU callouts on every reception of a L-PDU.

**[SWS\_Can\_00443]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | <LPDU\_CalloutName> | |
| ***Syntax:*** | boolean <LPDU\_CalloutName>( uint8 Hrh,  Can\_IdType CanId, uint8 CanDataLegth, const uint8\* CanSduPtr  ) | |
| ***Service ID[hex]:*** | 0x20 | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Non Reentrant | |
| ***Parameters (in):*** | Hrh | -- |
| CanId | -- |
| CanDataLegth | -- |
| CanSduPtr | -- |
| ***Parameters*** | None | |
| ***(inout):*** |  | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | boolean | -- |
| ***Description:*** | -- | |
| ***Available via:*** | Can\_Externals.h | |

⌋ ()

where <LPDU\_CalloutName> has to be substituted with the concrete L-PDU callout name which is configurable, see ECUC\_Can\_00434.

**[SWS\_Can\_00444] ⌈** If the L-PDU callout returns false, the L-PDU shall not be processed any further. ⌋ ()

##### Enabling/Disabling wakeup notification

**[SWS\_Can\_00445] ⌈** Can driver shall use the following APIs provided by Icu driver, to enable and disable the wakeup event notification:

* + - * Icu\_EnableNotification
      * Icu\_DisableNotification⌋ ()

**[SWS\_Can\_00446] ⌈** Icu\_EnableNotification shall be called when “external” Can controllers have been transitioned to SLEEP state.⌋ ()

**[SWS\_Can\_00447] ⌈** Icu\_DisableNotification shall be called when “external” Can controllers have been transitioned to STOPPED state.⌋ ()

#### Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non- reentrant.

**[SWS\_Can\_00110] ⌈** There is no requirement regarding the execution order of the CAN main processing functions.⌋ (SRS\_BSW\_00428)

###### Can\_MainFunction\_Write

**[SWS\_Can\_00225]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_MainFunction\_Write |
| ***Syntax:*** | void Can\_MainFunction\_Write( void  ) |
| ***Service ID[hex]:*** | 0x01 |
| ***Description:*** | This function performs the polling of TX confirmation when CAN\_TX\_PROCESSING is set to POLLING. |
| ***Available via:*** | SchM\_Can.h |

[SWS\_Can\_00031] ⌈ The function Can\_MainFunction\_Write shall perform the polling of TX confirmation when CanTxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled.⌋ (SRS\_BSW\_00432, SRS\_BSW\_00373, SRS\_SPAL\_00157)

[SWS\_Can\_00178] ⌈ The Can module may implement the function Can\_MainFunction\_Write as empty define in case no polling at all is used.⌋ ()

[SWS\_Can\_00441] ⌈ If more than one main function period is configured by CanMainFunctionRWPeriods (see ECUC\_Can\_00437), the name of the Can\_MainFunction\_Write() functions shall be

* Can\_MainFunction\_Write\_() for each CanMainFunctionRWPeriods that is referenced by at least one TRANSMIT CanHardwareObject (see ECUC\_Can\_00438). ⌋ ()

###### Can\_MainFunction\_Read

**[SWS\_Can\_00226]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_MainFunction\_Read |
| ***Syntax:*** | void Can\_MainFunction\_Read( void) |
| ***Service ID[hex]:*** | 0x08 |
| ***Description:*** | This function performs the polling of RX indications when CAN\_RX\_PROCESSING is set to POLLING. |
| ***Available via:*** | SchM\_Can.h |

[SWS\_Can\_00108] ⌈ The function Can\_MainFunction\_Read shall perform the polling of RX indications when CanRxProcessing is set to POLLING or MIXED. In case of MIXED processing only the hardware objects for which CanHardwareObjectUsesPolling is set to TRUE shall be polled.⌋ (SRS\_BSW\_00432, SRS\_SPAL\_00157)

[SWS\_Can\_00180] ⌈ The Can module may implement the function Can\_MainFunction\_Read as empty define in case no polling at all is used.⌋ ()

[SWS\_Can\_00442] ⌈ If more than one main function period is configured by CanMainFunctionRWPeriods (see ECUC\_Can\_00437), the name of the Can\_MainFunction\_Read() functions shall be ♣ Can\_MainFunction\_Read\_() for each CanMainFunctionRWPeriods that is referenced by at least one RECEIVE CanHardwareObject (see ECUC\_Can\_00438). ⌋ ()

###### Can\_MainFunction\_BusOff

**[SWS\_Can\_00227]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_MainFunction\_BusOff |
| ***Syntax:*** | void Can\_MainFunction\_BusOff( void) |
| ***Service ID[hex]:*** | 0x09 |
| ***Description:*** | This function performs the polling of bus-off events that are configured statically as 'to be polled'. |
| ***Available via:*** | SchM\_Can.h |

[SWS\_Can\_00109] ⌈ The function Can\_MainFunction\_BusOff shall perform the polling of bus-off events that are configured statically as ‘to be polled’.⌋ () (SRS\_BSW\_00432, SRS\_SPAL\_00157)

[SWS\_Can\_00183] ⌈ The Can module may implement the function Can\_MainFunction\_BusOff as empty define in case no polling at all is used.⌋ ()

###### Can\_MainFunction\_Wakeup

**[SWS\_Can\_00228]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_MainFunction\_Wakeup |
| ***Syntax:*** | void Can\_MainFunction\_Wakeup( void) |
| ***Service ID[hex]:*** | 0x0a |
| ***Description:*** | This function performs the polling of wake-up events that are configured statically as 'to be polled'. |
| ***Available via:*** | SchM\_Can.h |

[SWS\_Can\_00112] ⌈ The function Can\_MainFunction\_Wakeup shall perform the polling of wake-up events that are configured statically as ‘to be polled’.⌋ (SRS\_BSW\_00432, SRS\_SPAL\_00157)

[SWS\_Can\_00185] ⌈ The Can module may implement the function Can\_MainFunction\_Wakeup as empty define in case no polling at all is used.⌋ ()

###### Can\_MainFunction\_Mode

**[SWS\_Can\_00368]** ⌈

|  |  |
| --- | --- |
| ***Service name:*** | Can\_MainFunction\_Mode |
| ***Syntax:*** | void Can\_MainFunction\_Mode( void  ) |
| ***Service ID[hex]:*** | 0x0c |
| ***Description:*** | This function performs the polling of CAN controller mode transitions. |
| ***Available via:*** | SchM\_Can.h |

[SWS\_Can\_00369] ⌈ The function Can\_MainFunction\_Mode shall implement the polling of CAN status register flags to detect transition of CAN Controller state. Compare to chapter 7.3.2.⌋ ()

#### Expected Interfaces

In this chapter all interfaces required from other modules are listed.

##### Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module. All callback functions that are called by the Can module are implemented in the CanIf module. These callback functions are not configurable.

**[SWS\_Can\_00234]** ⌈

|  |  |  |
| --- | --- | --- |
| ***API function*** | ***Header File*** | ***Description*** |
| CanIf\_ControllerBusOff | CanIf\_Can.h | This service indicates a Controller BusOff event referring to the corresponding CAN Controller with the abstract CanIf ControllerId. |
| CanIf\_ControllerModeIndication | CanIf\_Can.h | This service indicates a controller state transition referring to the corresponding CAN controller with the abstract CanIf ControllerId. |
| CanIf\_RxIndication | CanIf\_Can.h | This service indicates a successful reception of a received CAN Rx L-PDU to the CanIf after passing all filters and validation checks. |
| CanIf\_TxConfirmation | CanIf\_Can.h | This service confirms a previously successfully processed transmission of a CAN TxPDU. |
| Det\_ReportRuntimeError | Det.h | Service to report runtime errors. If a callout has been  configured then this callout shall be called. |
| GetCounterValue | Os.h | This service reads the current count value of a counter (returning either the hardware timer ticks if counter is driven by hardware or the software ticks when user drives counter). |

**⌋ (SRS\_Can\_01055)**

##### Optional Interfaces

This chapter defines all interfaces that are required to fulfill an optional functionality of the module.

**[SWS\_Can\_00235]** ⌈

|  |  |  |
| --- | --- | --- |
| API function | Header File | Description |
| CanIf\_CurrentIcomConfiguration | CanIf\_Can.h | This service shall inform about the change of the Icom Configuration of a CAN controller using the abstract CanIf ControllerId. |
| CanIf\_TriggerTransmit | CanIf.h | Within this API, the upper layer module (called module) shall check whether the available data fits |
|  |  | into the buffer size reported by PduInfoPtr-  >SduLength.  If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr-  >SduLength.  If not, it returns E\_NOT\_OK without changing PduInfoPtr. |
| Det\_ReportError | Det.h | Service to report development errors. |
| EcuM\_CheckWakeup | EcuM\_Externals.h | This callout is called by the EcuM to poll a wakeup source. It shall also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the  same interrupt. |
| EcuM\_SetWakeupEvent | EcuM.h | Sets the wakeup event. |
| Icu\_DisableNotification | Icu.h | This function disables the notification of a channel. |
| Icu\_EnableNotification | Icu.h | This function enables the notification on the given channel. |

⌋ (SRS\_SPAL\_12056, SRS\_Can\_01054)

##### Configurable interfaces

There is no configurable target for the Can module. The Can module always reports to CanIf module.

#### API supporting Pretended Networking

**[SWS\_CAN\_91001]** ⌈

|  |  |  |
| --- | --- | --- |
| ***Service name:*** | Can\_SetIcomConfiguration | |
| ***Syntax:*** | Std\_ReturnType Can\_SetIcomConfiguration( uint8 Controller,  IcomConfigIdType ConfigurationId  ) | |
| ***Service ID[hex]:*** | 0x21 | |
| ***Sync/Async:*** | Asynchronous | |
| ***Reentrancy:*** | Reentrant only for different controller Ids | |
| ***Parameters (in):*** | Controller | CAN controller for which the status shall be changed. |
| ConfigurationId | Requested Configuration |
| ***Parameters (inout):*** | None | |
| ***Parameters (out):*** | None | |
| ***Return value:*** | Std\_ReturnType | E\_OK: CAN driver succeeded in setting a configuration with a valid Configuration id.  E\_NOT\_OK: CAN driver failed to set a configuration with a valid Configuration id. |
| ***Description:*** | This service shall change the Icom Configuration of a CAN controller to the requested one. | |
| ***Available via:*** | Can.h | |

⌋ ()

**[SWS\_Can\_00480] ⌈** The interface Can\_SetIcomConfiguration() shall activate or deactivate Pretended Networking and load the requested ICOM configuration for a given controller.⌋ ()

**[SWS\_Can\_00481] ⌈** If the requested ConfigurationId is not 0, the function Can\_SetIcomConfiguration() shall reconfigure the controller with the ICOM configuration parameters of the CanIcomConfig container which CanIcomConfigId matches the requested ConfigurationId.⌋ ()

**[SWS\_Can\_00495] ⌈** Can\_SetIcomConfiguration() shall be pre compile time configurable ON/OFF by the configuration parameter CAN\_PUBLIC\_ICOM\_SUPPORT.⌋ ()

**[SWS\_Can\_00475] ⌈** If development error detection for CanDrv is enabled, then function Can\_SetIcomConfiguration() shall report the development error CAN\_E\_ICOM\_CONFIG\_INVALID if it is called with an invalid ConfigurationId (i.e. neither 0 nor any of the configured CanIcomConfigId).⌋ ()

### Sequence diagrams

#### Interaction between Can and CanIf module

For sequence diagrams see the CanIf module Specification [[5]](#_bookmark8).

There are described the sequences for Transmission, Reception and Error Handling.

#### Wakeup sequence

For Wakeup sequence diagrams refer to Specification of ECU State Manager [[7]](#_bookmark9).

### Configuration specification

This chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the Can module.

Chapter 10.3 specifies published information of the Can module.

#### How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS\_BSWGeneral*

#### Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters [7](#_bookmark25) and Chapter [8](#_bookmark60).

The described parameters are input for the Can module configurator.

**[SWS\_Can\_00022] ⌈** The code configuration of the Can module is CAN controller specific. If the CAN controller is sited on-chip, the code generation tool for the Can module is µController specific. If the CAN controller is an external device, the generation tool must not be µController specific.⌋ (SRS\_BSW\_00159)

**[SWS\_Can\_00024] ⌈** The valid values that can be configured are hardware dependent. Therefore the rules and constraints can’t be given in the standard. The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (i.e. Port driver, MCU driver etc.)⌋

(SRS\_BSW\_00167, SRS\_SPAL\_12463)

**[SWS\_Can\_00507] DRAFT** ⌈ The Can Driver module shall reject configurations with partition mappings which are not supported by the implementation. ⌋ ()



+container

+destination

+reference

+subContainer

lowerMultiplicity = 0 upperMultiplicity = 1

upperMultiplicity = \* lowerMultiplicity = 1

CanIcom:

EcucParamConfContainerDef

CanControllerBaudrateConfig: EcucParamConfContainerDef

+subContainer

upperMultiplicity = \* lowerMultiplicity = 1

upperMultiplicity = \* lowerMultiplicity = 1

upperMultiplicity = 1 lowerMultiplicity = 1

CanControllerRef:

EcucReferenceDef

CanController: EcucParamConfContainerDef

CanHardwareObject:

EcucParamConfContainerDef

CanGeneral: EcucParamConfContainerDef

+subContainer

+subContainer

CanConfigSet: EcucParamConfContainerDef

+container

upperMultiplicity = \* lowerMultiplicity = 0

Can: EcucModuleDef

**Figure 10-1: Can Module Configuration Layout**



Can: EcucModuleDef

CanConfigSet: EcucParamConfContainerDef

+container

upperMultiplicity = \* lowerMultiplicity = 0

+subContainer

CanControllerId:

EcucIntegerParamDef

upperMultiplicity = 1 lowerMultiplicity = 1

symbolicNameValue = true min = 0

max = 255

+parameter

CanController:

EcucParamConfContainerDef

upperMultiplicity = \* lowerMultiplicity = 1

+reference

CanWakeupSourceRef:

EcucReferenceDef

EcuMWakeupSource:

+destination EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = 1

requiresSymbolicNameValue = true

lowerMultiplicity = 1 upperMultiplicity = 32

CanControllerActivation: EcucBooleanParamDef

+parameter

+reference

CanCpuClockRef:

EcucReferenceDef

+destination

McuClockReferencePoint:

EcucParamConfContainerDef

upperMultiplicity = \* lowerMultiplicity = 1

CanControllerBaseAddress: EcucIntegerParamDef

+parameter

CanControllerBaudrateConfig:

+subContainer EcucParamConfContainerDef

min = 0 upperMultiplicity = \*

max = 4294967295 lowerMultiplicity = 1

+destination

CanWakeupSupport: EcucBooleanParamDef

+parameter

CanControllerDefaultBaudrate:

+reference

EcucReferenceDef

CanWakeupFunctionalityAPI: +parameter

EcucBooleanParamDef

defaultValue = false

MIXED:

+literal EcucEnumerationLiteralDef

+parameter

CanTxProcessing:

EcucEnumerationParamDef

+literal

+literal

+parameter

CanRxProcessing:

EcucEnumerationParamDef

INTERRUPT:

+literal EcucEnumerationLiteralDef

+literal

+parameter CanBusoffProcessing:

EcucEnumerationParamDef

+literal

+literal

POLLING:

+parameter

+literal EcucEnumerationLiteralDef

+literal

CanWakeupProcessing:

EcucEnumerationParamDef

+literal

+reference CanControllerEcucPartitionRef: +destination

EcucReferenceDef

lowerMultiplicity = 0 upperMultiplicity = 1

EcucPartition:

EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

**Figure 10-2: Can Controller Configuration Layout**



+parameter

+parameter

+parameter

+parameter

+subContainer

lowerMultiplicity = 0 upperMultiplicity = 1

CanControllerFdBaudrateConfig:

EcucParamConfContainerDef

min = 0

max = 65535 defaultValue = 0

CanControllerBaudRateConfigID:

EcucIntegerParamDef

min = 0 max = 255

CanControllerSyncJumpWidth:

EcucIntegerParamDef

min = 0 max = 255

+parameter

CanControllerSeg2:

EcucIntegerParamDef

min = 0 max = 255

CanControllerSeg1:

EcucIntegerParamDef

min = 0 max = 255

+parameter

CanControllerPropSeg:

EcucIntegerParamDef

min = 0 max = 2000

upperMultiplicity = \* lowerMultiplicity = 1

CanControllerBaudRate:

EcucFloatParamDef

CanControllerBaudrateConfig: EcucParamConfContainerDef

**Figure 10-3: Can Controller Baud Rate Configuration Layout**



+container

+parameter

+parameter

+parameter

+parameter

+parameter

+parameter

+parameter

+subContaine

+reference

+destination

+parameter

+parameter

+reference

+destination

+subContainer

+parameter

defaultValue = false

CanPublicIcomSupport: EcucBooleanParamDef

lowerMultiplicity = 0 upperMultiplicity = 1

lowerMultiplicity = 0 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

CanIcomGeneral: EcucParamConfContainerDef

CanEcucPartitionRef:

EcucReferenceDef

EcucPartition:

EcucParamConfContainerDef

defaultValue = False lowerMultiplicity = 0 upperMultiplicity = 1

upperMultiplicity = 1 lowerMultiplicity = 0

CanSetBaudrateApi: EcucBooleanParamDef

CanLPduReceiveCalloutFunction:

EcucFunctionNameDef

lowerMultiplicity = 0 upperMultiplicity = 1 min = 0

max = INF

lowerMultiplicity = 0 upperMultiplicity = 1

lowerMultiplicity = 0 upperMultiplicity = \*

+parameter

CanMainFunctionBusoffPeriod: EcucFloatParamDef

CanOsCounterRef:

EcucReferenceDef

OsCounter:

EcucParamConfContainerDef

min = 0

max = INF

CanMainFunctionModePeriod: EcucFloatParamDef

+parameter

lowerMultiplicity = 0 upperMultiplicity = 1 min = 0

max = INF

CanMainFunctionWakeupPeriod: EcucFloatParamDef

min = 0

max = INF

r

lowerMultiplicity = 0 upperMultiplicity = \*

CanMainFunctionPeriod:

EcucFloatParamDef

min = 0.000001 max = 65.535

CanMainFunctionRWPeriods:

EcucParamConfContainerDef

CanTimeoutDuration: EcucFloatParamDef

defaultValue = false

CanVersionInfoApi:

EcucBooleanParamDef

CanMultiplexedTransmission: EcucBooleanParamDef

defaultValue = false

upperMultiplicity = 1 lowerMultiplicity = 1

min = 0

max = 255

CanGeneral: EcucParamConfContainerDef

CanDevErrorDetect: EcucBooleanParamDef

CanIndex: EcucIntegerParamDef

upperMultiplicity = \* lowerMultiplicity = 0

Can: EcucModuleDef

**Figure 10-4: Can General Configuration Layout**



+parameter

+parameter

+parameter

+parameter

min = 0 max = 255

lowerMultiplicity = 0 upperMultiplicity = 1

CanControllerSspOffset: EcucIntegerParamDef

min = 0 max = 255

CanControllerSyncJumpWidth:

EcucIntegerParamDef

+parameter

min = 0 max = 255

CanControllerSeg2: EcucIntegerParamDef

min = 0 max = 255

CanControllerSeg1:

EcucIntegerParamDef

+parameter

min = 0 max = 255

CanControllerPropSeg: EcucIntegerParamDef

defaultValue = true

+parameter

CanControllerTxBitRateSwitch:

EcucBooleanParamDef

min = 0

max = 16000

lowerMultiplicity = 0 upperMultiplicity = 1

CanControllerFdBaudRate: EcucFloatParamDef

CanControllerFdBaudrateConfig: EcucParamConfContainerDef

**Figure 10-5: CanControllerFdBaudrateConfig**



BASIC:

EcucEnumerationLiteralDef

FULL:

EcucEnumerationLiteralDef

TRANSMIT:

EcucEnumerationLiteralDef

RECEIVE:

EcucEnumerationLiteralDef

+literal

+literal

+literal

+literal

+container CanObjectId:

EcucIntegerParamDef

CanHandleType: EcucEnumerationParamDef

CanConfigSet: EcucParamConfContainerDef

+parameter

upperMultiplicity = 1 lowerMultiplicity = 1

symbolicNameValue = true min = 0

max = 65535

CanObjectType:

EcucEnumerationParamDef

+parameter

+subContainer

+parameter

CanHardwareObject: EcucParamConfContainerDef

upperMultiplicity = \* lowerMultiplicity = 1

+parameter

+reference

+parameter

+reference

CanFdPaddingValue: EcucIntegerParamDef

CanMainFunctionRWPeriodRef:

EcucReferenceDef

lowerMultiplicity = 0 upperMultiplicity = 1

min = 0

max = 255 defaultValue = 0

lowerMultiplicity = 0 upperMultiplicity = 1

CanControllerRef:

EcucReferenceDef

CanIdType: EcucEnumerationParamDef

+literal

+destination

+subContainer

+destination

STANDARD:

EcucEnumerationLiteralDef

CanMainFunctionRWPeriods: EcucParamConfContainerDef

CanHwFilter: EcucParamConfContainerDef

CanController: EcucParamConfContainerDef

+literal

lowerMultiplicity = 0 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = \*

upperMultiplicity = \* lowerMultiplicity = 1

EXTENDED:

EcucEnumerationLiteralDef

+literal

+parameter

+parameter

MIXED:

EcucEnumerationLiteralDef

CanHwFilterMask: CanHwFilterCode:

EcucIntegerParamDef EcucIntegerParamDef

min = 0

max = 4294967295

min = 0

max = 4294967295

+parameter

CanTriggerTransmitEnable:

EcucBooleanParamDef

lowerMultiplicity = 0 upperMultiplicity = 1 defaultValue = false

+parameter +parameter

CanHardwareObjectUsesPolling:

EcucBooleanParamDef

defaultValue = false lowerMultiplicity = 0 upperMultiplicity = 1

CanHwObjectCount:

EcucIntegerParamDef

min = 1

max = 65535 defaultValue = 1

upperMultiplicity = \* lowerMultiplicity = 0

Can: EcucModuleDef

**Figure 10-6: Can Hardware Object Configuration Layout**



+parameter

+subContainer

+parameter

+subContainer

CanIcomWakeupCauses:

EcucParamConfContainerDef

defaultValue = true

CanIcomWakeOnBusOff:

EcucBooleanParamDef

min = 1 max = 255

lowerMultiplicity = 1 upperMultiplicity = \*

lowerMultiplicity = 0 upperMultiplicity = 1

CanIcomConfigId:

EcucIntegerParamDef

CanIcomConfig: EcucParamConfContainerDef

CanIcom: EcucParamConfContainerDef

**Figure 10-7: CanICOM Layout**



+literal

+parameter

+literal

+literal

+parameter

+literal

+literal

CAN\_ICOM\_VARIANT\_HW:

EcucEnumerationLiteralDef

CAN\_ICOM\_VARIANT\_SW:

EcucEnumerationLiteralDef

defaultValue = CAN\_ICOM\_VARIANT\_NONE

CAN\_ICOM\_VARIANT\_NONE:

EcucEnumerationLiteralDef

CanIcomVariant: EcucEnumerationParamDef

CAN\_ICOM\_LEVEL\_TWO:

EcucEnumerationLiteralDef

lowerMultiplicity = 0 upperMultiplicity = 1

defaultValue = CAN\_ICOM\_LEVEL\_ONE lowerMultiplicity = 0

upperMultiplicity = 1

CAN\_ICOM\_LEVEL\_ONE:

EcucEnumerationLiteralDef

CanIcomLevel: EcucEnumerationParamDef

CanIcomGeneral: EcucParamConfContainerDef

**Figure 10-8: CanICOM General Configuration Layout**

+parameter



SMALLER:

EcucEnumerationLiteralDef

CanIcomSignalOperation: EcucEnumerationParamDef

XOR:

EcucEnumerationLiteralDef

AND:

EcucEnumerationLiteralDef

EQUAL:

EcucEnumerationLiteralDef

min = 0

CanIcomSignalMask: EcucIntegerParamDef

lowerMultiplicity = 0 upperMultiplicity = \*

CanIcomRxMessageSignalConfig: EcucParamConfContainerDef

+parameter

min = 0

CanIcomSignalValue: EcucIntegerParamDef

+literal

+parameter

+literal

+literal

GREATER:

EcucEnumerationLiteralDef

+literal

+literal

+reference +destination

lowerMultiplicity = 0 upperMultiplicity = 1

CanIcomSignalRef: EcucReferenceDef

ComSignal:

EcucParamConfContainerDef

lowerMultiplicity = 0 upperMultiplicity = \*

**Figure 10-9: CanIcomRxMessageSignal Configuration Layout**



+parameter

+parameter

+parameter

+subContainer

+parameter

+parameter

+subContainer

lowerMultiplicity = 0 upperMultiplicity = \*

CanIcomRxMessageSignalConfig:

EcucParamConfContainerDef

defaultValue = false

CanIcomPayloadLengthError:

EcucBooleanParamDef

lowerMultiplicity = 0 upperMultiplicity = 1

CanIcomMissingMessageTimerValue:

EcucFloatParamDef

min = 1

max = 65536

lowerMultiplicity = 0 upperMultiplicity = 1

CanIcomCounterValue:

EcucIntegerParamDef

min = 0

max = 536870912

lowerMultiplicity = 0 upperMultiplicity = 1

CanIcomMessageIdMask:

EcucIntegerParamDef

min = 0

max = 536870912

lowerMultiplicity = 1 upperMultiplicity = 1

lowerMultiplicity = 1 upperMultiplicity = \*

CanIcomMessageId:

EcucIntegerParamDef

CanIcomRxMessage: EcucParamConfContainerDef

CanIcomWakeupCauses: EcucParamConfContainerDef

**Figure 10-10: CanIcomWakeupCauses Configuration Layout**

##### Can

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00489 :** |
| ***Module Name*** | *Can* |
| ***Module Description*** | This container holds the configuration of a single CAN Driver. |
| ***Post-Build Variant Support*** | true |
| ***Supported Config Variants*** | VARIANT-POST-BUILD, VARIANT-PRE-COMPILE |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanConfigSet | 1 | This container contains the configuration parameters and sub  containers of the AUTOSAR Can module. |
| CanGeneral | 1 | This container contains the parameters related each CAN Driver Unit. |

##### CanGeneral

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00328 :** |
| ***Container Name*** | CanGeneral |
| ***Description*** | This container contains the parameters related each CAN Driver Unit. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00064 :** | | |
| ***Name*** | CanDevErrorDetect | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | Switches the development error detection and notification on or off.   * true: detection and notification is enabled. * false: detection and notification is disabled. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00320 :** | | | |
| ***Name*** | CanIndex | | | |
| ***Parent Container*** | CanGeneral | | | |
| ***Description*** | Specifies the InstanceId of this module instance. If only one instance is present it shall have the Id 0. | | | |
| ***Multiplicity*** | 1 | | | |
| ***Type*** | EcucIntegerParamDef | | | |
| ***Range*** | 0 .. 255 | |  | |
| ***Default value*** | -- | | | |
| ***Post-Build Variant Value*** | false | | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | | All Variants |
| ***Link time*** | -- | |  |
| ***Post-build time*** | -- | |  |
| ***Scope / Dependency*** | scope: local | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00434 :** | | |
| ***Name*** | CanLPduReceiveCalloutFunction | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | This parameter defines the existence and the name of a callout function that is called after a successful  reception of a received CAN Rx L-PDU. If this parameter is omitted no callout shall take place. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucFunctionNameDef | | |
| ***Default value*** | -- | | |
| ***maxLength*** | -- | | |
| ***minLength*** | -- | | |
| ***regularExpression*** | -- | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00355 :** | | |
| ***Name*** | CanMainFunctionBusoffPeriod | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | This parameter describes the period for cyclic call to Can\_MainFunction\_Busoff. Unit is seconds. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | ]0 .. INF[ |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00376 :** | | |
| ***Name*** | CanMainFunctionModePeriod | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | This parameter describes the period for cyclic call to Can\_MainFunction\_Mode. Unit is seconds. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | ]0 .. INF[ |  | |
| ***Default value*** | -- |  | |
| ***Post-Build Variant Value*** | false |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00095 :** | | |
| ***Name*** | CanMultiplexedTransmission | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | Specifies if multiplexed transmission shall be supported.ON or OFF | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU  dependency: CAN Hardware Unit supports multiplexed transmission | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00483 :** | | |
| ***Name*** | CanPublicIcomSupport | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | Selects support of Pretended Network features in Can driver. True: Enabled  False: Disabled | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00482 :** | | |
| ***Name*** | CanSetBaudrateApi | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | The support of the Can\_SetBaudrate API is optional.  If this parameter is set to true the Can\_SetBaudrate API shall be supported. Otherwise the API is not supported. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00113 :** | | |
| ***Name*** | CanTimeoutDuration | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | Specifies the maximum time for blocking function until a timeout is detected. Unit is seconds. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | [1E-6 .. 65.535] |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00106 :** | | |
| ***Name*** | CanVersionInfoApi | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | Switches the Can\_GetVersionInfo() API ON or OFF. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00491 :** |
| ***Name*** | CanEcucPartitionRef |
| ***Parent Container*** | CanGeneral |
| ***Description*** | Maps the CAN driver to zero or multiple ECUC partitions to make the |

|  |  |  |  |
| --- | --- | --- | --- |
|  | modules API available in this partition. The CAN driver will operate as an independent instance in each of the partitions.  **Tags:**  atp.Status=draft | | |
| ***Multiplicity*** | 0..\* | | |
| ***Type*** | Reference to [ EcucPartition ] | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00431 :** | | |
| ***Name*** | CanOsCounterRef | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | This parameter contains a reference to the OsCounter, which is used by the CAN driver. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | Reference to [ OsCounter ] | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00430 :** | | |
| ***Name*** | CanSupportTTCANRef | | |
| ***Parent Container*** | CanGeneral | | |
| ***Description*** | The parameter refers to CanIfSupportTTCAN parameter in the CAN Interface Module configuration.  The CanIfSupportTTCAN parameter defines whether TTCAN is supported. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | Reference to [ CanIfPrivateCfg ] | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanIcomGeneral | 0..1 | This container contains the general configuration parameters of the ICOM Configuration. |
| CanMainFunctionRWPeriods | 0..\* | This container contains the parameter for configuring the period for cyclic call to Can\_MainFunction\_Read or |

|  |  |  |
| --- | --- | --- |
|  |  | Can\_MainFunction\_Write depending on the referring item. |

**[SWS\_Can\_CONSTR\_00508] DRAFT** ⌈ The module will operate as an independent instance in each of the partitions, means the called API will only target the partition it

is called in. ⌋ ()

##### CanController

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00354 :** |
| ***Container Name*** | CanController |
| ***Description*** | This container contains the configuration parameters of the CAN controller(s). |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00314 :** | | |
| ***Name*** | CanBusoffProcessing | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Enables / disables API Can\_MainFunction\_BusOff() for handling busoff events in polling mode. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | INTERRUPT | Interrupt Mode of operation. | |
| POLLING | Polling Mode of operation. | |
| ***Post-Build Variant***  ***Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00315 :** | | |
| ***Name*** | CanControllerActivation | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Defines if a CAN controller is used in the configuration. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00382 :** | |
| ***Name*** | CanControllerBaseAddress | |
| ***Parent Container*** | CanController | |
| ***Description*** | Specifies the CAN controller base address. | |
| ***Multiplicity*** | 1 | |
| ***Type*** | EcucIntegerParamDef | |
| ***Range*** | 0 .. 4294967295 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Default value*** | -- |  | |
| ***Post-Build Variant Value*** | false |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00316 :** | | |
| ***Name*** | CanControllerId | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | This parameter provides the controller ID which is unique in a given CAN Driver. The value for this parameter starts with 0 and continue without any gaps. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00317 :** | | |
| ***Name*** | CanRxProcessing | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Enables / disables API Can\_MainFunction\_Read() for handling PDU reception events in polling mode. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | INTERRUPT | Interrupt Mode of operation. | |
| MIXED | Mixed Mode of operation | |
| POLLING | Polling Mode of operation. | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00318 :** | | |
| ***Name*** | CanTxProcessing | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Enables / disables API Can\_MainFunction\_Write() for handling PDU transmission events in polling mode. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | INTERRUPT | Interrupt Mode of operation. | |
| MIXED | Mixed Mode of operation | |
| POLLING | Polling Mode of operation. | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class*** | ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00466 :** | | |
| ***Name*** | CanWakeupFunctionalityAPI | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Adds / removes the service Can\_CheckWakeup() from the code. True: Can\_CheckWakeup can be used.  False: Can\_CheckWakeup cannot be used. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local  dependency: H/W should support the wakeup functionality to enable this parameter. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00319 :** | | |
| ***Name*** | CanWakeupProcessing | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Enables / disables API Can\_MainFunction\_Wakeup() for handling wakeup events in polling mode. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | INTERRUPT | Interrupt Mode of operation. | |
| POLLING | Polling Mode of operation. | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00330 :** | | |
| ***Name*** | CanWakeupSupport | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | CAN driver support for wakeup over CAN Bus. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** |  | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00435 :** |
| ***Name*** | CanControllerDefaultBaudrate |
| ***Parent Container*** | CanController |
| ***Description*** | Reference to baudrate configuration container configured for the Can |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Controller. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | Reference to [ CanControllerBaudrateConfig ] | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00492 :** | | |
| ***Name*** | CanControllerEcucPartitionRef | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Maps the CAN controller to zero or one ECUC partitions. The ECUC partition referenced is a subset of the ECUC partitions where the CAN driver is mapped to.  **Tags:**  atp.Status=draft | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | Reference to [ EcucPartition ] | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00313 :** | | |
| ***Name*** | CanCpuClockRef | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | Reference to the CPU clock configuration, which is set in the MCU driver configuration | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | Reference to [ McuClockReferencePoint ] | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00359 :** | | |
| ***Name*** | CanWakeupSourceRef | | |
| ***Parent Container*** | CanController | | |
| ***Description*** | This parameter contains a reference to the Wakeup Source for this controller as defined in the ECU State Manager.  Implementation Type: reference to EcuM\_WakeupSourceType | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | Symbolic name reference to [ EcuMWakeupSource ] | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration*** | ***Pre-compile time*** | X | All Variants |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class*** | ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local |  | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanControllerBaudrateConfi g | 1..\* | This container contains bit timing related configuration parameters of the CAN controller(s). |
| CanTTController | 0..1 | CanTTController is specified in the SWS TTCAN and contains the configuration parameters of the TTCAN controller(s) (which are needed in addition to the configuration parameters of the CAN controller(s)).  This container is only included and valid if TTCAN is supported by the controller, enabled (see CanSupportTTCANRef, ECUC\_Can\_00430), and used. |

**[SWS\_Can\_CONSTR\_00509] DRAFT** ⌈ The ECUC partitions referenced by CanControllerEcucPartitionRef shall be a subset of the ECUC partitions referenced

by CanEcucPartitionRef. ⌋ ()

communication channel shall all reference the same ECUC partition. ⌋ () **[SWS\_Can\_CONSTR\_00510] DRAFT** ⌈ CanController and CanTrcvChannel of one

##### CanControllerBaudrateConfig

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00387 :** |
| ***Container Name*** | CanControllerBaudrateConfig |
| ***Description*** | This container contains bit timing related configuration parameters of the CAN controller(s). |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00005 :** | | |
| ***Name*** | CanControllerBaudRate | | |
| ***Parent Container*** | CanControllerBaudrateConfig | | |
| ***Description*** | Specifies the baudrate of the controller in kbps. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | [0 .. 2000] |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00471 :** |
| ***Name*** | CanControllerBaudRateConfigID |
| ***Parent Container*** | CanControllerBaudrateConfig |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Description*** | Uniquely identifies a specific baud rate configuration. This ID is used by SetBaudrate API. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 65535 |  | |
| ***Default value*** | 0 | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU  dependency: CanSetBaudrateApi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00073 :** |  | |
| ***Name*** | CanControllerPropSeg |  | |
| ***Parent Container*** | CanControllerBaudrateConfi | g | |
| ***Description*** | Specifies propagation delay | in time quantas. | |
| ***Multiplicity*** | 1 |  | |
| ***Type*** | EcucIntegerParamDef |  | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- |  | |
| ***Post-Build Variant Value*** | true |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00074 :** | | |
| ***Name*** | CanControllerSeg1 | | |
| ***Parent Container*** | CanControllerBaudrateConfig | | |
| ***Description*** | Specifies phase segment 1 in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00075 :** | | |
| ***Name*** | CanControllerSeg2 | | |
| ***Parent Container*** | CanControllerBaudrateConfig | | |
| ***Description*** | Specifies phase segment 2 in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00383 :** | | |
| ***Name*** | CanControllerSyncJumpWidth | | |
| ***Parent Container*** | CanControllerBaudrateConfig | | |
| ***Description*** | Specifies the synchronization jump width for the controller in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanControllerFdBaudrateConfi g | 0..1 | This optional container contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN FD frame. If this container  exists the controller supports CAN FD frames. |

##### CanControllerFdBaudrateConfig

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00473 :** |
| ***Container Name*** | CanControllerFdBaudrateConfig |
| ***Description*** | This optional container contains bit timing related configuration parameters of the CAN controller(s) for payload and CRC of a CAN FD frame. If this container exists the controller supports CAN FD frames. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00481 :** | | |
| ***Name*** | CanControllerFdBaudRate | | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | | |
| ***Description*** | Specifies the data segment baud rate of the controller in kbps. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | [0 .. 16000] |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00476 :** | |
| ***Name*** | CanControllerPropSeg | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | |
| ***Description*** | Specifies propagation delay in time quantas. | |
| ***Multiplicity*** | 1 | |
| ***Type*** | EcucIntegerParamDef | |
| ***Range*** | 0 .. 255 |  |
| ***Default value*** | -- | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Post-Build Variant Value*** | true |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00477 :** | | |
| ***Name*** | CanControllerSeg1 | | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | | |
| ***Description*** | Specifies phase segment 1 in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00478 :** | | |
| ***Name*** | CanControllerSeg2 | | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | | |
| ***Description*** | Specifies phase segment 2 in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00494 :** |
| ***Name*** | CanControllerSspOffset |
| ***Parent Container*** | CanControllerFdBaudrateConfig |
| ***Description*** | Specifies the Transmitter Delay Compensation Offset in minimum time quanta (see [17]). Transmitter Delay Compensation Offset is used to adjust the position of the Secondary Sample Point (SSP), relative to the beginning of the received bit. If this parameter is configured, the Transmitter Delay Compensation is done by measurement of the CAN controller. If not specified, Transmitter Delay Compensation is disabled.  Note:  MTQ == Minimum Time Quanta in seconds == 1/(frequency of the CAN controller clock)  Secondary Sample Point Offset in seconds = CanControllerSspOffset \* MTQ  Example:  CAN controller clock frequency = 20MHz => MTQ = 1/20 \* 10^(-6) s = 0,05 us = 50ns  Baud rate = 1MBit/s => BitTime = 1/(1 \* 10^6) s/Bit = 1 \* 10^(-6) = 1us/Bit SSP = 75% => SSP in seconds = 0,75 \* 1us = 750 ns CanControllerSspOffset in MTQ = 750ns / 50ns = 15 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Note: Please consider the minimum range (0..63) stated in [17] and the range definition (0..127) used as per [19]. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00479 :** | | |
| ***Name*** | CanControllerSyncJumpWidth | | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | | |
| ***Description*** | Specifies the synchronization jump width for the controller in time quantas. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00475 :** | | |
| ***Name*** | CanControllerTxBitRateSwitch | | |
| ***Parent Container*** | CanControllerFdBaudrateConfig | | |
| ***Description*** | Specifies if the bit rate switching shall be used for transmissions. If FALSE: CAN FD frames shall be sent without bit rate switching. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

***No Included Containers***

##### CanHardwareObject

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00324 :** |
| ***Container Name*** | CanHardwareObject |

|  |  |
| --- | --- |
| ***Description*** | This container contains the configuration (parameters) of CAN Hardware Objects. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00485 :** | | |
| ***Name*** | CanFdPaddingValue | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Specifies the value which is used to pad unspecified data in CAN FD frames > 8 bytes for transmission. This is necessary due to the discrete possible values of the DLC if > 8 bytes.  If the length of a PDU which was requested to be sent does not match the allowed DLC values, the remaining bytes up to the next possible value  shall be padded with this value. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 255 |  | |
| ***Default value*** | 0 | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00323 :** | | |
| ***Name*** | CanHandleType | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Specifies the type (Full-CAN or Basic-CAN) of a hardware object. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | BASIC | For several L-PDUs are hadled by the hardware object | |
| FULL | For only one L-PDU (identifier) is handled by the hardware object | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU  dependency: This configuration element is used as information for the CAN Interface only. The relevant CAN driver configuration is done with the filter mask  and identifier. | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00490 :** |
| ***Name*** | CanHardwareObjectUsesPolling |
| ***Parent Container*** | CanHardwareObject |
| ***Description*** | Enables polling of this hardware object. |
| ***Multiplicity*** | 0..1 |
| ***Type*** | EcucBooleanParamDef |
| ***Default value*** | false |

|  |  |
| --- | --- |
| ***Scope / Dependency*** | dependency: This parameter shall exist if CanRxProcessing/CanTxProcessing is set to Mixed. |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00467 :** | | |
| ***Name*** | CanHwObjectCount | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Number of hardware objects used to implement one HOH. In case of a HRH this parameter defines the number of elements in the hardware FIFO or the number of shadow buffers, in case of a HTH it defines the number of hardware objects used for multiplexed transmission or for a hardware FIFO  used by a FullCAN HTH. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 1 .. 65535 |  | |
| ***Default value*** | 1 | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00065 :** |  | |
| ***Name*** | CanIdType |  | |
| ***Parent Container*** | CanHardwareObject |  | |
| ***Description*** | Specifies whether the IdValue is of type   * standard identifier * extended identifier * mixed mode ImplementationType: Can\_IdType |  | |
| ***Multiplicity*** | 1 |  | |
| ***Type*** | EcucEnumerationParamDef |  | |
| ***Range*** | EXTENDED | All the CANIDs are of type extended only (29 bit). | |
| MIXED | The type of CANIDs can be both Standard or Extended. | |
| STANDARD | All the CANIDs are of type standard only  (11bit). | |
| ***Post-Build Variant Value*** | true |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00326 :** |
| ***Name*** | CanObjectId |
| ***Parent Container*** | CanHardwareObject |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Description*** | Holds the handle ID of HRH or HTH. The value of this parameter is unique in a given CAN Driver, and it should start with 0 and continue without any gaps.  The HRH and HTH Ids share a common ID range.  Example: HRH0-0, HRH1-1, HTH0-2, HTH1-3 | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef (Symbolic Name generated for this parameter) | | |
| ***Range*** | 0 .. 65535 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00327 :** | | |
| ***Name*** | CanObjectType | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Specifies if the HardwareObject is used as Transmit or as Receive object | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | RECEIVE | Receive HOH | |
| TRANSMIT | Transmit HOH | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: ECU | | |

i

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00486 :** |  | |
| ***Name*** | CanTriggerTransmitEnable |  | |
| ***Parent Container*** | CanHardwareObject |  | |
| ***Description*** | This parameter defines if or this handle. | f not Can supports the trigger-transmit API for | |
| ***Multiplicity*** | 0..1 |  | |
| ***Type*** | EcucBooleanParamDef |  | |
| ***Default value*** | false |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00322 :** | | |
| ***Name*** | CanControllerRef | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Reference to CAN Controller to which the HOH is associated to. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | Reference to [ CanController ] | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |

|  |  |
| --- | --- |
| ***Scope / Dependency*** | scope: local |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00438 :** | | |
| ***Name*** | CanMainFunctionRWPeriodRef | | |
| ***Parent Container*** | CanHardwareObject | | |
| ***Description*** | Reference to CanMainFunctionPeriod | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | Reference to [ CanMainFunctionRWPeriods ] | | |
| ***Post-Build Variant Multiplicity*** | true | | |
| ***Post-Build Variant Value*** | true | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanHwFilter | 0..\* | This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter. |
| CanTTHardwareObjectTrigge r | 0..\* | CanTTHardwareObjectTrigger is specified in the SWS TTCAN and contains the configuration (parameters) of TTCAN triggers for Hardware Objects, which are additional to the configuration (parameters) of CAN Hardware Objects.  This container is only included and valid if TTCAN is supported by the controller and, enabled (see  CanSupportTTCANRef, ECUC\_Can\_00430), and used. |

##### CanHwFilter

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00468 :** |
| ***Container Name*** | CanHwFilter |
| ***Description*** | This container is only valid for HRHs and contains the configuration (parameters) of one hardware filter. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00469 :** | | |
| ***Name*** | CanHwFilterCode | | |
| ***Parent Container*** | CanHwFilter | | |
| ***Description*** | Specifies (together with the filter mask) the identifiers range that passes the hardware filter. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 4294967295 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |

|  |  |
| --- | --- |
| ***Scope / Dependency*** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00470 :** | | |
| ***Name*** | CanHwFilterMask | | |
| ***Parent Container*** | CanHwFilter | | |
| ***Description*** | Describes a mask for hardware-based filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with the appropriate CanFilterMaskValue. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the respective bit position.  The mask shall be build by filling with leading 0. In case of CanIdType EXTENDED or MIXED a 29 bit mask shall be build. In case of CanIdType STANDARD a 11 bit mask shall be build | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 4294967295 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | true | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | VARIANT-PRE-COMPILE |
| ***Link time*** | -- |  |
| ***Post-build time*** | X | VARIANT-POST-BUILD |
| ***Scope / Dependency*** | dependency: The filter mask settings must be known by the CanIf configuration for optimization of the SW filters. | | |

***No Included Containers***

##### CanConfigSet

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00343 :** |
| ***Container Name*** | CanConfigSet |
| ***Description*** | This container contains the configuration parameters and sub containers of the AUTOSAR Can module. |
| ***Configuration Parameters*** | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanController | 1..\* | This container contains the configuration parameters of the CAN controller(s). |
| CanHardwareObject | 1..\* | This container contains the configuration (parameters) of CAN Hardware Objects. |
| CanIcom | 0..1 | This container contains the parameters for configuring pretended networking |

##### CanMainFunctionRWPeriods

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00437 :** |
| ***Container Name*** | CanMainFunctionRWPeriods |
| ***Description*** | This container contains the parameter for configuring the period for cyclic call to Can\_MainFunction\_Read or Can\_MainFunction\_Write depending on the referring item. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00484 :** | | |
| ***Name*** | CanMainFunctionPeriod | | |
| ***Parent Container*** | CanMainFunctionRWPeriods | | |
| ***Description*** | This parameter describes the period for cyclic call to Can\_MainFunction\_Read or Can\_MainFunction\_Write depending on the referring item. Unit is seconds. Different poll-cycles will be configurable if more than one CanMainFunctionPeriod is configured. In this case multiple Can\_MainFunction\_Read() or Can\_MainFunction\_Write() will be provided  by the CAN Driver module. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucFloatParamDef | | |
| ***Range*** | ]0 .. INF[ |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

***No Included Containers***

##### CanIcom

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00440 :** |
| ***Container Name*** | CanIcom |
| ***Description*** | This container contains the parameters for configuring pretended networking |
| ***Configuration Parameters*** | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanIcomConfig | 1..\* | This container contains the configuration parameters of the ICOM Configuration. |

##### CanIcomConfig

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00459 :** |
| ***Container Name*** | CanIcomConfig |
| ***Description*** | This container contains the configuration parameters of the ICOM Configuration. |
| ***Configuration Parameters*** | |

|  |  |  |
| --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00441 :** | |
| ***Name*** | CanIcomConfigId | |
| ***Parent Container*** | CanIcomConfig | |
| ***Description*** | This parameter identifies the ID of the ICOM configuration. | |
| ***Multiplicity*** | 1 | |
| ***Type*** | EcucIntegerParamDef | |
| ***Range*** | 1 .. 255 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Default value*** | -- |  | |
| ***Post-Build Variant Value*** | false |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00442 :** |  | |
| ***Name*** | CanIcomWakeOnBusOff |  | |
| ***Parent Container*** | CanIcomConfig |  | |
| ***Description*** | This parameter defines that or not. | the MCU shall wake if the bus off is detected | |
| ***Multiplicity*** | 1 |  | |
| ***Type*** | EcucBooleanParamDef |  | |
| ***Default value*** | true |  | |
| ***Post-Build Variant Value*** | false |  | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanIcomWakeupCauses | 1 | This container contains the configuration parameters of the wakeup causes to leave the power saving mode. |

##### CanIcomGeneral

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00444 :** |
| ***Container Name*** | CanIcomGeneral |
| ***Description*** | This container contains the general configuration parameters of the ICOM Configuration. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00445 :** | | |
| ***Name*** | CanIcomLevel | | |
| ***Parent Container*** | CanIcomGeneral | | |
| ***Description*** | Defines the level of Pretended Networking.  This parameter is reserved for future implementations (Pretended Networking level 2). | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | CAN\_ICOM\_LEVEL\_ONE | -- | |
| CAN\_ICOM\_LEVEL\_TWO | -- | |
| ***Default value*** | CAN\_ICOM\_LEVEL\_ONE | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00446 :** | | |
| ***Name*** | CanIcomVariant | | |
| ***Parent Container*** | CanIcomGeneral | | |
| ***Description*** | Defines the variant, which is supported by this CanController | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | CAN\_ICOM\_VARIANT\_HW | -- | |
| CAN\_ICOM\_VARIANT\_NONE | -- | |
| CAN\_ICOM\_VARIANT\_SW | -- | |
| ***Default value*** | CAN\_ICOM\_VARIANT\_NONE | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

***No Included Containers***

##### CanIcomRxMessage

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00447 :** |
| ***Container Name*** | CanIcomRxMessage |
| ***Description*** | This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause.  constraint: For all CanIcomRxMessage instances the Message IDs which  are defined in CanIcomMessageId and in CanIcomRxMessageIdMask shall not overlap. |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00448 :** | | |
| ***Name*** | CanIcomCounterValue | | |
| ***Parent Container*** | CanIcomRxMessage | | |
| ***Description*** | This parameter defines that the MCU shall wake if the message with the ID is received n times on the communication channel. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 1 .. 65536 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration*** | ***Pre-compile time*** | X | All Variants |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class*** | ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00449 :** | | |
| ***Name*** | CanIcomMessageId | | |
| ***Parent Container*** | CanIcomRxMessage | | |
| ***Description*** | This parameter defines the message ID the wakeup causes of this CanIcomRxMessage are configured for. In addition a mask  (CanIcomMessageIdMask) can be defined, in that case it is possible to define a range of rx messages, which can create a wakeup condition. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 536870912 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00465 :** | | |
| ***Name*** | CanIcomMessageIdMask | | |
| ***Parent Container*** | CanIcomRxMessage | | |
| ***Description*** | Describes a mask for filtering of CAN identifiers. The CAN identifiers of incoming messages are masked with this CanIcomMessageIdMask. If the masked identifier matches the masked value of CanIcomMessageId, it can create a wakeup condition for this CanIcomRxMessage. Bits holding a 0 mean don't care, i.e. do not compare the message's identifier in the  respective bit position. The mask shall be build by filling with leading 0. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 .. 536870912 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU  dependency: CanIcomMessageIdMask and CanIcomRxMessageSignalConfig shall not be defined together. | | |

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00450 :** |
| ***Name*** | CanIcomMissingMessageTimerValue |
| ***Parent Container*** | CanIcomRxMessage |
| ***Description*** | This parameter defines that the MCU shall wake if the message with the ID is not received for a specific time in s on the |

|  |  |  |  |
| --- | --- | --- | --- |
|  | communication channel. |  | |
| ***Multiplicity*** | 0..1 |  | |
| ***Type*** | EcucFloatParamDef |  | |
| ***Range*** | [-INF .. INF] |  | |
| ***Default value*** | -- |  | |
| ***Post-Build Variant Multiplicity*** | false |  | |
| ***Post-Build Variant Value*** | false |  | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00451 :** | | |
| ***Name*** | CanIcomPayloadLengthError | | |
| ***Parent Container*** | CanIcomRxMessage | | |
| ***Description*** | This parameter defines that the MCU shall wake if a payload error occurs | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucBooleanParamDef | | |
| ***Default value*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanIcomRxMessageSignalConfi g | 0..\* | This container contains the configuration parameters for the wakeup causes for matching signals.  It has to be configured as often as a signal is defined as wakeup cause. If at least one Signal conditions defined in a CanIcomRxMessageSignalConfig evaluates to true or if no CanIcomRxMessageSignalConfig are defined, the whole wakeup condition is considered to be true. All  instances of this container refer to the same frame/pdu (see CanIcomMessageId). |

##### CanIcomRxMessageSignalConfig

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00452 :** |
| ***Container Name*** | CanIcomRxMessageSignalConfig |
| ***Description*** | This container contains the configuration parameters for the wakeup causes for matching signals.  It has to be configured as often as a signal is defined as wakeup cause. If at least one Signal conditions defined in a CanIcomRxMessageSignalConfig evaluates to true or if no CanIcomRxMessageSignalConfig are defined, the whole wakeup condition is considered to be true. All instances of this container refer to the same |

|  |  |
| --- | --- |
|  | frame/pdu (see CanIcomMessageId). |
| ***Configuration Parameters*** | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00487 :** | | |
| ***Name*** | CanIcomSignalMask | | |
| ***Parent Container*** | CanIcomRxMessageSignalConfig | | |
| ***Description*** | This parameter shall be used to mask a signal in the payload of a CAN message.  The mask is binary AND with the signal payload. The result will be used in combination of the operations defined in CanIcomSignalOperation with the CanIcomSignalValue. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 ..  18446744073709551615 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00462 :** | | |
| ***Name*** | CanIcomSignalOperation | | |
| ***Parent Container*** | CanIcomRxMessageSignalConfig | | |
| ***Description*** | This parameter defines the operation, which shall be used to verify the signal value creates a wakeup condition. | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucEnumerationParamDef | | |
| ***Range*** | AND | The received signal value masked by CanIcomSignalMask has at least one bit  set in common with CanIcomSignalValue (binary AND). | |
| EQUAL | The received signal value masked by CanIcomSignalMask is equal to  CanIcomSignalValue. | |
| GREATER | The received signal value masked by CanIcomSignalMask is strictly greater than CanIcomSignalValue.  Values are interpreted as unsigned integers. | |
| SMALLER | The received signal value masked by CanIcomSignalMask is strictly smaller than CanIcomSignalValue.  Values are interpreted as unsigned integers. | |
| XOR | The received signal value masked by CanIcomSignalMask then XORed to  CanIcomSignalValue is not null. | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00488 :** | | |
| ***Name*** | CanIcomSignalValue | | |
| ***Parent Container*** | CanIcomRxMessageSignalConfig | | |
| ***Description*** | This parameter shall be used to define a signal value which shall be compared (CanIcomSignalOperation) with the masked  CanIcomSignalMask value of the received signal (CanIcomSignalRef). | | |
| ***Multiplicity*** | 1 | | |
| ***Type*** | EcucIntegerParamDef | | |
| ***Range*** | 0 ..  18446744073709551615 |  | |
| ***Default value*** | -- | | |
| ***Post-Build Variant Value*** | false | | |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: local | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00456 :** | | |
| ***Name*** | CanIcomSignalRef | | |
| ***Parent Container*** | CanIcomRxMessageSignalConfig | | |
| ***Description*** | This parameter defines a reference to the signal which shall be checked additional to the message id (CanIcomMessageId).  This reference is used for documentation to define which ComSignal originates this filter setting. All signals being referred by this reference shall point to the same PDU. | | |
| ***Multiplicity*** | 0..1 | | |
| ***Type*** | Reference to [ ComSignal ] | | |
| ***Post-Build Variant Multiplicity*** | false | | |
| ***Post-Build Variant Value*** | false | | |
| ***Multiplicity Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Value Configuration Class*** | ***Pre-compile time*** | X | All Variants |
| ***Link time*** | -- |  |
| ***Post-build time*** | -- |  |
| ***Scope / Dependency*** | scope: ECU  dependency: The signal referenced by CanIcomSignalRef shall be included in a ComIPdu which matches with the current CAN Controller and  the CAN Identifier (CanIcomMessageId) configured for this CanIcomRxMessage. | | |

***No Included Containers***

##### CanIcomWakeupCauses

|  |  |
| --- | --- |
| ***SWS Item*** | **ECUC\_Can\_00443 :** |
| ***Container Name*** | CanIcomWakeupCauses |
| ***Description*** | This container contains the configuration parameters of the wakeup causes to leave the power saving mode. |
| ***Configuration Parameters*** | |

|  |  |  |
| --- | --- | --- |
| ***Included Containers*** | | |
| ***Container Name*** | ***Multiplicity*** | ***Scope / Dependency*** |
| CanIcomRxMessage | 1..\* | This container contains the configuration parameters for the wakeup causes for matching received messages. It has to be configured as often as received messages are defined as wakeup cause.  constraint: For all CanIcomRxMessage instances the Message  IDs which are defined in CanIcomMessageId and in CanIcomRxMessageIdMask shall not overlap. |

### Not applicable requirements

**[SWS\_Can\_00999] ⌈** These requirements are not applicable to this specification. ⌋ (SRS\_BSW\_00170, SRS\_BSW\_00383, SRS\_BSW\_00395, SRS\_BSW\_00397, SRS\_BSW\_00398, SRS\_BSW\_00399, SRS\_BSW\_00400, SRS\_BSW\_00168,

SRS\_BSW\_00423, SRS\_BSW\_00424, SRS\_BSW\_00425, SRS\_BSW\_00426, SRS\_BSW\_00427, SRS\_BSW\_00429, SRS\_BSW\_00433, SRS\_BSW\_00336, SRS\_BSW\_00422, SRS\_BSW\_00417, SRS\_BSW\_00409, SRS\_BSW\_00162, SRS\_BSW\_00415, SRS\_BSW\_00325, SRS\_BSW\_00342, SRS\_BSW\_00453, SRS\_BSW\_00413, SRS\_BSW\_00307, SRS\_BSW\_00447, SRS\_BSW\_00353, SRS\_BSW\_00361, SRS\_BSW\_00439, SRS\_BSW\_00449, SRS\_BSW\_00378, SRS\_BSW\_00359, SRS\_BSW\_00440, SRS\_SPAL\_12163, SRS\_SPAL\_12462, SRS\_SPAL\_12068, SRS\_SPAL\_12064, SRS\_Can\_01125, SRS\_Can\_01126)