
# Open SDV API 仕様

202603α

# 目次

# Chapter 1. Open SDV API仕様の位置付け

## 1.1. オープンSDVとOpen SDV Initiativeの取り組み

### 1.1.1. SDVとオープンSDV

自動車産業が100年に1度の大変革期を迎える中，次世代モビリティの鍵としてSDV（Software Defined Vehicle）の概念が大きな注目を浴びている。SDVとは，無線通信を通じたソフトウェア更新（OTA：Over The Air software update）により，機能を継続的にアップデートすることで，新たな価値を創出することができる自動車のことを言う。

従来の自動車は，製造段階で作り込まれたハードウェアが価値の大部分を占めていた。しかし，SDVにおいては価値の源泉がハードウェアからソフトウェアへと大きく移行する。これにより，車両購入後もソフトウェアの更新を通じて機能が進化し続け，ユーザの嗜好に応じたパーソナライゼーションや，新たなモビリティサービスの継続的な提供が可能となる。

このようなSDVのポテンシャルを，個別のOEM（自動車メーカ）に閉じるのではなく，多様なプレイヤーが参画できるエコシステムの構築を目指すのが，オープンSDVの考え方である。具体的には，サードパーティ（自動車メーカやその委託先以外の組織や個人）が開発したアプリケーションをインストールすることができる自動車を，オープンSDVと呼ぶ。

オープンSDVは，モビリティにおけるイノベーションを大きく加速させるポテンシャルを持っている。従来の自動車やオープンでないSDVにおいて，自動車に新しい機能を追加し，新しいサービスを提供するためには，OEMの同意と協力が不可欠であった。しかし，オープンSDVの世界では，サードパーティが自らアプリケーションを開発・提供することで，独自に新しい価値を生み出すことが可能となる。これにより，自動車は単なる移動手段から，スマートフォンのように，サードパーティの多様なサービスを載せる基盤（プラットフォーム）へとその商品性を大きく変化させることになる。

### 1.1.2. ビークルAPIと標準化

SDVにおいて，アプリケーションから車両の機能を利用し，車両を操作するためには，明確に定義されたインタフェースが必要となる。このインタフェースが，ビークルAPIである。ビークルAPIは，車両の物理的な構造やハードウェアの差異を吸収し，アプリケーション開発者が個々のハードウェアの詳細を意識することなく車両にアクセスする手段を提供する。そして，このビークルAPIを介した要求を受け付け，車両のハードウェアリソースおよびソフトウェアリソースを統合的に管理・制御するソフトウェアプラットフォームが，ビークルOSである。

オープンSDVのエコシステムを形成するためには，OEMの枠を超えたビークルAPIの標準化が不可欠である。サードパーティのアプリケーション開発者の立場からは，開発投資に見合う十分な市場規模を確保するために，開発したアプリケーションが可能な限り多くの車両に共通して適用できることが望まれる。アプリケーションが特定のOEMの車両にしか適用できない状態では，市場規模に限界が生じることから，ビークルAPIが標準化されていることが強く求められる。

また，ビークルAPIの標準化は，SDV化に伴う最大の技術課題であるソフトウェア開発工数の爆発的増加を解決するための手段としても極めて有用である。開発工数を抑制するためには，ハードウェアとソフトウェアのライフサイクルを分離し，ハードウェアの差異に依存せずにソフトウェアを開発できるアーキテクチャへの移行が必要となる。その分離の界面となるビークルAPIを業界全体で標準化することで，OEM間や車種間でのソフトウェア資産の流用・再利用が容易になり，業界全体の開発効率が飛躍的に向上する。

さらに，近年の自動運転技術，特にEnd-to-End（E2E）AI自動運転の急速な進展においても，ビークルAPIの標準化が果たす役割は大きい。高度な自動運転システムを実現するにあたり，OEMが外部の自動運

転システム企業の技術を採用するケースが増加している。この両者間を繋ぐインタフェースを標準化しておくことで，OEMは特定の自動運転システム企業へのロックインを回避し，異なるシステムを柔軟に採用することが可能となる。これは，自動運転分野における健全な競争を促進し，技術革新を持続させるためにも重要である。

現在，国内外においてビークルAPIの標準化に向けた様々な活動が進められている。例えば，国際的なコンソーシアムであるCOVESAは，車両のデータへアクセスするためのVSS（Vehicle Signal Specification）やVISSの仕様を策定している。また，中国のCAAM-SDV（中国自動車工業協会 SDV委員会）や国内のJASPARにおいても，それぞれAPI仕様の策定が進んでいる。しかし，これらの既存の標準化活動は，車両制御の専門知識を持たないサードパーティのアプリケーション開発者が利用するAPIとしては，必ずしも使いやすいものになっているとは言えない状況にある。

### 1.1.3. Open SDV Initiativeの取り組み

Open SDV Initiativeは，近い将来にオープンSDVが実現・普及することを想定して，オープンSDVのためのビークルAPIを策定することを目的に，名古屋大学が産業界に呼びかけて進めている活動である。策定したビークルAPIは，JASPARやCOVESAなどの標準化団体に提案することを考えている。Open SDV Initiativeで策定したビークルAPI仕様を，Open SDV API仕様と呼ぶ。本書は，その最新の仕様を規定するものである。

2024年6月に活動の立ち上げを発表し，2024年10月に本格的に活動を開始した。2026年3月時点で，約60社の企業が活動に参加している。ビークルAPIの策定が急がれる状況であり，他団体との協調のためにも，策定の途上であっても成果を積極的に公表すべきと考えており，活動開始以降，半年ごとにその時点での仕様を公表している。

また，Open SDV Initiativeでは，API仕様の策定に留まらず，それを評価するためのシミュレータや実車を用いたPoC（Proof of Concept）の開発も行っている。さらに，オープンSDVにより実現可能となるサービス検討からビークルAPIへの要求を抽出する活動や，オープンSDVが提供するサービスを体験するためのシミュレーション環境であるMESH（Mobility Experience Simulation Hub）の開発も並行して進めている。

### 1.1.4. 本仕様の策定状況

Open SDV API仕様の現バージョン（バージョン：202603α）は，仕様策定のために立ち上げたAPIコンセプトWG，ボディWG，AD/ADAS WG，HMI WGの4つのWGの現時点の検討成果を取りまとめたものである。本格的な活動開始から1年半で，ビークルAPIのコンセプトは固まってきたが，個々のAPI定義については初期ドラフトの段階である。また，各WGの検討成果間における整合性の確保や，専門的知見を持つメンバの不足により未検討となっている領域の補完など，API群としての網羅性において今後の課題を残している。

今後も，個々のAPIの完成度を上げるとともに，その有効性検証のための実装を通じて，ビークルAPI仕様の継続的な拡充とブラッシュアップを進めていく計画である。

## 1.2. APIの位置付けと策定方針

### 1.2.1. 策定するビークルAPIの位置付け

ビークルOSとは，ハードウェアとしての車両を，アプリケーション開発者およびユーザが，より容易に，より効率的に，かつ安全に利用できるようにすることを目的として，車両のハードウェアリソースおよびソフトウェアリソースを管理し，機能提供を行う一連のソフトウェアをいう。アプリケーションからビークルOSの機能を利用するためのインタフェースをビークルAPIと呼び，本仕様の規定対象とする。

車載コンピュータは車両の一部であるため，車載コンピュータのOSはビークルOSの一部である。ビークルAPIは，車両の持つ様々な機能を提供するインタフェースであり，（コンピュータの）OSが提供するAPIも含む。本仕様では，（コンピュータの）OSのAPIについては，可能な限り既存の仕様を活用することとし，主たる策定対象とはしない。具体的には，AUTOSAR Adaptive Platform（AUTOSAR AP）やAndroid Automotiveが提供する機能のAPIは，原則として策定対象外とする。ただし，既存のOS APIのみではビークルAPIとして不十分である場合には，新たに仕様を策定する。

また，アプリケーションが車両のハードウェアを直接操作したい場合もあるため，車両のハードウェア抽象化層に相当するビークルデバイスインタフェースについても，必要に応じて策定する。

## 1.2.2. ビークルAPIの利用者と適用範囲

策定するビークルAPIは，OEMおよびその委託先のサプライヤに加え，サードパーティの開発者も利用することを想定する。そのため，自動車分野に必ずしも精通していない開発者にとっても利用しやすい仕様とする。

対象とする車両は，まずは一般の乗用車（シェアリング車両やタクシーを含む）を想定し，可能な範囲で，バスやトラックなどのサービスカーへも適用できるように配慮する。

また，対象とするアプリケーションとしては，ボディ制御，自動運転や運転支援，HMIのパーソナライズ，情報提供，エンターテインメント，プローブデータ活用，エネルギー管理など多岐にわたる用途を想定するが，将来的に新たな用途が出現する可能性を考慮し，高い汎用性および拡張性を備えたAPIとする。

なお，クラウド上のAPI（クラウドサーバ上やスマートフォン上のアプリケーションから，ネットワークを介して車両の機能を操作するためのAPI）も重要な課題であると認識しているが，まずは車両上のAPIにフォーカスする。これは，車両の機能を操作するためのAPIにおいて，車両上のAPIとクラウド上のAPIに大きな違いはないと考えられるためである（クラウド上のAPIには，蓄積データにアクセスする機能が必要になる点を除く）。まずは，クラウド上のアプリケーションは，車両上のアプリケーションを介して間接的に車両を操作するという想定を置くものとする。

## 1.2.3. ビークルAPIに対する要件

ビークルAPIの策定にあたっては，以下の要件を満たすことを目標とする。

#### (1) アプリケーションの開発を容易にすること

一度の開発で，多くの車両に適用できるアプリケーションが開発できること。また，自動車に詳しくないサードパーティのアプリケーション開発者にとっても使いやすいAPIであること。

#### (2) OEMの競争を阻害しないものであること

OEMによる車両の差別化を通じた競争を阻害しないために，ハードウェア構成の自由度を狭めず，多様な車両に適用できるAPIであること。

#### (3) 安全性に関わるアプリケーションを扱えること

安全性に関わるアプリケーションの開発をサードパーティには一切許さないものとすると，価値の高いアプリケーションが開発できなくなる可能性がある。そこで，どのアプリケーションにどのAPIの使用を認めるかを，OEMとユーザが決定できる仕組みを用意して，安全性に関わるアプリケーションを扱えるAPIとすること。

### 1.2.4. 要件を満たすためのアプローチ

前節の(1)及び(2)の要件を満たすために，本仕様では以下のアプローチを採用 する。

#### (1) 車両の多様性を吸収するための車両の構成記述

車両のハードウェアの違いは大きいため，APIで完全に隠蔽することは不可能と考えられる。そこで，アプリケーション側で車両の違いを吸収できるよう，車両の構成記述法を定める。

例えば，車両によってウィンドウ（窓）の数が異なることに対しては，ウィンドウのリストをAPIを使って取得し，各ウィンドウに対してAPIを呼び出す方法をとる。このようなケースで，車両の違いに依存せずに使えるAPI（例えば，すべてのウィンドウを操作するAPI）が欲しい場合には，ライブラリ等で用意する。

ただし，車両（主にハードウェア）の機能不足により，アプリケーションが動作できない場合があるのはやむを得ない。

#### (2) サードパーティの開発者が使いやすいサービスベースのAPI

車両の状態を表現するシグナル（車速，ドアの開閉，ウィンドウの開度など）を読み書きするシグナルベースのAPIではなく，「ウィンドウを開ける」といった具体的な操作や機能を呼び出すサービスベースのAPIとする。

サービスベースのAPIは，車両制御に不慣れなサードパーティのアプリケーション開発者にとっては，シグナルベースのAPIより使いやすいものと考えられる。また，アプリケーション間の調停やロックが実現しやすく，独立して開発されたアプリケーションの共存が可能になる。

#### (3) OEMによる車両の差別化を妨げない抽象度の高いAPI

実現手法を制約しない抽象度の高いAPIとし，APIを実現する車両やビークルOSの側で，OEMによる差別化を可能にする。

#### (4) E/Eアーキテクチャへの非依存性の実現

E/Eアーキテクチャの自由度を狭めないために，アプリケーションを実行するECUは限定しない（ただし，Section 1.4で述べるように，ある程度の想定は行う）。下位層に分散プラットフォーム（AUTOSAR APやROSなど）があることを想定すれば，異なるECU上で実現されたサービスをネットワーク経由で呼び出すことは可能であるため，アプリケーションを実行するECUを限定する必要はないためである。

また，現状のE/Eアーキテクチャ上で効率的に実装できるようにするために，サードパーティのアプリケーションを実行するECUと，OEMが作り込む制御アプリケーションを実行するECUが，CANで接続されていることを想定して，APIが効率的に実現できるように仕様の細部を規定する。例えば，ユースケースが成立する範囲で，アプリケーションに伝えるイベントの抜けを許容するような仕様とする。

#### (5) 論理APIの先行策定

本仕様では，まずは論理API（セマンティクスレベルのAPI）を策定する方針とする。これは，物理API（シンタックスレベルのAPI）を規定するためには，プログラミング環境（プログラミング言語やコンピュータのOS）を決めることが必要なためである。また，物理APIが変わったとしても，アプリケーションの作り直しは機械的に行うことができ，その工数は小さいと考えられる。

論理APIでは，APIを介して受け渡しされるデータが取りうる値とその意味を規定する。数値データについては，単位が異なるとアプリケーションの修正が大きくなるため，単位まで規定する。

論理APIの策定に続いて，代表的なプログラミング環境に対する物理API（バインディング）を規定する。これは，異なるプログラミング環境向けの物理API間でのデータ変換等のオーバヘッドが大きくなることを防ぐためと，シミュレータやPoCの開発にあたって必要なためである。

**(6) コアビークルAPIと拡張ビークルAPI**

ビークルAPIを定義するためには，アプリケーションとビークルOSの役割分担を決める必要があるが，この時に，アプリケーション開発者が使いやすいAPIであることという要求を重視すると，車両の違いを隠蔽するという目的からは過剰なAPIとなる場合がある。

そこで，ビークルAPIを，車両の違いを吸収するという目標を達成するためのコアビークルAPIと，アプリケーション開発者が使いやすいAPIとするという目標を達成するための拡張ビークルAPIの2階層で構成するアプローチをとる。

## 1.2.5. リスクのあるAPIの扱い

Section 1.2.3の(3)の要件に対応して，策定するビークルAPIには，使用により危険を生じる可能性があるAPIを排除しないものとする。

APIの使用により危険を生じる可能性がある場合に，車両およびビークルOSで，危険を防止する機能（安全機能）を持っていることが望ましい。そのため，策定するAPIは，可能な範囲で，車両とビークルOSの持つ安全機能により危険を防止できるようなものとする。

一方で，車両とビークルOSが特定の安全機能を備えることを必須とすると，APIを適用できる車両を限定してしまう可能性がある。そこで，Section 1.2.3の(2)の要件を踏まえて，車両とビークルOSが特定の安全機能を持つことを必須とはしない。

あるAPIを使用することでリスクがあるか，そのようなリスクが許容されるかは，以下の要因によって変化する。

- あるAPIにリスクがあるかどうかは，車両とビークルOSが持つ安全機能によって異なる。
- 現時点ではリスクがあるAPIが，技術の発展（車両の安全機能の進化）により，リスクをなくせる可能性がある。
- リスクのあるAPIをサードパーティに使わせて良いかどうかが，国や地域の規制によって異なる場合もある。

そこで，どのアプリケーションにどのAPIの使用を認めるかを，OEMとユーザが決定できる仕組みとして，「リスク制御機能」を導入する。リスク制御機能の概要は以下の通り。

**(1) リスククラスの定義**

オブジェクトの操作によって生じるリスク（人身，財産，環境，プライバシ上のリスク等）をリスククラスとして分類・定義する。

**(2) 車両の構成記述でのリスク記述**

車両やビークルOSの持つ安全機能で対応できていないリスククラスの集合を，車両の構成記述に記述する。

**(3) アプリケーションでのリスクへの対応とOEMによる審査**

アプリケーションで対応したリスククラスの集合を，アプリケーションのリスク制御情報に記述し，OEMがその妥当性を審査して署名を付与する。

**(4) ユーザの承認**

車両の安全機能で対応できておらず，アプリケーションで対応するリスクについて，その対応責任が，OEMからアプリケーション提供者とユーザに移転されていることに関して，ユーザの承認を求める。

**(5) 操作可否の決定**

ビークルOSは，以上の情報を元に，APIによる操作の可否を決定する。

# 1.3. 想定するシステム階層

## 1.3.1. ソフトウェアのモジュール化/階層化の目的

ソフトウェア（特にOS）をモジュール化（階層化を含む）する際の目的の1つは，ハードウェアや周辺環境（外部との通信仕様や交通規則など）の変更があった場合に，1つのモジュールのみの変更で対応できるようにすることである。

例えば，車両のハードウェアに変更があった場合には，ビークルOSのみを変更すれば，アプリケーションはそのまま使用できることが望ましい。また，外部との通信仕様，交通規則（国・地域によって異なる），道路地図の仕様の違いに対しては，1つのモジュールの差し替えのみで対応できるのが望ましい。

## 1.3.2. システム全体の階層構造

Figure 1に，想定するシステム全体の階層構造を示す。

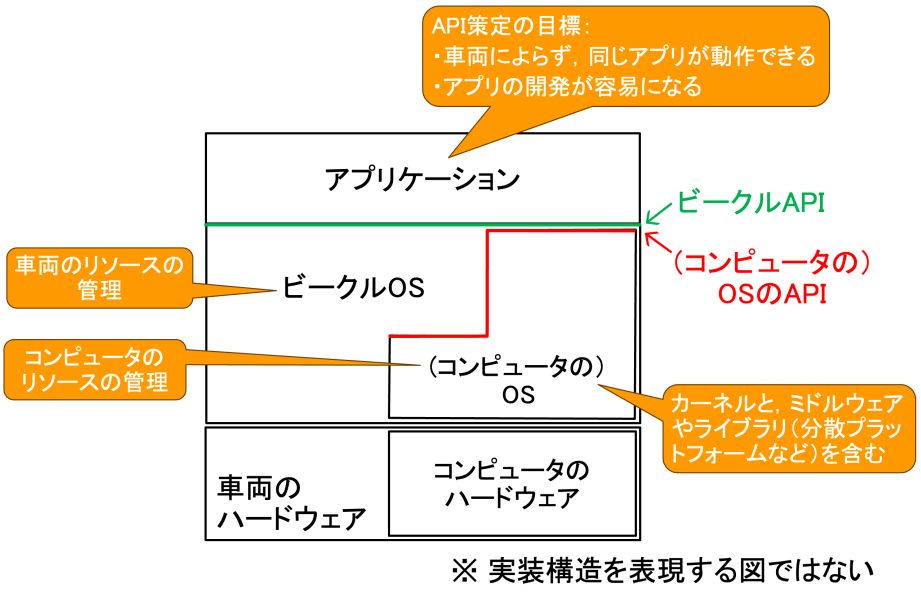



*Figure 1.* 想定するシステム全体の階層構造

Section 1.2.1で述べた通り，車載コンピュータは車両のハードウェアの一部と位置付けられる。したがって，車両のリソースを管理するビークルOSは，コンピュータのリソースを管理する（コンピュータの）OSを含むものと位置付けられる。

ここで，OSという用語は，広義のOSを意味しており，ソフトウェアプラットフォーム（SPF）と同義で用いている。具体的には，図中の（コンピュータの）OSには，OSカーネルに加えて，ミドルウェアやライブラリ，さらにはAUTOSAR　AP（のARA::COM）やROSのような分散プラットフォームを含んでいる。AUTOSAR APのPlatform Foundation FCsに含まれるサービスは，ここに含まれるものと位置付ける。

## 1.3.3. ビークルOSの内部構造

Figure 2に，ビークルOSの内部構造を示す。

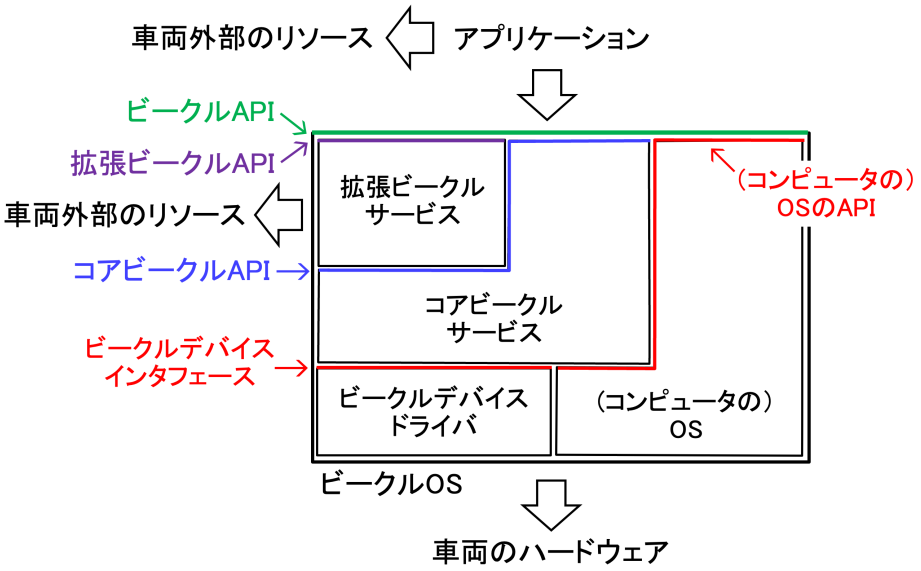



*Figure 2. ビークルOSの内部構造*

### (1) コアビークルサービスとコアビークルAPI

コアビークルサービスは，車両のハードウェアの違いの吸収，車両のハードウェアへのアクセス制御・調停，（可能な範囲での）安全性の確保という3つの役割を担うソフトウェア階層である。コアビークルサービスの機能は，これらの役割を果たせる範囲で最小限に留め，車両のハードウェア以外のリソースを管理する機能は，原則として持たせない方針とする。

コアビークルサービスは，車両のハードウェアの違いを吸収する役割を持つことから，OEMが提供することを想定している。OEM自身やOEMから委託を受けたサプライヤが開発することもあるが，OSメーカが開発してOEMに提供することや，OEMがオープンソースソフトウェアのコアビークルサービスを利用することも考えられる。

アプリケーションや他のビークルサービスが，コアビークルサービスを使用するためのインタフェースを，コアビークルAPIと呼ぶ。コアビークルAPIは，車両によらず同じアプリケーションを動作させるために鍵となるAPIであり，標準化と継続性が強く求められる。

### (2) 拡張ビークルサービスと拡張ビークルAPI

拡張ビークルサービスは，アプリケーションの開発を容易にするためのソフトウェア階層である。例えば道路地図の管理のように，車両のハードウェア以外のリソースを管理する機能は，拡張ビークルサービスとして実現する。また，自動運転ソフトウェアやナビゲーションソフトウェアのように，他のアプリケーションから要求を受け付けるアプリケーションも，拡張ビークルサービスと位置付けることができる。

拡張ビークルサービスは，OEMが提供する場合もあれば，サードパーティが提供する場合もある。その開発は，OEM，サプライヤ，OSメーカ，アプリケーション開発者のいずれが行う可能性もある。

アプリケーションや他のビークルサービスが，拡張ビークルサービスを使用するためのインタフェースを，拡張ビークルAPIと呼ぶ。拡張ビークルAPIは，アプリケーション開発者の視点での使いやすいAPIであることが重要である。車両外部のリソースを管理する機能に対して，1つのAPIに標準化するのは難しい場合もあることから，類似の機能を持つ拡張ビークルAPIが複数あってもよいものとする。

なお，コアビークルサービスと拡張ビークルサービスを総称して，ビークルサービスと呼ぶ。ビークル

APIは，コアビークルAPI，拡張ビークルAPI，（コンピュータの）OSのAPIで構成される。

#### (3) ビークルデバイスドライバ

ビークルデバイスドライバは，車両のハードウェアデバイスを，それぞれ単独で抽象化するためのソフトウェア階層である。

ただし，何を1つのハードウェアデバイスとして扱うか（例えば，ハイブリッド車において，パワートレイン全体を1つのハードウェアデバイスと扱うか，エンジンやモータ等を別々のハードウェアデバイスと扱うか）や，ハードウェアデバイスをどの程度抽象化するか（例えば，カメラからの生画像をそのまま扱うか，画像の認識結果を扱うか）には，一通りには定まらない。

## 1.4. 想定する実装構造

Figure 3に，想定する実装構造を示す。この図は，あくまでも，想定する実装構造の概要を示したものであり，実際の実装構造はより複雑なものとなる。

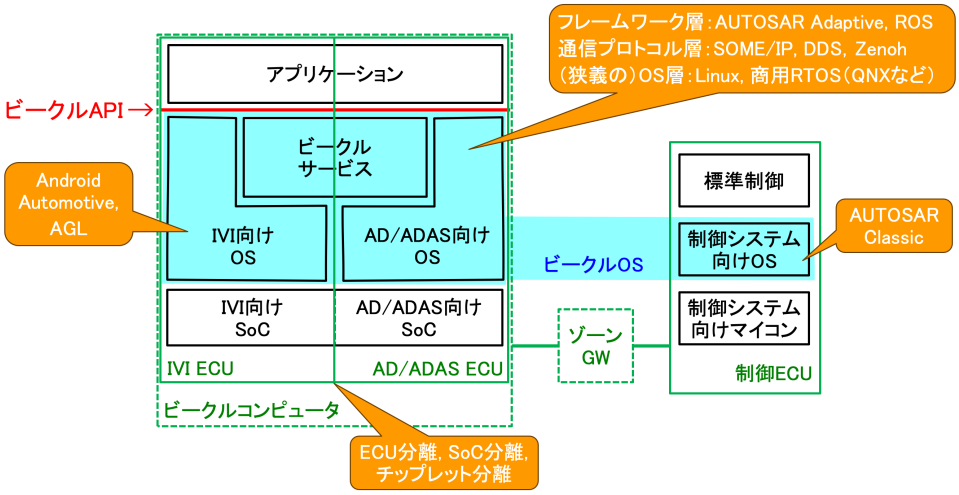


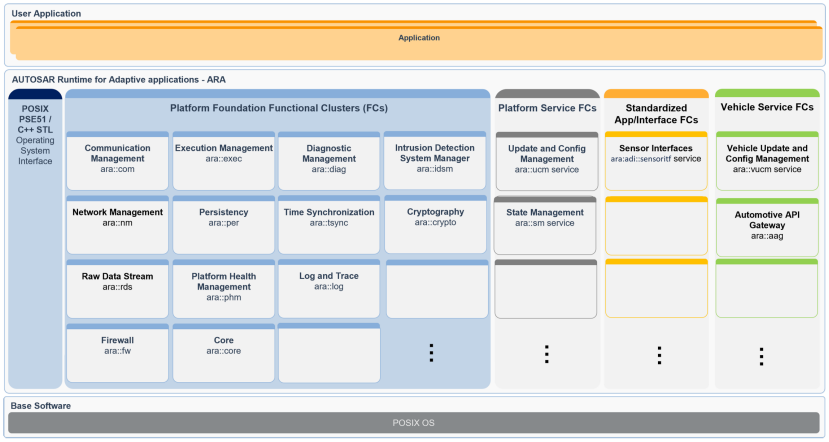

*Figure 3.* 想定する実装構造

機能安全対応が必須のAD/ADAS向けSoC/OSと，機能安全対応が容易でないIVI向けのSoC/OSは，当面は1つにならないと想定している（IVIの一部の機能には機能安全要求があるため，実際には，IVI向けにも複数のOSが使用される）。IVI向けのSoCとAD/ADAS向けのSoCが，チップレットの形で1つのパッケージに入ることは考えられる。

アプリケーションは，IVI ECUまたはAD/ADAS ECUまたはそれらを統合したビークルコンピュータ上で実行することを想定する。さらに別の方法として，アプリケーション実行専用のECUを用意することも想定する。また，アプリケーションによって異なるECUで実行することや，1つのアプリケーションを複数のECUに分散して実行することも想定する。

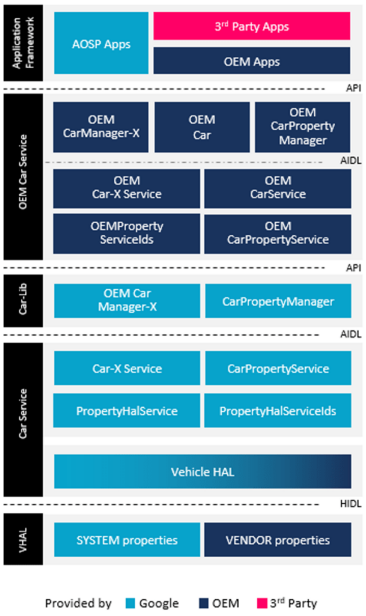
あるECU上で実現されたビークルサービスを，他のECUで実行されているアプリケーションから呼び出すことは可能である。ただし，同じECUで実行されているアプリケーションから呼び出した方が効率は良い。

ビークルサービスの実装構造は，用いる（コンピュータの）OSにより大きく異なる。例えば，AUTOSAR AP上でビークルサービスを実現する場合には，Figure 4のUser ApplicationとARAの間にService層を設け， Service層でビークルサービスを実現することになる。また，Android Automotive上でビークルサービスを実現する場合には，Figure 5の“OEM Car Service”および“Vehicle HAL”で，ビークルサービスを実現することになる。




※ “Explanation of Adaptive Platform Design (R24-11)”より

*Figure 4. AUTOSAR APIの構成*




※ “Android Automotive OS Whitepaper”より

*Figure 5. Android Automotiveの構成*

# Chapter 2. 主要な概念

## 2.1. APIに関する用語とデータ型

### 2.1.1. 用語の定義

#### (1) オブジェクト

ビークルOSが操作対象とするリソース（車両のデバイスや外部リソース）を，ビークルオブジェクト，または単にオブジェクトと呼ぶ。

各オブジェクトに対して，システム内に実体が1つのみ存在する（シングルトン）か，複数存在しうる（マルチインスタンス）かを定める。マルチインスタンスの場合，必要に応じて，オブジェクトの種類をオブジェクトクラス（または単にクラス），個々の実体をオブジェクトインスタンス（または単にインスタンス）と呼んで区別する。

#### (2) 構成と状態

オブジェクトの属性の中で，静的に決まるものを構成（configuration）または構成情報，動的に変化するものを状態（status）と呼ぶ。

両者の中間的な属性もあるため，構成が変わる場合もあるものとする。

#### (3) 状態とイベント

オブジェクトは，一般に，状態を持つ。オブジェクトの状態の離散的な変化のことを，イベントと呼ぶ。

イベントには，状態変化の原因や状態変化前後の状態などの情報が付属している場合がある。

#### (4) インスタンスIDとハンドル

ビークルオブジェクトなどの静的なリソースは，その種類毎（オブジェクトの場合はオブジェクトクラス毎）に，ID（オブジェクトの場合はインスタンスID）と呼ばれる番号で識別する。IDは，1から連続する正の整数値とし，IDを表すデータ型を，IdTypeとする。

動的なリソースは，ハンドルと呼ばれる値で識別する。リソースを生成するサービスコールが，ハンドルの値を決定し，アプリケーションに知らせる。ハンドルのデータ型は，リソース毎に定める。

#### (5) 標準制御

OEMが自動車の出荷時に組み込んでいる制御ロジックを，標準制御と呼ぶ。標準制御は，アプリケーション，ビークルOS，ハードウェアのいずれで実現されている場合もある。

標準制御の動作を止めることができるのが望ましいが，標準制御がハードウェアで実現されている場合など，止められないものがあっても良い。

### 2.1.2. データ型

本仕様では，以下のデータに対して，データ型を定める。

- 車両の構成情報
- サービスコールのパラメータとリターンパラメータ

- イベント

様々なプログラミング環境（プログラミング言語やOS）に適用できるように，データ型は論理レベルで規定する。実装におけるデータ型との対応については，プログラミング環境向けの物理APIで定める。

データ型の名称は，先頭が大文字のキャメルケース（PascalCase）とし，末尾に`Type`を付けるものとする。ただし，基本データ型と文字列型の名称は，この原則の例外とする。

### 2.1.2.1. 論理レベルのデータ型の定義

論理レベルのデータ型の定義では，データ型が取りうる値とその意味を規定する。数値を表すデータ型については，単位まで規定する。

ただし，アプリケーションIDのデータ型（ApplicationIdType）のように，プログラミング環境に依存するデータ型については，物理APIに規定を委ねる場合もある。

論理レベルのデータ型として，以下のデータ型を用意する。

- 基本データ型（ブール型，符号付き/なし整数型，浮動小数点数型）
- 文字列型
- 固定長配列，可変長配列
- 列挙型
- 列挙型の集合型
- 構造体
- バリアント（タグ付き共用体）
- オプショナル型

一方，文字型，ポインタ型，参照型は用意しない。文字型を用意しないのは，本仕様では必要ないためである。

データ型が取りうる値を記述するために，形式的な記述法を導入する。既存のもので目的に合致するものがあれば使いたいが，合致するものが見つからないため，独自に定義する。

### 2.1.2.2. データ型の定義と記述法

ここでは，データ型が取りうる値と，それをプログラミング言語のようなスタイルで定義するための記述法とYAMLによる記述法を規定する。

#### (1) 基本データ型

基本データ型は，ベースとなるデータ型と，オプションの範囲指定で定義/記述する。範囲指定は，数値を表すデータ型（具体的には，ブール型以外のデータ型）に対して，取りうる値の範囲（特殊値を含む）を限定するものである。

ベースとなるデータ型の記述と，取りうる値の範囲は次の通り。

| | | |
|---|---|---|
| ブール型 | Bool | 真（true）と偽（false）の2つの値のみを取るデータ型 |
| 8/16/32/64ビット符号付き整数型 | Int8/Int16/Int32/Int64 | 8/16/32/64ビットの長さの符号付き整数型 |

| | | |
|---|---|---|
| 8/16/32/64ビット<br>符号なし整数型 | Uint8/Uint16/<br>Uint32/Uint64 | 8/16/32/64ビットの長さの符号なし整数型 |
| 32/64ビット<br>浮動小数点数型 | Float32/Float64 | IEEE 754準拠の32/64ビットの浮動小数点数型 |

範囲指定は，数値を表すデータ型（ブール型以外の基本データ型）に対して，取りうる値の範囲（特殊値を含む）を限定するものである。取りうる値の範囲は，以下の区間指定の任意個の和集合の形で指定する。ここで，複数の区間指定に含まれる数値があってはならない。

| | |
|---|---|
| 最小値*a*と最大値*b*で指定 | [*a*,*b*] |
| 排他的下限*a*と最大値*b*で指定 | (*a*,*b*] |
| 最小値*a*と排他的上限*b*で指定 | [*a*,*b*) |
| 排他的下限*a*と排他的上限*b*で指定 | (*a*,*b*) |
| 単一の数値*a*で指定 | *a* |
| 特殊値の定数名*NAME*と数値*value*で指定 | *NAME*(*value*) |

特殊値の定数名は，1つの数値型の中でユニークであれば良い。特殊値の定数名が数値型を超えてユニークでなければならないプログラミング環境向けの物理APIでは，特殊値の定数名の前に，名前の衝突を避けるための文字列を付加する。

範囲指定は，ベースとなるデータ型の名称の後に，()内に，上記の区間指定の記述（[*a*,*b*]，(*a*,*b*]，[*a*,*b*)，(*a*,*b*)，*a*，*NAME*(*value*)のいずれか）を，|で区切って列挙することで記述する。

範囲指定を持つ基本データ型の例は次の通り。

例）8ビットの長さの符号付き整数により，0以上100以下の値と，UNKNOWN（値は-1），NONZERO（値は-2）のいずれかの値をとるデータ型。

```
Int8 ([0,100] | UNKNOWN(-1) | NONZERO(-2))
```

例）IEEE 754準拠の32ビットの浮動小数点数により，0.0以上で無限大（+INF）より小さい値と，NaNのいずれかの値をとるデータ型。

```
Float32 ([0.0,+INF) | NaN)
```

YAMLでは，範囲指定のない基本データ型は，その名称をYAMLのスカラーで記述する。範囲指定を持つ基本データ型は，ベースとなる基本データ型の名称を*BaseType*とすると，次のように記述する。

```
range:
  baseType: BaseType
  intervals:
    - minimum: a
      maximum: b
    - exclusiveMinimum: a
      maximum: b
    - minimum: a
```

```
      exclusiveMaximum: b
    - exclusiveMinimum: a
      exclusiveMaximum: b
    - singleValue: a
    - specialName: NAME
      specialValue: value
```

上記の範囲指定を持つ基本データ型の例は，YAMLでは次のようになる。

```
range:
  baseType: Int8
  intervals:
    - minimum: 0
      maximum: 100
    - specialName: UNKNOWN
      specialValue: -1
    - specialName: NONZERO
      specialValue: -2
```

```
range:
  baseType: Float32
  intervals:
    - minimum: 0.0
      exclusiveMaximum: +INF
    - singleValue: NaN
```

なお，以上で規定したデータ型の記述法は，取りうる値を記述するものであり，実装時のデータ型を決めるものではない。そのため，例えばInt8型のデータを，物理APIの規定で，32ビットの符号付き整数型で実現することとしても良い。

**(2) 文字列型**

Unicodeの文字列を取るデータ型。Stringと記述する。

YAMLでは，YAMLのスカラーでStringと記述する。

**(3) 固定長配列**

指定された要素型のデータの指定された個数の配列。要素型が*ElementType*，要素数が*size*である時，*ElementType*[*size*]と記述する。

YAMLでは，次のように記述する。

```
array:
  elementType: ElementType
  size: size
```

**(4) 可変長配列**

指定された要素型のデータの任意個数の配列。要素数が0の場合もある。可変長配列型のデータからは，配列の要素数を得ることができるものとする。

要素型が*ElementType*である時，*ElementType*[]と記述する。

YAMLでは，次のように記述する。

```
array:
  elementType: ElementType
```

**【補足説明】**

物理APIにおいて，配列を置いた領域へのポインタと要素数，ベクタ（例えば，C++のstd::vector），リストなどで実現することを想定している。

**(5) 列挙型**

あらかじめ定義された列挙子のいずれかを取るデータ型。列挙子は，1つの列挙型の中でユニークであれば良い。

列挙子が$Enumerator_1$，$Enumerator_2$，…，$Enumerator_n$である時，次のように記述する。

```
enum {
    Enumerator₁,
    Enumerator₂,
    …
    Enumeratorₙ
}
```

YAMLでは，次のように記述する。

```
enum:
  - enumerator: Enumerator₁
  - enumerator: Enumerator₂
  …
  - enumerator: Enumeratorₙ
```

列挙子が列挙型を超えてユニークでなければならないプログラミング環境向けの物理APIでは，列挙子の名前の前に，名前の衝突を避けるための文字列を付加する。

**(6) 列挙型の集合型**

指定された列挙型の列挙子の集合を取るデータ型。列挙型が*EnumType*である時，enumSet(*EnumType*)と記述する。

YAMLでは，次のように記述する。

```
enumSet: EnumType
```

**【補足説明】**

物理APIにおいて，列挙型を整数型で実現する場合には，列挙型の集合型は，各列挙子が1つのビットに対応する符号なし整数型（ビットマップ）で実現することを想定している。

列挙型以外のデータ型の集合型も考えられるが，可変長配列で扱えば十分と考え，用意しない。

### (7) 構造体

フィールド名とデータ型が指定されたフィールドを，複数まとめて扱うデータ型。フィールド毎にデータ型が異なっていても良い。

フィールド名が$fieldName_1$，$fieldName_2$，…，$fieldName_n$，それぞれのデータ型が$FieldType_1$，$FieldType_2$，…，$FieldType_n$である時，次のように記述する。

```
struct {
    fieldName₁ : FieldType₁,
    fieldName₂ : FieldType₂,
    …
    fieldNameₙ : FieldTypeₙ
}
```

YAMLでは，次のように記述する。

```
struct:
  - fieldName: fieldName₁
    fieldType: FieldType₁
  - fieldName: fieldName₂
    fieldType: FieldType₂
  …
  - fieldName: fieldNameₙ
    fieldType: FieldTypeₙ
```

### (8) バリアント（タグ付き共用体）

列挙型をパラメータに取り，データの種別を表す列挙子と，その列挙子に対応するデータ（ペイロード）をまとめて扱うデータ型。各列挙子に対するペイロードは，データ型が異なっていても良い。また，ペイロードを持たない列挙子があっても良い。

列挙型が*EnumType*，列挙子が$Enumerator_1$，$Enumerator_2$，…，$Enumerator_n$，それぞれに対応するペイロードのデータ型が$PayloadType_1$，$PayloadType_2$，…，$PayloadType_n$である時，次のように記述する。

```
variant(EnumType) {
    Enumerator₁ : PayloadType₁,
    Enumerator₂ : PayloadType₂,
    …
```

```
    Enumerator_n : PayloadType_n
}
```

ペイロードを持たない列挙子については，列挙子とそれに対応するペイロードのデータ型を記述しない。

ペイロードのデータ型は，構造体とすることを原則とし，バリアントの記述中に構造体の定義を直接記述する場合には，`: struct`は省略できるものとする。具体的には，$Enumerator_i$に対応するペイロードのデータ型が，フィールド名が$fieldName_{i,1}$，$fieldName_{i,2}$，…，$fieldName_{i,mi}$，それぞれのデータ型が$FieldType_{i,1}$，$FieldType_{i,2}$，…，$FieldType_{i,mi}$である構造体である時，次のように記述できる。

```
variant(EnumType) {
    …
    Enumerator_i {
        fieldName_i,1 : FieldType_i,1,
        fieldName_i,2 : FieldType_i,2,
        …
        fieldName_i,mi : FieldType_i,mi
    },
    …
}
```

YAMLでは，上記の2つの記述を，それぞれ次のように記述する。

```
variant:
  parameterType: EnumType
  payloads:
    - enumerator: Enumerator_1
      payloadType: PayloadType_1
    - enumerator: Enumerator_2
      payloadType: PayloadType_2
    …
    - enumerator: Enumerator_n
      payloadType: PayloadType_n
```

```
variant:
  parameterType: EnumType
  payloads:
    …
    - enumerator: Enumerator_i
      payloadType:
        - fieldName: fieldName_i,1
          fieldType: FieldType_i,1
        - fieldName: fieldName_i,2
          fieldType: FieldType_i,2
        …
        - fieldName: fieldName_i,mi
          fieldType: FieldType_i,mi
```

```
    ...
```

ペイロードを持たない列挙子については，それに対応するペイロードのデータ型（payloadType）の記述を省略する。または，列挙子（enumerator）とそれに対応するペイロードのデータ型（payloadType）の両方の記述を省略しても良い。

**【補足説明】**

物理APIにおいて，タグを表す列挙型と（タグのない）共用体，バリアント，関連値付き列挙型などで実現することを想定している。

#### (9) オプショナル型

基礎となるデータ型の値と，値が存在しないことを示す値（null）のいずれかを取るデータ型。オプショナル型のデータからは，値が存在するか否かを判定でき，値が存在する場合にはその値を取得することができる。

基礎となるデータ型が*UnderlyingType*である時，*UnderlyingType*?と記述する。

YAMLでは，次のように記述する。

```
optional: UnderlyingType
```

#### (10) データ型への名前付け

*TypeDescription*で記述されたデータ型に，*TypeName*という名前を付ける時は，次のように記述する。

```
type TypeName = TypeDescription;
```

YAMLでは，次のように記述する。

```
typeDefinition:
  name: TypeName
  type: TypeDescription
```

### 2.1.2.3. YAMLでのデータの記述法

ここでは，車両の構成記述ファイル等で，YAMLによりデータを記述する方法を規定する。データ型をYAMLで記述する方法ではないことに注意すること。

#### (1) 基本データ型，文字列型

YAMLのスカラーの文法に従って記述する。

#### (2) 固定長配列，可変長配列

配列の要素を，YAMLのシーケンスで記述する。

#### (3) 列挙型

列挙子を文字列で記述する。

#### (4) 列挙型の集合型

集合に含まれる列挙子の文字列を，YAMLのシーケンスで記述する。

#### (5) 構造体

フィールド名の文字列をキー，フィールド値の記述を値とするYAMLのマッピングで記述する。

#### (6) バリアント（タグ付き共用体）

列挙子の文字列をキー，ペイロードの記述を値とする1つのエントリを持つYAMLのマッピングで記述する。ペイロードを持たない列挙子の場合は，値をnullとするか，YAMLのマッピングに代えて，列挙子の文字列をYAMLのスカラーとして記述する。

#### (7) オプショナル型

値が存在しない場合は，nullと記述する。ただし，構造体のフィールドである場合に限り，そのフィールド（フィールド名とフィールド値）の記述を省略することで，値が存在しないことを表現しても良い。

## 2.2. 車両の構成記述

車両の構成記述は，車両の構造・装備・機能・性能など，車両の静的な属性を表現するものである。

ある車両のウィンドウの構成情報をYAMLで記述した例を，Figure 6に示す。

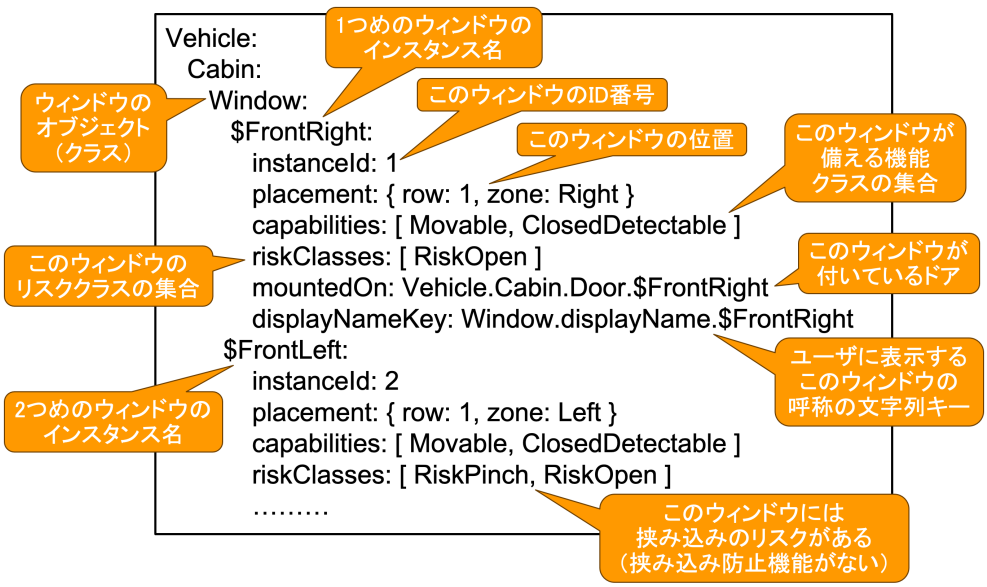



*Figure 6.* 車両構成記述の例

### 2.2.1. 構成記述に対する要求

車両の構成記述法を規定するにあたり，以下の要求を設定した。

**(1) 車両の構造・機能を，トップダウンに整理する**

例えば，洗車モードに遷移させるアプリケーションを作成する場合，水が入る可能性がある開口部をすべて数え挙げられることが必要である。そのためには，キャビンの開口部に何があるか（例えば，ドア，ウィンドウ，ガラス）を整理し，各開口部の性質（例えば，ウィンドウは閉めれば水を通さない）を規定する必要がある。

**(2) 車両依存の拡張を考慮した記述方法とする**

車両の多様性を考えると，車両の構成記述法を網羅的に規定するのは難しく，車両依存で拡張できる必要がある。

## 2.2.2. 車両構成の階層構造

構成記述では，車両の構成を階層構造で表現する。操作の対象となるオブジェクトは，階層構造の中間レベルのノードに対応し，その下のノードに，オブジェクトの構成情報を記述する。

オブジェクトや構成情報の正式な名称は，トップレベルのノード名から開始して，各レベルのノード名を.でつないだ名称とする。

### 2.2.2.1. 階層構造の設計方針

車両構成の階層構造は，COVESA　VSS[1]をベースに，DVE+Uモデル[2]も参考に定める。ただし，以下の設計方針を設定したため，VSSとはかなり違ったものとなっている。

- 多様な車両を記述可能なものとする
- 階層構造のレベルをあまり深くならないようにする
- トップレベルの直下に構成情報を置かない
- マルチインスタンスの扱いを明確にする
- アプリケーションのデータモデルは含めない

[1] https://covesa.github.io/vehicle_signal_specification/

[2] M. Gondo: "(The) Four Principles of SDV", 15th AUTOSAR OPEN CONFERENCE, 2025年6月.

### 2.2.2.2. トップレベルのノード

車両構成の階層構造のトップレベルのノードは，VSSと同様に，Vehicleのみとする。

**【仕様決定の理由】**

DVE+Uモデルをベースに考えると，トップレベルのノードをVehicle，Driver，Environmentの3つとすることも考えられるが，Driverは車両の運転者であること，Environmentも車両の周辺環境であることを考えて，COVESA VSSと同様に，トップレベルのノードはVehicleのみとした。

### 2.2.2.3. 第2レベルのノード

車両構成の階層構造の第2レベルのノードは，現時点では以下のようにする。

| | |
|---|---|
| VersionOSDVI | Open SDV API仕様のバージョン情報 |
| VehicleIdentification | 車両の識別情報 |

| | |
|---|---|
| Specification | 車両諸元 |
| Cabin | 乗員が乗る空間 |
| CargoSpace | 荷物を乗せる空間 |
| Body | 主に車両の外側にある装備 |
| Infotainment | インフォテイメント |
| HMI | HMI |
| Motion | 車両の運動制御 |
| CurrentLocation | 車両の現在位置 |
| Driver | 運転者 |
| Occupant | 運転者以外の乗員 |
| SurroundModel | 車両の周辺モデル |
| Atmosphere | 車両の環境情報 |

**【補足説明】**

運転者は，遠隔運転を考慮に入れて，車内にいるとは限らないものとする。

**【COVESA VSSとの関係と仕様決定の理由】**

VehicleIdentification，Cabin，Body，CurrentLocation，Driver，Occupantは，VSSの第2レベルのノードと同様である。InfotainmentはVSSでは第3レベル（Cabinの下），HMIはVSSでは第4レベル（Cabin.Infotainmentの下）に置かれているが，階層構造のレベルをあまり深くならないようにするという方針から，第2レベルに移動した。Atmosphereは，VSSではExteriorという名称になっている。

VersionOSDVIは，VersionVSSに対応するものとして設けた。

その他は，VSSにはない第2レベルのノードであるが，以下の理由で新設した。

Motionは，VSSでVehicle直下に置かれている車両の移動に関する属性を置くために，Specificationは，Vehicle直下に置かれているその他の車両諸元を置くために設けた。

CargoSpaceは，物流用の車両を扱うために新設した。

SurroundModelは，車両の周辺モデルの表現方法をISO　23150をベースに定めることとしたことから，ISO 23150の用語を導入した。

**【未決定事項】**

VSSにあるその他の第2レベルのノード（Powertrain，Chassis，ADAS，Service，Connectivity，Diagnostics，MotionManagement，ControlUnitなど）については十分に検討できておらず，さらに検討が必要である。

第3レベル以下のノードについては，今後の課題である。

#### 2.2.2.4. インスタンスを表すノードとインスタンスID

マルチインスタンスのオブジェクトに対しては，クラスを表すノードの下に，インスタンスを表すノードを置く。

インスタンスを表すノード名は，インスタンス名の前に$を付けた文字列とする。インスタンス名は，車両毎に自由に決めることができる。

インスタンスの構成情報には，サービスコールなどでインスタンスを数値で識別する場合に用いるインスタンスID（instanceId）を含める。

#### 2.2.2.5. オブジェクトに付属するオブジェクト

オブジェクトに付属するオブジェクトは，前者のオブジェクトのインスタンスの下に置くのではなく，独立したオブジェクトと扱い，付属関係は，構成情報により表現する。

例えば，ウィンドウやガラスに付属しているシェード（日除け）は，各種のシェードをすべてVehicle.Cabin.Shadeのインスタンスとし，構成情報を使ってウィンドウやガラスと紐付ける。これにより，どのシェードも同じサービスコールで操作することができる。

**【COVESA VSSとの関係】**

VSSでは，リアガラスのシェードはVehicle.Cabin.RearShade，ウィンドウのシェードはVehicle.Cabin.Door.Row1.DriverSide.Shadeなど，サンルーフのシェードはVehicle.Cabin.Sunroof.Shadeとなっている。

#### 2.2.2.6. OEM固有のオブジェクト/属性

OEMは，固有のオブジェクトや属性（構成情報や状態）を追加することができる。

OEM固有のオブジェクト/属性の名称は，先頭に，OEM名と$を付けた文字列とする。

### 2.2.3. 車両の構成情報の取得方法

車両の構成情報を取得する方法には，構成記述ファイルを参照する方法と，構成情報を取得するためのサービスコールによる方法がある。

#### 2.2.3.1. 構成記述ファイル

車両の構成記述を含むファイルを，車両の構成記述ファイルと呼ぶ。

構成記述ファイルには，車両の構成情報をYAML形式で記述する。具体的には，車両構成の階層構造の各レベルのノードを，ノード名をキー，ノードの構成情報を値とするYAMLのマッピングで記述する（Figure 6）。

**【補足説明】**

構成記述ファイルのユースケースは以下の通り。

- ビークルOSは，構成記述ファイルを読み込む。ただし，ビークルOSが単一の車両のみに対応するのであれば，ビークルOS自身が車両の構成記述を参照することは必須ではない。
- アプリケーション開発者は，アプリケーションを対応させたい車両群の構成記述ファイルを参考にすることができる。

- シミュレータや各種ツールが，構成記述ファイルを利用することも考えられる。
- アプリケーションは，車両の構成情報の取得するために，構成記述ファイルを読んでも良い。ただし，一般には，getConfig/getConfigAllサービスコールにより取得した方が効率的である。

#### 2.2.3.2. 構成情報を取得するためのサービスコール

シングルトンオブジェクトは，getConfigサービスコールにより，その全構成情報を参照できる。マルチインスタンスオブジェクトは，getConfigAllサービスコールにより，そのすべてのインスタンスの全構成情報を参照できる。

マルチインスタンスオブジェクトに対するgetConfigAllは，インスタンスの全構成情報を含む構造体の可変長配列を返す（配列の要素数は，インスタンスの数となる）。getConfigAllでインスタンス名を取得するために，インスタンスの構成情報の構造体に，インスタンス名を格納するためのString型のフィールド（instanceName）を追加する。ただし，連想配列が扱えるプログラミング環境では，可変長配列を連想配列で実現することで，インスタンス名を連想配列のキーの形で表すことができる。

関連するオブジェクトを表すためのRelatedObjectTypeのデータ型は，プログラミング環境によって異なる場合がある。getConfig/getConfigAllは，構成情報中のRelatedObjectTypeのデータを，そのプログラミング環境向けの物理APIで規定された形に変換して返す。

### 2.2.4. 車両の構成情報の多言語対応

オブジェクトの構成情報の中で，ユーザに表示することを目的としたオブジェクトの呼称などは，言語によって異なる文字列となる。そこで，以下の方法で多言語対応する。

構成記述ファイル中には，表示用の文字列ではなく，多言語対応する文字列のキーを記述する。言語毎の実際の文字列は，ロケール毎に用意する別のファイル（文字列定義ファイル）に記述する。文字列定義ファイルの例を下に示す。

```
Window:
  displayName:
    $FrontRight: "運転席の窓"
    $FrontLeft: "助手席の窓"
    $RearRight: "後席右側の窓"
    $RearLeft: "後席左側の窓"
```

getConfig/getConfigAllは，構成記述ファイル中の文字列のキーを，呼び出し環境のロケールに対応する文字列に置き換えて返す。

### 2.2.5. 構成情報の変化

構成情報はオブジェクトの静的な属性であるが，変化することも想定している。ただし，構成情報が頻繁に変化することは想定しておらず，構成情報が変化した場合には，アプリケーションの再起動が必要になっても良いものとする。

オブジェクトの構成情報が変化した場合には，そのオブジェクトに対して，構成情報が変化したことを伝えるイベント（ConfigChanged）を生成する。トップレベルや中間レベルのノードに対応するオブジェクトに対しては，そのノードの下のオブジェクトの構成情報が変化した時にも，ConfigChangedを生成する。

マルチインスタンスオブジェクトにおいて，インスタンスが増えたことを知るためには，イベント通知対

象のインスタンスIDを指定せずにイベント通知を要求する必要がある。

## 2.3. APIの要素と規定内容

オブジェクト毎のAPIは，Chapter 4とChapter 5で規定する。ただし，複数のオブジェクトに共通の事項については，共通規定としてChapter 3で規定する。

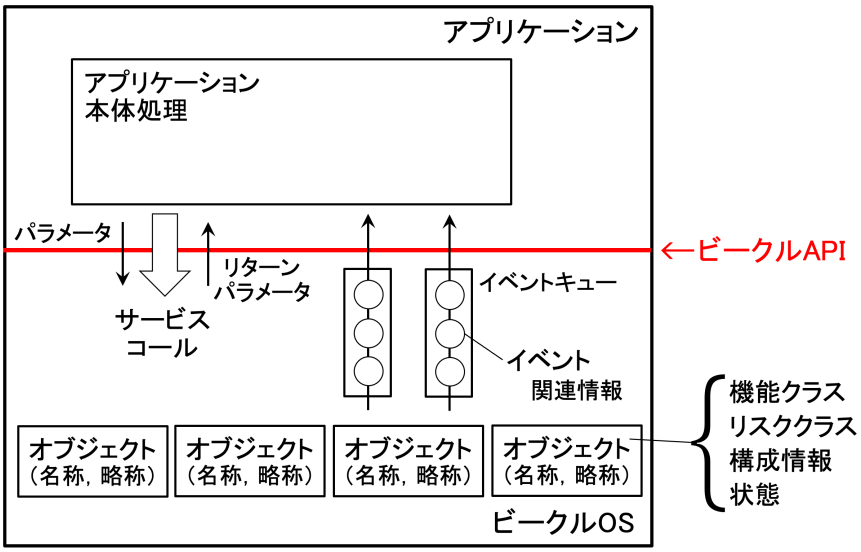



*Figure 7. APIの要素*

オブジェクト毎のAPI規定では，Figure 7に示すAPIの各要素に関して，以下の内容を規定する。

- 名称と略称
- 位置付けと機能
- 機能クラス
- リスククラス
- 構成情報
- 状態
- サービスコール一覧
- イベント
- データ型の定義
- サービスコールの詳細規定

### 2.3.1. 名称と略称

オブジェクトクラスの正式な名称は，車両の構成記述における階層構造を示す名称（フルパス名）とする。例えば，ウィンドウのクラスの正式な名称は，Vehicle.Cabin.Windowである。

各オブジェクトクラスに対して，ユニークな略称を与える。例えば，ウィンドウのクラスの略称は，Windowである。略称は，データ型名の一部やObjectKindTypeの列挙子などに使用する。

### 2.3.2. 位置付けと機能

オブジェクトの位置付けとして，車両内での位置付けや他のオブジェクトとの関係，オブジェクトの性質を説明する。また，オブジェクトが，シングルトンかマルチインスタンスかを規定する。

オブジェクトの機能として，オブジェクトが持つ機能と動作について説明する。オブジェクトがMovableObjectの機能を持つ場合には，そのことを明記する。また，オブジェクトに対する操作の調停方法について説明する。標準的な調停機能が適用される場合には，そのことを明記する。オブジェクトがLockableObjectである場合には，そのことを明記する。

### 2.3.3. 機能クラス

オブジェクトの備える機能を分類したものを機能クラスと呼ぶ。車両の構成記述で，各オブジェクト（の各インスタンス）に対して，備えている機能クラスの集合を記述する。

オブジェクトが特定の機能安全要求を満たすことは，リスククラスではなく，機能クラスで表現する。例えば，車両のバック時に後方を見るためのリアビューアプリケーションには，画像がフリーズしてはならないという機能安全要求がある。これを満たすためには，フリーズしないビュー（ディスプレイの全体またはその一部分）を用いる必要がある。これを実現するために，ビューがフリーズする可能性がない（または，フリーズを防止する機能を持っている）ことを機能クラスで記述し，リアビューアプリケーションはその機能クラスを持ったビューを選択して使用する設計とする。

機能クラスのデータ型は，オブジェクト毎に，*xxx*CapabilityType（*xxx*はオブジェクト略称）の名称で，列挙型として定義する。

オブジェクト毎のAPIの規定では，オブジェクトが持つ各機能クラスの名称と，どのような機能であるかについて規定する。

**【補足説明】**

上記の例において，リアビューアプリケーションは，車両が走行中にセンターディスプレイを使用するため，RiskDistractionCid（センターディスプレイに描画することで生じるドライバディストラクションのリスク）に対応できていることが，OEMによる審査で認められる必要がある。リアビューアプリケーションが，フリーズする可能性がないビューを使用する設計になっていることは，このリスク対応の一部として審査対象となる。

### 2.3.4. リスククラス

オブジェクトを操作することで生じるリスクを分類したものをリスククラスと呼ぶ。リスククラスの詳細については，Section 2.5.2.1を参照のこと。

リスククラスのデータ型は，オブジェクト毎に，*xxx*RiskType（*xxx*はオブジェクト略称）の名称で，列挙型として定義する。

操作することによって何らかのリスクが生じる可能性があるオブジェクトのAPIの規定では，オブジェクトが持つ各リスククラスの名称と，どのようなリスクであるかについて規定する。また，危険を生じる可能性がある操作の種類（サービスコールとそのパラメータ）と車両状態について，サービスコールの機能の項で規定する。

### 2.3.5. 構成情報

オブジェクトの機能・性能など，オブジェクトの属性の中で静的に決まるものを，構成情報と呼ぶ。マルチインスタンスオブジェクトでは，インスタンス毎に構成情報を持つ。

オブジェクト毎のAPIの規定では，オブジェクト毎に，構成情報の名称，データ型，意味などを規定する。

## 2.3.6. 状態

オブジェクトの属性の中で動的に変化するものを，状態と呼ぶ。マルチインスタンスオブジェクトでは，インスタンス毎に状態を持つ。

オブジェクト毎のAPIの規定では，オブジェクト毎に，状態の名称，データ型，意味などを規定する。

## 2.3.7. サービスコール

アプリケーションが，ビークルOSのサービスを呼び出すインタフェースを，サービスコールと呼ぶ。

オブジェクト毎のAPIの規定では，オブジェクト毎に，サービスコールの名称と機能，パラメータとリターンパラメータの名称，データ型，意味などを規定する。

### 2.3.7.1. パラメータとリターンパラメータ

サービスコールは，パラメータとリターンパラメータを持つ。マルチインスタンスオブジェクトに対するサービスコールにおいては，原則として，第1パラメータを，インスタンスを識別するためのインスタンスIDとする。

サービスコールは，正常に実行された場合にはE_OK，エラーになった場合にはエラーの種類を示すエラーコードを返す。これを，サービスコールの戻り値と呼び，リターンパラメータの1つとする。戻り値のデータ型（ReturnType）は，すべてのサービスコールで共通とする。そのため，特定のサービスコールに特化したエラーコードは設けず，自分がロックしていないオブジェクトに対してunlockを呼んだ場合のエラーをE_OBJECT_STATUSとするなど，一般的なエラーコードを用いる。

サービスコールが複数のエラーを検出した場合には，その内のどのエラーコードを返しても良い。物理APIで，エラーコードを追加する場合がある。

別途の規定がない限り，サービスコールがエラーになった場合には，操作対象のオブジェクトの状態は変化しないものとする。

**【補足説明】**

オブジェクト指向言語など向けの物理APIでは，インスタンスIDを表す第1パラメータは，メソッドのパラメータとはせず，メソッドを受け取るオブジェクトで表しても良い。

### 2.3.7.2. 並行して呼び出されたサービスコール

並行して呼び出されたサービスコール（あるサービスコールからリターンする前に呼び出されたサービスコール）は，別途の規定がない限り，サービスコールが呼び出されてからリターンするまでの間のある瞬間にアトミックに実行された場合と同等の振る舞いをする。

どの瞬間に実行されるかは規定されないため，先に呼び出されたサービスコールが先に実行されるとは限らない。言い換えると，並行して呼び出されたサービスコール間の実行順序は任意となる。

### 2.3.7.3. サービスコール実装上の前提

ビークルサービスは以下の情報を取得できるものとする。

- サービスコールを呼び出したアプリケーションID。制御の調停などに用いる。
- サービスコールを呼び出したアプリケーションのリスク制御情報。リスク制御に用いる。リスク制御情報については，Section 2.5.2.3を参照のこと。

- サービスコールを呼び出したアプリケーションのロケール（言語および地域設定）。

## 2.3.8. イベント

ビークルOSが管理するオブジェクトの状態変化を，アプリケーションに伝えるためのデータを，イベントと呼ぶ。イベントは，イベント種別を持つことに加えて，イベントの発生原因などを伝えるために，イベント種別毎の関連情報を持つことができる。

イベントのデータ型は，イベントが発生したインスタンスのID（マルチインスタンスオブジェクトの場合のみ）と，イベント情報をフィールドに持つ構造体とする。イベント情報のデータ型は，イベント種別を表す列挙型をパラメータとし，イベント種別毎の関連情報を表す構造体をペイロードとするバリアントとする。

オブジェクト毎のAPIの規定では，オブジェクト毎に，イベント種別の名称と意味，イベント種別毎の関連情報の名称，データ型，意味，イベントのデータ型などを規定する。

### 2.3.8.1. イベントの発生条件と発生順序

イベントは，別途の規定がない限り，オブジェクトの状態が変化しない場合には発生するとは限らない。

サービスコールの実行によって生成されるイベントは，サービスコールの実行順序でイベントキューに入れられる。1つのサービスコールで複数のイベントが生成される時には，それらの間に因果関係がある場合は原因となるイベントが先に，因果関係がない場合には任意の順序で，イベントキューに入れられる。

### 2.3.8.2. イベントキュー

ビークルOSからのイベントはイベントキューに格納され，アプリケーションは，イベントキューからイベントを受け取る。イベントキューは，複数のイベントを入れることができ，FIFO順でイベントを管理する。

アプリケーションは，複数のイベントキューを持つことができる。イベントキューは通知ハンドルで識別する。

イベントの優先度管理をしたい場合には，複数のイベントキューを用いて，アプリケーション側で実現する。

### 2.3.8.3. イベント操作のサービスコール

イベントを操作するためのサービスコールとして，オブジェクト毎に，イベント通知を要求するサービスコール（notify），イベント通知を停止するサービスコール（unnotify），イベントを取り出すサービスコール（getEvent）を用意する。

notifyサービスコールが正常に実行されると，イベントキューた生成され，イベントキューを識別するための通知ハンドルが返される。受け取るイベントの種類を限定する場合には，notifyサービスコールのパラメータで指定する。受け取るイベントの種類を指定する方法が複数ある場合には，オブジェクトに対して複数種類のnotifyサービスコールを用意する。

unnotifyサービスコールは，通知ハンドルを指定してイベント通知を停止する。正常に実行されると，通知ハンドルに対応するイベントキューは削除される。

getEventサービスコールは，通知ハンドルで指定したイベントキューからイベントを取り出す。イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，getEventサービルコールを用意しなくても良い。

#### 2.3.8.4. イベントキューのオーバフロー

イベントキューがオーバフローした場合，イベントキュー中の古いイベントから順に削除する。

イベントを削除した時には，イベントキューがオーバフローしたことを示すイベント（EventOverflow）を生成し，イベントキューの先頭に入れる（イベントキューはFIFO順でイベントを管理するという規定の例外）。

EventOverflowは，関連情報として，失われたイベント数を持つ。イベントキューがオーバフローした時にイベントキューの先頭にEventOverflowがある場合には，失われたイベント数を増加させ，EventOverflowをもう1つ生成することはしない。

#### 2.3.8.5. イベントキューの一本化

プログラミング環境によっては，物理APIの規定により，すべてのイベントを，1つのイベントキューで管理することとしても良い。

この場合，イベントのデータ型（構造体）に，オブジェクトの種別（クラス）と通知ハンドルのフィールドを追加する。

また，notify/unnotifyサービスコールは，イベントキューの生成/削除を行わない。getEventサービスコールは，システム全体で1種類のみとし，パラメータに通知ハンドルを含めない。

## 2.4. APIの設計方針

### 2.4.1. サービスコールの設計方針

サービスコールは，機械動作にかかる時間と比べて，短い時間で処理が完了するような設計とする。例えば，ウィンドウを操作するサービスコールは，開閉動作が完了するまで待つのではなく，目標位置を指定して，開閉動作を開始するのみとする。言い換えると，機械動作に対しては，サービスコールは非同期（完了を待たない）とする。

一方，サービスコールの処理に複数のECUが関係する場合に，ネットワーク経由で他のECUと通信する間は，サービスコールの処理が継続しても良い。言い換えると，コンピュータおよびネットワークの動作に対して，サービスコールは同期（動作の完了まで処理が続く）とする。

ただし，サービスコールからリターンするタイミングは，プログラミング環境依存で，サービスコールの処理完了後にリターンする場合（ブロッキング）と，サービスコールの処理完了を待たずにリターンし，リターンパラメータは処理完了後に受け取るようにする場合（ノンブロッキング）がある。

サービスコールの呼び出し頻度/回数に関して，サービスコールを短い周期（具体的には，10ミリ秒未満）で呼び出す必要があるような設計としない。また，サービスコールの粒度を大きくし，呼び出し回数が少なくなるような設計とする。これは，サービスコールが，ネットワークを越えて呼び出されることが想定されるためである。

サービスコールを特定の順序で呼び出さなければならないような設計にはしない。具体的には，POSIXにおけるファイル/デバイスのopen/closeのようなサービスコールは設けない。

### 2.4.2. OSに対する想定

ビークルサービスを実現するベースとなる（コンピュータの）OSは，OSが一般的に持つ機能を備えており，アプリケーションはそれらの機能を利用できるものとする。特に，以下の機能を持つものとする。

- ECUを越えたサービス呼び出し機能とPub/Sub通信機能

- 他のECU上で新しいプロセスを立ち上げる機能

# 2.5. 制御の調停とリスク制御

## 2.5.1. 制御の調停機能

ビークルOSは，複数のアプリケーションが同一のオブジェクトを操作（または，出力装置に出力）した場合に，それらを調停する機能を持つ。調停機能に必要となるロックと操作優先度の概念を，APIに導入する。

コアビークルサービスには汎用的な調停機能のみを持たせ，アプリケーション特有の調停機能が必要な場合には，拡張ビークルサービスで実現する方針とする。

ここでは，多くのオブジェクトに適用できる標準的な調停機能を規定する。標準的な調停機能では不十分なオブジェクト（例えば，音声出力）に対しては，オブジェクト固有の調停機能を持たせる。

### 2.5.1.1. 後勝ちの原則

後に実行されたサービスコールによる操作が，先に実行されたサービスコールによる操作をオーバライドすることを原則とする。

例えば，ウィンドウを閉めるサービスコールが実行され，ウィンドウが閉じる動作の途中で，ウィンドウを開くサービスコールが実行されると，最初のサービスコールによる操作はキャンセルされ，ウィンドウは開く動作を始める。なお，後述の操作優先度は，この振る舞いには関係しない。

**【補足説明】**

後勝ちの原則は，サービスコールが呼び出されたタイミングではなく，実行されたタイミングに適用される。並行して呼び出されたサービスコール間の実行順序は任意であることから，並行して呼び出されたサービスコールの中のどれが勝つかは規定されない。

サービスコールを，短い時間で処理が完了するように設計するとしたことから，この原則が自然と考えられる。

### 2.5.1.2. ロック機能

あるアプリケーションが制御しているオブジェクトを，他のアプリケーションのインスタンスが操作するのを禁止するために，必要なオブジェクトに対して，ロック機能を持たせる。ロック機能は，ハードウェアを用いて実現される場合（例えば，ドアロック）と，ソフトウェアのみで実現される場合がある。

ロック機能を持つオブジェクトは，ロックを取得するサービスコールとロックを解除するサービスコールを持つ。

**【補足説明】**

ロック機能のユースケースの例として，洗車モードに遷移させるアプリケーションが，他のアプリケーションやウィンドウ開閉スイッチでウィンドウを開けられないようにするために使用することが挙げられる。

ロックを所有するのは，アプリケーションのインスタンスである。したがって，アプリケーションがマルチスレッド実装されている場合，アプリケーションのインスタンス内のスレッド間では，ロック状態が共有される。あるスレッドがロックを取得すれば，同じインスタンス内の他のスレッドもロック権限を持つ。逆に，同じアプリケーションであっても，別インスタンスであればロックは共有されない。

#### 2.5.1.3. 操作優先度

あるアプリケーションがオブジェクトをロック中に，他のアプリケーションがそれをオーバーライドすることを可能にするために，操作優先度を導入する。操作優先度は，標準的には，ロックに対してのみ意味を持つ。ただし，オブジェクトによっては，操作優先度に追加の意味を持たせる場合もある。

ロックを取得するサービスコールは，パラメータに操作優先度を持つ。アプリケーションAが操作優先度Pを指定してロックを取得した場合，そのオブジェクトは，アプリケーションAによって操作優先度Pでロックされていると言う。

また，ロック機能を持つオブジェクトを操作するサービスコール（オブジェクトの状態を変化させないものを除く）も，パラメータに操作優先度を持つ。

アプリケーションAによって操作優先度Pでロックされているオブジェクトに対して，別のアプリケーションBが操作優先度Qを指定してロック取得/操作するサービスコールを呼び出した場合の振る舞いは次の通り。

- QがPと同じかより低い操作優先度である場合，ロック取得/操作サービスコールは，E_OBJECT_LOCKEDエラーとなる。
- QがPより高い操作優先度である場合，アプリケーションAによるロックは強制解除され，ロック取得/操作サービスコールは成功する（ロック取得サービスコールの場合，アプリケーションBがロックを奪うことになる）。

なお，アプリケーションA自身がPより高い操作優先度Qを指定してロック取得サービスコールを呼び出した場合，ロックの強制解除は行われず，ロックを取得した操作優先度がQに更新される。

ロック取得/操作サービスコールに対して，そのアプリケーションが使用できる最高値よりも高い操作優先度を指定した場合，そのサービスコールはE_DENIED_PRIORITYエラーとなる。

なお，ロック取得/操作サービスコールの操作優先度パラメータはオプションである。操作優先度を指定しない場合，そのアプリケーションが使用できる操作優先度の最高値を指定したものと扱う。

#### 2.5.1.4. 他の調停方法

先勝ちの調停は，ロックを用いて実現できる。例えば，ウィンドウを閉めるサービスコールを呼び出してからウィンドウが完全に閉まるまで，他のアプリケーションによる操作を許したくない場合には，ウィンドウをロックした後にウィンドウを閉めるサービスコールを呼び出し，ウィンドウが完全に閉まったらロックを解除すれば良い。

### 2.5.2. リスク制御機能

ビークルOSは，何らかのリスクがあるAPIの使用をアプリケーションに許すかどうかを制御するために，リスク制御の機能を持つ。リスク制御機能は，人身に対するリスクだけでなく，財産・物品・環境に対するリスクやプライバシ上のリスクも対象とする。

ここでは，リスク制御機能の仕様を規定する。

#### 2.5.2.1. リスククラスの定義

操作することによって何らかのリスクが生じる可能性があるオブジェクトに対しては，本仕様で，そのオブジェクトに対するリスククラスを定義する。リスククラスは，「リスクを分類したもの」の意味である。

一般には，各オブジェクトに対して複数のリスククラスがある。例えば，ウィンドウに対しては，次のリスククラスがある。

| | |
|---|---|
| RiskPinch | ウィンドウを閉めることに伴う挟み込みのリスク |
| RiskOpen | ウィンドウを開けることに伴う諸々のリスク |

あるリスククラスが，実際に危険を生じる可能性があるかは，操作の種類（サービスコールとそのパラメータ）や車両状態に依存する。例えば，ウィンドウに対するRiskPinchは，ウィンドウの状態参照やウィンドウを開く操作では危険を生じない。また，ビュー（ディスプレイの全体またはその一部分）に対するRiskDistractionCid（センターディスプレイに描画することで生じるドライバディストラクションのリスク）は，駐車状態では危険を生じない。リスククラスに対して危険を生じない条件がある場合には，サービスコールの仕様において，それを明確にする。

プライバシ上のリスクの例として，アプリケーションを通して自車位置が漏洩することが考えられる。これを制御するために，自車位置・方位は，次のリスククラスを持つ。

| | |
|---|---|
| RiskPrivacy | プライバシ上のリスク |

**【補足説明】**

リスククラスは，ISO　26262と関連づけて説明すると，「ハザードを，その重大度や対策の方法・難易度によって分類したもの」と定義することができる。

リスクをどれだけまとめるか（逆に言うと，どれだけ分離するか）は難しい課題である。アプリケーションを想定し，リスクの重大度と，ビークルOSおよびアプリケーションでの対策方法を検討した上で，決める必要がある。

### 2.5.2.2. 車両の構成記述でのリスク記述

車両によって備えている安全機能が異なることから，オブジェクトを操作することに伴うリスクも車両によって異なる。例えば，ウィンドウに挟み込み防止機能を持つ車両は，RiskPinchに対応できているため，アプリケーションがRiskPinchのリスクを生じる可能性がある操作をしても危険を生じない。

車両によるリスクの違いを表現するために，車両の構成記述では，各オブジェクト（の各インスタンス）に対して，車両およびビークルOSで対応できていない（危険を防止できていない）リスククラスの集合を記述する。

### 2.5.2.3. アプリケーションでのリスクへの対応とOEMによる審査

車両およびビークルOSで対応できていないリスクのあるAPIを使用したいアプリケーションは，アプリケーションでリスクに対応する必要がある。例えば，RiskPinchのリスクがあるウィンドウをアプリケーションが閉める場合，音声等で「これからウィンドウを閉めます」と伝え，ユーザがいつでも閉動作を停止できる手段を提供して，閉動作を行うといった方法が考えられる。リスクへの対応方法には，「ユーザに制限事項を説明する」や「アプリケーション提供者が事故に対する責任を取る」といった方法も含まれる。

アプリケーションでリスクに対応した場合には，アプリケーションのリスク制御情報に，各オブジェクトクラスに対して，アプリケーションで対応したリスククラスの集合を記述する。リスク制御情報には，対象のオブジェクトクラス毎に，アプリケーションが使用できる操作優先度の最高値も記述する。

OEM（またはOEMから委託された者）は，使用できる操作優先度の最高値を考慮に入れて，アプリケーションによるリスクへの対応が十分であるかを審査し，十分であると判断した場合には，リスク制御情報に署名を付与する。

**【補足説明】**

アプリケーションのリスク制御情報では，対応したリスククラスの集合と使用できる操作優先度の最高値を，オブジェクトインスタンス毎ではなく，オブジェクトクラス毎に記述する。これは，オブジェクトインスタンス毎に記述するとした場合，リスク制御情報が車両に依存したものとなり，アプリケーションの

審査の工数が大きくなるためである。

OEMによる審査は，すべてのリスクに対する責任を，アプリケーション提供者とユーザに移転できる場合（例えば，プライバシ上のリスクのみがある場合）には，必須ではない。また，OEM間で審査基準を統一できる場合には，複数のOEMに対して一括して行うことができる。一方，国や地域によって法制度や規制が異なるため，国や地域毎に行うことが必要と思われる。

OEMによる審査において，各リスククラスへの対応が十分であると判断する基準は，本仕様のスコープ外である。

#### 2.5.2.4. ユーザの承認

ビークルOSは，リスクのあるAPIを使用するアプリケーションのインストール時（または，最初の実行時）に，そのリスクに対応する責任が，OEMからアプリケーション提供者とユーザに移転されていることに関して承認を求める。

例えば，RiskPinchのリスクがあるウィンドウをアプリケーションが閉める場合には，以下のようなメッセージによりユーザの承認を求める。

> 本アプリケーションは，ウィンドウの閉動作を行います。動作中，身体の一部や物品等が挟まれるおそれがあります。周囲の安全を確認し，異常時は直ちに停止してください。
>
> 本アプリケーションの使用により生じた損害について，［OEM］は責任を負いません。責任および補償の範囲は，［アプリケーション提供者］の利用規約に従います。
>
> アプリケーションによるウィンドウの閉動作を許可しますか？

ユーザの承認結果は，ユーザ個人に紐づいたユーザプリファレンスデータ内に格納する。ユーザプリファレンスデータ内に格納した承認結果は，設定などの機能により，ユーザが変更できるようにする。

OEMの純正アプリケーション（標準アプリケーションを含む）については，アプリケーション毎のユーザの判断を省略できるようにする。プライバシ上のリスクについては，OEMは，車両を販売する時点で，ユーザから包括的な承認をもらっていることを想定している。

#### 2.5.2.5. 操作可否の決定

ビークルOSは，以下の条件を満たす場合に，何らかの危険を生じる可能性があるオブジェクト操作をアプリケーションに許可する。

**(a)** 車両およびビークルOSで対応できていないリスククラスの集合が，アプリケーションで対応したリスククラスの集合に含まれている

**(b)** ユーザが，それらのリスクに対応する責任がアプリケーション提供者とユーザに移転されていることを承認している

**(c)** 操作に使用する操作優先度が，アプリケーションが使用できる操作優先度の最高値と同じかそれより低い

いずれかの条件が満たされない場合，危険を生じる可能性がある操作は禁止される。具体的には，禁止された操作を要求するサービスコールは，(a)の条件が満たされない場合はE_RISK_APPLICATIONエラーを，(b)の条件が満たされない場合はE_RISK_USERエラーを，(c)の条件が満たされない場合はE_DENIED_PRIORITYエラーを返す。

なお，前述の通り，危険を生じる可能性があるかは，操作の種類（サービスコールとそのパラメータ）や車両状態に依存する。

必要なサービスコール以外は呼び出せないようにするアクセス制御の機能によりサービスコールの呼び出しが拒否された場合，戻り値はE_DENIED_ACCESSとする。

# Chapter 3. 共通規定

この章では，複数のオブジェクトのAPIに共通する事項について規定する。データ型の名称中の*xxx*は，オブジェクトの略称を示す。

## 3.1. 共通データ型

この節では，複数のオブジェクトのAPIに共通に使用するデータ型について規定する。

#### (1) 戻り値のデータ型

サービスコールの戻り値を表すデータ型として，ReturnTypeを用意する。現時点の定義は以下の通り。今後，必要に応じてエラーコードを追加する。

```
type ReturnType = enum {
    E_OK,                   // 正常終了
    E_INVALID_ID,           // IDが不正
    E_INVALID_HANDLE,       // ハンドルが不正
    E_INVALID_PARAMETER,    // その他のパラメータが不正
    E_RISK_APPLICATION,     // アプリケーションが対応していないリスクがある
    E_RISK_USER,            // ユーザがリスクを承認していない
    E_DENIED_PRIORITY,      // 使用できない操作優先度
    E_DENIED_ACCESS,        // アクセスが認められていない
    E_OBJECT_FAULT,         // 対象オブジェクトが故障中
    E_OBJECT_LOCKED,        // 対象オブジェクトがロックされている
    E_OBJECT_STATUS,        // 対象オブジェクトが指定された操作をできない状
                            // 態にある
    E_INFEASIBLE,           // 指定された操作が現在の状態では実現できない
    E_NO_DATA,              // データがない
    E_NO_MEMORY,            // メモリ不足
    E_NOT_SUPPORTED,        // サポートされていない機能
    E_COMMUNICATION,        // 通信エラー
    E_NOT_OK                // その他のエラー
}
```

E_COMMUNICATIONは，サービスコールが複数のECUで実現される場合で，他のECUとの通信エラーや他のECUのダウンによりサービスコールが正常に実行できなかったことを示す。

E_NO_MEMORY，E_NOT_SUPPORTED，E_COMMUNICATIONは，すべてのサービスコールで発生する可能性があるため，個々のサービスコールが返すエラーコードには列挙しない。

物理APIにおいて，エラーコードを整数値で表現する場合には，ReturnTypeを32ビットの符号付き整数型で実現する。

#### (2) インスタンスIDのデータ型

オブジェクトのインスタンスIDを表すデータ型として，IdTypeを用意する。定義は以下の通り。

```
type IdType = UInt16 ([1,65535]);
```

オプショナル型をサポートしていないプログラミング環境では，IdType?型のnullを表すために，値0（ID_NULL）を使用する。すなわち，ID_NULLは，IDが指定されていないことを示す。

#### (3) 操作優先度のデータ型

操作優先度を表すデータ型として，PriorityTypeを用意する。操作優先度は，値が小さい方が優先度が高い。定義は以下の通り。

```
type PriorityType = UInt8 ([1,100]);
```

オプショナル型をサポートしていないプログラミング環境では，PriorityType?型のnullを表すために，値0（PRIORITY_NULL）を使用する。すなわち，PRIORITY_NULLは，操作優先度が指定されていないことを示す。

#### (4) アプリケーションIDのデータ型

アプリケーションのインスタンスを識別するためのIDをアプリケーションIDと呼び，それを表すデータ型として，ApplicationIdTypeを用意する。具体的な型定義は，物理APIにおいて規定する。

標準制御（OEMが自動車の出荷時に組み込んでいる制御ロジック）に対しても，アプリケーションIDを割り当てる。

#### (5) オブジェクトを表すデータ型

オブジェクト種別を表すデータ型として，ObjectKindTypeを用意する。定義は以下の通り（列挙子は順次増やしていく）。

```
type ObjectKindType = enum {
    Vehicle,
    VersionOSDVI,
    VehicleId,
    VehicleSpec,
    Cabin,
    Door,
    Window,
    CargoSpace,
    Body,
    Infotainment,
    HMI,
    Motion,
    CurrentLocation,
    Driver,
    Occupant,
    SurroundModel,
    Atmosphere,
    .........
};
```

関連するオブジェクトを表すデータ型として，RelatedObjectTypeを用意する。定義は以下のいずれかから，物理APIで選択する。

```
type RelatedObjectType = String;
```

```
type RelatedObjectType = struct {
    objectKind : ObjectKindType,
    instanceName : String?
};
```

```
type RelatedObjectType = struct {
    objectKind : ObjectKindType,
    instanceId : IdType?
};
```

instanceNameおよびinstanceIdは，マルチインスタンスの場合のみ値を持つ。

YAML形式で記述した構成記述ファイル中では，第1の形式（つまり，文字列）で記述するが，構成情報を参照するサービスコールは，物理APIで定められた形式に変換して返す。

#### (6) オブジェクトの場所を表すデータ型

キャビン内のオブジェクトの場所を表すデータ型として，PlacementTypeを用意する。定義は以下の通り。

```
type ZoneType = enum {
    Left,       // 左
    Right,      // 右
    Center,     // 中央
    Top,        // 上
    Bottom      // 下
};

type PlacementType = struct {
    row : UInt8,        // 座席の何列目か
    zone : ZoneType     // 左右上下方向の位置
};
```

#### (7) タイムスタンプ

相対時刻を表すデータ型として，TimestampMsTypeを用意する。 TimestampMsTypeで表されるタイムスタンプは，1ミリ秒単位で増加するカウン タであり，最大値に達すると0に戻る。時刻の差分の計算は，ラップアラウン ドを考慮して行う必要がある。

```
type TimestampMsType = UInt32;
```

## 3.2. すべてのオブジェクトが持つサービスコール

この節では，原則としてすべてオブジェクトが持つサービスコールについて規定する。

### getConfig

オブジェクトの構成情報の参照（シングルトンの場合）

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | *xxx*ConfigType | 対象オブジェクトのすべての構成情報 |

**【エラーコード】**

なし

**【機能】**

対象オブジェクトのすべての構成情報を返す。

### getConfigAll

オブジェクトのすべてのインスタンスの構成情報の参照（マルチインスタンスの場合）

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | *xxx*ConfigType[] | 対象オブジェクトのすべてのインスタンスのすべての構成情報 |

**【エラーコード】**

なし

**【機能】**

対象オブジェクトのすべてのインスタンスのすべての構成情報を返す。

### getStatus

オブジェクト状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 状態参照対象のインスタンスID（マルチインスタンスの場合のみ） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | *xxx*StatusType | 対象オブジェクトの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外（マルチインスタンスの場合のみ） |

**【機能】**

状態参照対象のオブジェクト（のインスタンス）の状態を返す。

## notify

イベント通知の要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のインスタンスID（マルチインスタンスの場合のみ） |
| eventFilter | enumSet(*xxx*EventKindType)? | 通知するイベントの種類の集合 |

※ 通知するイベントの種類をより細かく指定するために，パラメータを追加しても良い。

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 生成したイベントキューの通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外（マルチインスタンスの場合のみ） |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のオブジェクト（のインスタンス）で発生したeventFilterで指定した種別のイベントの通知を要求する。イベントを格納するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種別のイベントの通知を要求する。

### unnotify

イベント通知の停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 通知停止するイベントキューの通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。イベントを格納するためのイベント キューを削除する。

### getEvent

イベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントを取得するイベントキューの通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | *xxx*EventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

**【補足説明】**

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意しなくても良い。

## 3.3. すべてのオブジェクトが持つイベント

この節では，原則としてすべてオブジェクトが持つイベントについて規定する。

オブジェクトに対するイベント種別には，原則として，以下のイベント種別を含む。

| | |
|---|---|
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

EventOverflowは，イベントキューがオーバフローし，古いイベントが削除された時に，イベントキューの先頭に入れられる。

ConfigChangedは，そのオブジェクトの構成情報が変化した時に発生する。マルチインスタンスオブジェクトにおいて，インスタンスが増えたことを知るためには，イベント通知対象のインスタンスIDを指定せずにイベント通知を要求する必要がある。

# 3.4. LockableObject

API仕様の記述を簡潔にするために，標準的なロック機能を，LockableObjectとして定義する。オブジェクト毎の仕様に「このオブジェクトはLockableObjectである」という記述があると，ここで説明する標準的なロック機能を持つことが意味する。

LockableObjectは，以下のAPIを持つ。

- ロック/ロック解除のサービスコールを持つ
- 状態参照によりロック状態を参照できる
- ロック操作に関するイベントを持つ

## 3.4.1. LockableObjectの機能

LockableObjectは，他のアプリケーションがオブジェクトの操作をすることを禁止するためのロック機能を持つ。ロックは，ハードウェアで行われる場合（例えば，ドアロック）と，ソフトウェアでのみ行われる場合がある。ロック機能の詳細については，Section 2.5.1.2を参照のこと。

オブジェクトが異常状態（故障中やオブジェクト依存の異常状態）になった場合，ロックは解除され，その間はロックを取得することができない。

アプリケーションが終了すると，そのアプリケーションが取得していたロックは，ロック解除される。

## 3.4.2. LockableObjectの状態

LockableObjectは，オブジェクト状態を参照するサービスコールにより，ロック状態を参照することができる。

ロック状態には，以下の情報を含む。

- ロックされているか否か（ロックされている場合にtrue）
- ロックしたアプリケーションID（ロックされている場合のみ）
- ロックを取得した操作優先度（ロックされている場合のみ）

オブジェクトが異常状態の間は，ロックされているか否かはfalseになる。

### 3.4.3. LockableObjectのデータ型

LockableObjectの状態には，以下のデータ型で表されるロック状態を含む。

```
type LockStatusType = struct {
    locked : Bool,                          // ロックされているか否か
    lockApplication : ApplicationIdType?,   // ロックしたアプリケーションID
    lockPriority : PriorityType?            // ロックを取得した操作優先度
};
```

lockApplicationとlockPriorityは，lockedがtrueの場合にのみ値を持つ。

### 3.4.4. LockableObjectのイベント

LockableObjectに対するイベント種別には，以下のイベント種別を含む。

| | |
|---|---|
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |

OwnLockRevokedは，自アプリケーションがロックしているオブジェクトが，他のアプリケーションが高い操作優先度で操作またはロック取得したことや，異常状態になったことにより，強制的にロック解除された時に発生する。

OtherLockReleasedは，他のアプリケーションがロックしているオブジェクトが，何らかの理由でロック解除され，その結果，ロックを取得できる状態になった時に発生する。ただし，異常状態の間はロックを取得できないため，OtherLockReleasedは，異常状態になった時ではなく，異常状態から回復した時に，異常状態になった時にロックが取得されていたか否かにかかわらず発生する。なお，ロックしているアプリケーションが変わった時には，OtherLockReleasedは発生しない。

ロック状態の遷移と発生するイベントの関係をFigure 8に示す。

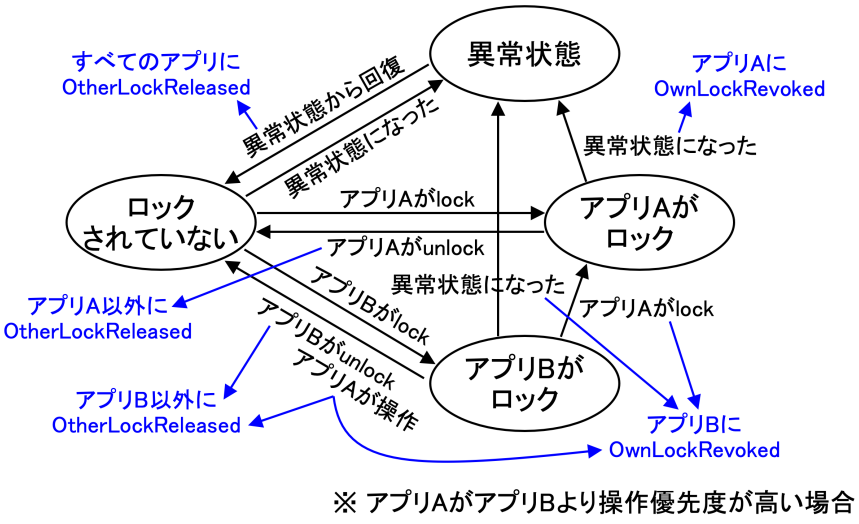



*Figure 8.* ロック状態の遷移とイベント

**【補足説明】**

OtherLockReleasedのユースケースとして，自アプリケーションがオブジェクトを操作またはロックできるようになったタイミングを知るために使用することを想定している。

## 3.4.5. LockableObjectのサービスコール

LockableObjectは，以下のサービスコールを持つ。

### lock

ロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | ロック対象のインスタンスID（マルチインスタンスの場合のみ） |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外（マルチインスタンスの場合のみ） |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のオブジェクトが，priorityと同じかそれより高い優先度でロックされている（自アプリケーションが，priorityと同じかそれより高い優先度でロックしている場合を含む） |
| E_OBJECT_FAULT | 対象オブジェクトが故障中（対象オブジェクトに故障中の状態がある場合のみ） |
| E_OBJECT_STATUS | 対象オブジェクトがロックできない状態にある（対象オブジェクトにそのような状態がある場合のみ） |

**【機能】**

ロック対象のオブジェクト（のインスタンス）を，指定した操作優先度でロックする。

ロックすることで，制御対象の状態（ロック状態を除く）は変化しない。すなわち，現在の動作をそのまま継続する。

### unlock

ロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | ロック解除対象のインスタンスID（マルチインスタンスの場合のみ） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外（マルチインスタンスの場合のみ） |
| E_OBJECT_STATUS | 自アプリケーションが，ロック解除対象のオブジェクトをロックしていない |

**【機能】**

ロック解除対象のオブジェクト（のインスタンス）を，ロック解除する。

# 3.5. MovableObject

API仕様の記述を簡潔にするために，移動する可動部を持つオブジェクトに共通の機能を，MovableObjectとして定義する。オブジェクト毎の仕様に「このオブジェクトはMovableObjectの機能を持つ」という記述があると，ここで説明する機能を持つことを意味する。以下でMovableObjectと言った場合には，MovableObjectの機能を持つオブジェクトを指す。

MovableObjectは，以下のAPIを持つ。

- 状態参照により移動に関する状態を参照できる。
- 移動を開始/停止するサービスコールを持つ。
- 移動に関する状態の変化のイベントを持つ。

## 3.5.1. MovableObjectの機能

MovableObjectは一次元に移動する可動部を持つ。二次元（またはそれ以上）に移動する可動部を持つオブジェクトも，2つの次元それぞれがMovableObjectと同等の機能を持つという形で規定できる。

MovableObjectの可動部の移動には，時間がかかると想定している。そのため，MovableObjectの状態に，可動部の現在位置に加えて，目標位置を持つ。

現在位置と目標位置は，パーセンテージで表す。片方の移動端を0，反対側の移動端を100で表し，どちらの移動端を0とするかはオブジェクト毎に定める。現在位置と目標位置は，不明を示す値（UNKNOWN），または，0で表される移動端でないことを示す値（NONZERO）になる場合がある。

現在位置の検知精度は，車両に依存する。現在位置が全くわからない場合，現在位置としてUNKNOWNを返す。現在位置が，0で表される移動端でないことのみがわかっている場合，現在位置としてNONZEROを返す。現在位置が，どちらの移動端でもないことのみがわかっている場合の特殊値は用意していないが，検知精度は車両依存としていることから，現在位置として50を返すことができる。

MovableObjectが正常動作している（故障中やオブジェクト依存の異常状態でない）場合でも，現在位置がUNKNOWNやNONZEROになる場合がある（現在位置の学習中の状況などを想定）。目標位置は，UNKNOWNになる場合がある（故障からの回復の直後などを想定）が，NONZEROになることはない。

MovableObjectが正常動作している場合，現在位置と目標位置がUNKNOWNやNONZEROでなく，現在位置と目標位置が一致していない場合，目標位置に向けて可動部を移動する。現在位置がUNKNOWNまたは

NONZEROの場合，目標位置が0または100であれば，目標位置に向けて可動部が移動し，他の目標位置では可動部は移動しない。目標位置がUNKNOWNの場合，可動部は移動しない。

可動部の移動中に，現在位置と目標位置が一致したら，移動を停止する。現在位置の検知精度が低いなどの理由で，現在位置が目標位置に完全には一致させられない場合，目標位置に到達した時点で，目標位置を変更して，現在位置に一致させる。

目標位置の設定精度は，車両に依存する。目標位置に0または100を指定した場合，指定された移動端まで移動する。目標位置にそれ以外の値を指定した場合，最終的な位置は移動端にはならない（移動端まで移動してから，少し戻る可能性はある）。

可動部の移動停止にも，時間がかかると想定している。そのため，移動停止サービスコールは，目標位置を現在位置に設定して移動停止を要求するだけで，移動が停止するまで待つわけではない。移動が停止した時点で，再度，目標位置を変更して，現在位置に一致させる。

MovableObjectは，オプションで，可動部をどのように移動させるか（これを移動プロファイルと呼ぶ）を指定する機能を持つ。移動プロファイルでは，移動速度に基づく3段階（標準/高速/低速）の指定を標準とするが，オブジェクトによっては，さらに多様な動作パターンを指定できる場合もある。また，OEM固有の移動プロファイルを指定できるようにしても良い。移動プロファイルを指定しない場合，標準的な移動プロファイルが適用される。なお，オブジェクトが移動プロファイルをサポートしていない場合，移動プロファイルの指定は無視される。

### 3.5.2. MovableObjectの状態

MovableObjectは，オブジェクト状態を参照するサービスコールにより，移動に関する状態を参照することができる。

移動に関する状態には，以下の情報を含む。

- 移動主状態
- 現在位置
- 目標位置

移動主状態は，以下のいずれかの状態を表す。

- 停止中
- 位置のパーセンテージが大きくなる方へ移動中
- 位置のパーセンテージが小さくなる方へ移動中
- 故障中

ここでは，恒久故障（修理するまで治らない）と一時故障（自然に治る可能性がある）は区別しない。

移動主状態とその間の状態遷移をFigure 9に示す。

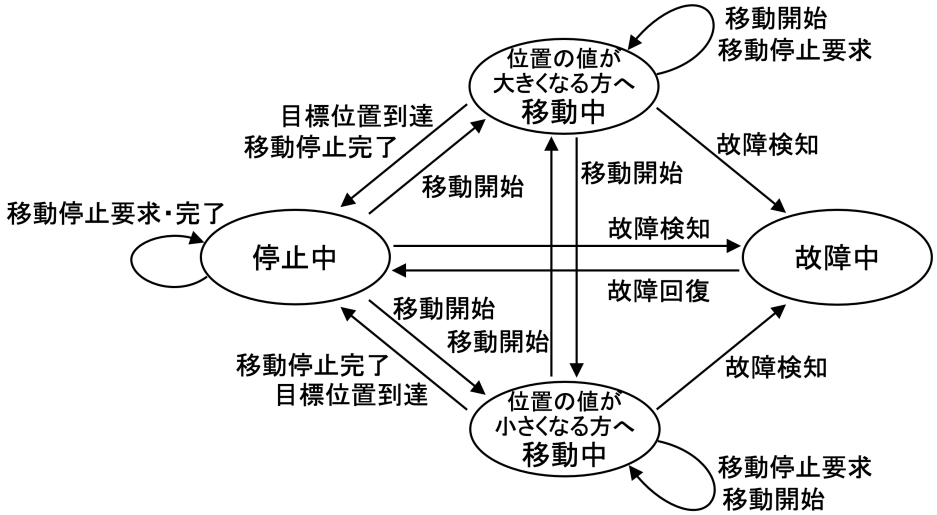



*Figure 9.* 移動主状態の状態遷移

以上で挙げた以外に，オブジェクト固有の主状態や状態遷移を追加しても良い。

### 3.5.3. MovableObjectのイベント

MovableObjectに対するイベント種別には，以下のイベント種別を含む。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |

この内，イベント種別がControlledByOtherかTargetReachedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplicatio n | ApplicationIdType | 操作したアプリケーション |

ControlledBySelfとControlledByOtherは，移動開始/停止サービスコールが呼び出され，正常終了した場合（戻り値がE_OKの場合）に発生する。

TargetReachedは，移動開始サービスコールによって指定された目標位置に到達して停止した時と，移動停止サービスコールによって要求された移動停止が完了した時に発生する。現在位置を目標位置に指定して呼び出した移動開始サービスコールや，停止中に呼び出した移動停止サービスコールに対しても発生する。

なお，ControlledBySelf，ControlledByOther，TargetReachedは，オブジェクトの状態が変化しない場合にも発生する（イベントの発生条件に関する一般規則の例外）。

他の理由で移動主状態が変化するオブジェクトの場合には，その移動主状態の変化に関するイベント種別を追加する。

MovableObjectに対するイベントが連続して発生した場合に，一部のイベントが抜ける場合がある。ただし，ControlledBySelfが抜けることはない。また，連続して発生した移動に関するイベントの中で，最後のイベントが抜けることはない。

**【補足説明】**

MovableObjectに対するイベントのユースケースとして，次の2つを想定している。

**(a)** 自アプリケーションによる移動開始/停止要求が完了したこと（または，他のアプリケーション等によって移動要求がオーバライドされたこと）を知るため。具体的な方法は，startMove/stopMoveのAPIの使用方法の項で説明する。

**(b)** ウィンドウの状態を常に知りたい場合に，状態を参照するタイミングを知るため。ただし，現在位置のみの変化ではイベントは発生しないため，ウィンドウの開閉動作中の現在位置を常に知りたい時には，開閉動作中はポーリングすることが必要である。

## 3.5.4. MovableObjectのデータ型

### (1) 位置を表すデータ型

MovableObjectの可動部の位置をパーセンテージで表すデータ型として，PositionTypeを用意する。0と100の意味は，オブジェクト毎に定める。定義は以下の通り。

```
type PositionType = Int8 ([0,100] | UNKNOWN(-1) | NONZERO(-2))
```

UNKNOWNは位置が不明であることを，NONZEROは0で表される移動端でないことを示す。

### (2) 移動に関する状態のデータ型

MovableObjectは，移動主状態を表すデータ型として，オブジェクト毎に，*xxx*MainStatusTypeを用意する。定義は以下のようになる。

```
type xxxMainStatusType = enum {
    Stopped,        // 停止中
    Increasing,     // 位置のパーセンテージが大きくなる方へ移動中
    Decreasing,     // 位置のパーセンテージが小さくなる方へ移動中
    Fault,          // 故障中
    .........
};
```

IncreasingとDecreasingは，オブジェクト毎に，その動きに合わせて名前を変えても良い。また，オブジェクト毎に，この他の主状態を追加しても良い。

MovableObjectの状態には，移動に関する状態を示す以下のフィールドを含む。

```
type xxxStatusType = struct {
    .........
    mainStatus : xxxMainStatusType, // 移動主状態
    position : PositionType,            // 現在位置
    targetPosition : PositionType,      // 目標位置
    .........
};
```

#### (3) 移動プロファイルのデータ型

MovableObjectは，移動プロファイルを表すデータ型として，オブジェクト毎に，*xxx*MoveProfileTypeを用意する。定義は以下のようになる。

```
type xxxMoveProfileType = enum {
    Standard,       // 標準的な移動
    Fast,           // 高速で移動
    Slow,           // 低速で移動
    .........
};
```

#### (4) MovableObjectのイベントのデータ型

MovableObjectのイベント種別には，MovableObjectが持つイベント種別を含む。

```
type xxxEventKindType = enum {
    .........
    ControlledBySelf,       // 自アプリケーションによる制御
    ControlledByOther,      // 他のアプリケーションによる制御
    TargetReached,          // 目標位置到達，移動停止完了
    FaultDetected,          // 故障検知
    FaultRecovered,         // 故障回復
    .........
};
```

MovableObjectのイベントのデータ型は，以下のような定義となる。

```
type xxxEventType = struct {
    instanceId : IdType,    // マルチインスタンスオブジェクトの場合のみ
    eventInfo : variant(xxxEventKindType) {
        .........
        ControlledByOther {
            sourceApplication : ApplicationIdType
        }
        TargetReached {
            sourceApplication : ApplicationIdType
        }
        .........
    }
};
```

### 3.5.5. MovableObjectのサービスコール

MovableObjectは，以下のサービスコールを持つ。

## startMove

移動開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のインスタンスID（マルチインスタンスの場合のみ） |
| targetPosition | PositionType | 目標位置 |
| moveProfile | *xxx*MoveProfileType? | 移動プロファイル |
| priority | PriorityType? | 操作優先度（LockableObjectの場合のみ） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

オブジェクト毎に規定する

**【機能】**

目標位置をtargetPositionに設定して，操作対象のMovableObject（のインスタンス）の可動部の移動を開始する。

可動部の移動には時間がかかると想定しているため，このサービスコールは移動開始を要求するだけで，目標位置に到達するまで待つわけではない。

可動部が停止している状態で，targetPositionに現在位置を指定してこのサービスコールを呼び出した場合でも，ControlledBySelfとTargetReachedは，この順序で発生する。

**【APIの使用方法】**

自アプリケーションが発行したstartMoveにより要求した可動部の移動が，目標位置に到達して完了したか，または，他のアプリケーション等によって移動要求がオーバライドされたかを知るためには，次の手順をとる。

- 移動に関するイベントの通知を要求しておく。
- この時点までに生成されたイベントを読み出しておく。
- startMoveを発行する。
- ControlledBySelfを受け取るまで，イベントを読み捨てる。
- 関連情報のsourceApplicationが自アプリケーションであるTargetReachedを受け取れば，目標位置に到達して移動が完了したと判断する。
- それ以外のイベントを受け取れば，他のアプリケーション等によって移動要求がオーバライドされたと判断する。

なお，この方法は，1つのアプリケーションが同一のオブジェクトに対して並行してサービスコールを呼び出すことは想定していない。サービスコールを並行して呼び出さないようにするのは，アプリケーションの責任である。

**【補足説明】**

制御精度が低いなどの理由で，移動が完了した時点の現在位置が，指定した目標位置に完全には一致しないことを許容しているため（この場合，移動が完了した時点で，目標位置を変更して，現在位置に一致させる），現在位置と自アプリケーションが指定した目標位置が一致したことをもって，自アプリケーションが要求した移動が完了したと判断することはできない。

## stopMove

移動停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のインスタンスID（マルチインスタンスの場合のみ） |
| priority | PriorityType? | 操作優先度（LockableObjectの場合のみ） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

オブジェクト毎に規定する

**【機能】**

目標位置を現在位置に設定して，操作対象のMovableObject（のインスタンス）の可動部の移動停止を要求する。

可動部の移動停止には時間がかかると想定しているため，このサービスコールは移動停止を要求するだけで，移動が停止するまで待つわけではない。そのため，このサービスコールからのリターン後に主状態を参照しても，停止中（STOPPED）になっているとは限らない。

可動部が停止している状態でこのサービスコールを呼び出した場合でも，ControlledBySelfとTargetReachedは，この順序で発生する。

**【APIの使用方法】**

自アプリケーションが発行したstopMoveにより要求した可動部の移動停止が完了したか，または，他のアプリケーション等によって停止要求がオーバライドされたかを知るためには，次の手順をとる。

- 移動に関するイベントの通知を要求しておく。
- この時点までに生成されたイベントを読み出しておく。
- stopMoveを発行する。
- ControlledBySelfを受け取るまで，イベントを読み捨てる。
- 関連情報のsourceApplicationが自アプリケーションであるTargetReachedを受け取れば，停止が完了したと判断する。
- それ以外のイベントを受け取れば，他のアプリケーション等によって停止要求がオーバライドされたと判断する。

なお，この方法は，1つのアプリケーションが同一のオブジェクトに対して並行してサービスコールを呼び出すことは想定していない。サービスコールを並行して呼び出さないようにするのは，アプリケーションの責任である。

# Chapter 4. コアビークルAPI

## 4.1. Vehicle（略称：Vehicle）

＜TBD＞

## 4.2. Vehicle.VersionOSDVI（略称：VersionOSDVI）

### 4.2.1. 位置付けと機能

Vehicle.VersionOSDVI（略称：VersionOSDVI）は，車両がサポートするOpen SDV API仕様のバージョンを参照するためのシングルトンオブジェクトである。

### 4.2.2. 構成情報

VersionOSDVIは，以下の構成情報を持つ。

| | | |
|---|---|---|
| release | String | バージョン番号（リリース年月） |
| stage | String? | 開発ステージ（α，βなど） |
| suffix | String? | プレリリース識別子（RC1など） |

### 4.2.3. サービスコール一覧

VersionOSDVIに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | VersionOSDVIの構成情報の参照 |

### 4.2.4. データ型の定義

#### 4.2.4.1. VersionOSDVIの構成情報（getConfigが返すデータ型）

```
type VersionOSDVIConfigType = struct {
    release : String,
    stage : String?,
    suffix : String?
};
```

### 4.2.5. サービスコールの詳細規定

#### getConfig

VersionOSDVIの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | VersionOSDVIConfigType | VersionOSDVIの構成情報 |

**【エラーコード】**

なし

**【機能】**

VersionOSDVIの構成情報を返す。

# 4.3. Vehicle.VehicleIdentification（略称：VehicleId）

## 4.3.1. 位置付けと機能

Vehicle.VehicleIdentification（略称：VehicleId）は，車両の識別情報を参照するためのシングルトンオブジェクトである。


COVESA VSS 6.0のVehicle.VehicleIdentificationに対して，車両の最大乗客数（vehicleMaxCapacity）を追加した。各構成情報の詳細は，COVESA VSSの仕様書に記載されている。

## 4.3.2. 構成情報

VehicleIdは，以下の構成情報を持つ。

| | | |
|---|---|---|
| vin | String? | ISO 3779で定義されるの車両識別番号（VIN） |
| wmi | String? | ISO 3780で定義される世界製造業者識別コード（WMI） |
| brand | String? | 車両のブランドまたは製造メーカ |
| model | String? | 車両のモデル |
| year | String? | 車両のモデルイヤー |
| acrissCode | String? | ACRISS車種分類コード |
| bodyType | String? | 車両のデザインとボディスタイル |
| dateVehicleFirstRegistered | String? | 車両の初回登録日（ISO 8601形式） |
| licensePlate | String? | 車両のナンバープレート |
| meetsEmissionStandard | String? | 車両が満たしている排出ガス基準 |

| | | |
|---|---|---|
| productionDate | String? | 製造日（ISO 8601形式） |
| purchaseDate | String? | 購入日（ISO 8601形式） |
| vehicleModelDate | String? | 車両モデルのリリース日（ISO 8601形式） |
| vehicleConfiguration | String? | 車両の構成を示す短い記述 |
| vehicleSeatingCapacity | Uint16? | 車両に着座可能な乗客数 |
| vehicleMaxCapacity | Uint16? | 車両の最大乗客数 |
| vehicleSpecialUsage | String? | 車両の特別な使用目的 |
| vehicleExteriorColor | String? | 外装の主たる色（基本色） |
| vehicleInteriorColor | String? | 内装の色または色の組み合わせ |
| vehicleInteriorType | String? | 内装のタイプまたは素材 |
| knownVehicleDamages | String? | 既知の損傷（修復済み，未修復の両方） |
| optionalExtras | String[] | 車両の装備オプション |

## 4.3.3. サービスコール一覧

VehicleIdに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | VehicleIdの構成情報の参照 |

## 4.3.4. データ型の定義

### 4.3.4.1. VehicleIdの構成情報（getConfigが返すデータ型）

```
type VehicleIdConfigType = struct {
    vin : String?,
    wmi : String?,
    brand : String?,
    model : String?,
    year : String?,
    acrissCode : String?,
    bodyType : String?,
    dateVehicleFirstRegistered : String?,
    licensePlate : String?,
    meetsEmissionStandard : String?,
```

```
    productionDate : String?,
    purchaseDate : String?,
    vehicleModelDate : String?,
    vehicleConfiguration : String?,
    vehicleSeatingCapacity : Uint16?,
    vehicleMaxCapacity : Uint16?,
    vehicleSpecialUsage : String?,
    vehicleExteriorColor : String?,
    vehicleInteriorColor : String?,
    vehicleInteriorType : String?,
    knownVehicleDamages : String?,
    optionalExtras : String[]
};
```

### 4.3.5. サービスコールの詳細規定

#### getConfig

VehicleIdの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | VehicleIdConfigType | VehicleIdの構成情報 |

**【エラーコード】**

なし

**【機能】**

VehicleIdの構成情報を返す。

## 4.4. Vehicle.Specification（略称：VehicleSpec）

### 4.4.1. 位置付けと機能

Vehicle.Specification（略称：VehicleSpec）は，車両の諸元を参照するためのシングルトンオブジェクトである。


COVESA VSS 6.0でVehicleの直下にある車両の諸元に関するデータを，Vehicle.Specificationで参照できるようにした。各構成情報の詳細は，COVESA VSSの仕様書に記載されている。

### 4.4.2. 構成情報

VehicleSpecは，以下の構成情報を持つ。

| | | |
|---|---|---|
| curbWeight | Uint16? | 整備重量（潤滑油や冷却水に加えて燃料を全量入れた状態で，乗員や貨物を含まない状態での車両重量。単位：kg） |
| grossWeight | Uint16? | 車両総重量（潤滑油や冷却水に加えて燃料を全量入れた状態で，乗員や貨物を満載した状態での車両重量。単位：kg） |
| maxTowWeight | Uint16? | 最大牽引重量（トレーラーの最大重量。単位：kg） |
| maxTowBallWeight | Uint16? | トレーラーのトウボールにかけられる最大垂直荷重（単位：kg） |
| length | Uint16? | 車両全長（単位：mm） |
| height | Uint16? | 車両全高（単位：mm） |
| widthExcludingMirrors | Uint16? | SAE J1100-2009 W144で定義されるミラーを含んだ車両全幅（単位：mm） |
| widthFoldedMirrors | Uint16? | SAE J1100-2009 W145で定義されるミラーを折りたたんだ状態での車両全幅（単位：mm） |
| turningDiameter | Uint16? | SAE J1100-2009 D102で定義される最小回転直径（単位：mm）。最小回転半径の2倍 |
| roofLoad | Int16? | 車両上部の貨物および設置物の許容総重量（単位：kg） |
| cargoVolume | Float32? | 貨物や荷物に利用できる容積（単位：l） |
| emissionsCO2 | Int16? | CO2排出量（単位：g/km） |

### 4.4.3. サービスコール一覧

VehicleSpecに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | VehicleSpecの構成情報の参照 |

### 4.4.4. データ型の定義

#### 4.4.4.1. VehicleSpecの構成情報（getConfigが返すデータ型）

```
type VehicleSpecConfigType = struct {
    curbWeight : Uint16?,
    grossWeight : Uint16?,
    maxTowWeight : Uint16?,
    maxTowBallWeight : Uint16?,
    length : Uint16?,
    height : Uint16?,
```

```
    widthExcludingMirrors : Uint16?,
    widthFoldedMirrors : Uint16?,
    turningDiameter : Uint16?,
    roofLoad : Int16?,
    cargoVolume : Float32?,
    emissionsCO2 : Int16?
};
```

### 4.4.5. サービスコールの詳細規定

#### getConfig

VehicleSpecの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | VehicleSpecConfigType | VehicleSpecの構成情報 |

**【エラーコード】**

なし

**【機能】**

VersionSpecの構成情報を返す。

## 4.5. Vehicle.Cabin（略称：Cabin）

＜TBD＞

## 4.6. Vehicle.Cabin.Door（略称：Door）

### 4.6.1. 位置付けと機能

Vehicle.Cabin.Door（略称：Door）は，車両のドア（客室）を操作するためのマルチインスタンスオブジェクトである。

#### 4.6.1.1. ドアの位置付けと性質

開閉できる車室の開口部の中で、車両の走行中は通常閉じておくもの。客室にアクセスでき人が通ることは想定している。

### Doorの機能

Doorは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉，100：全開）で表す。また，0に近づく動作を閉動作，100に近づく動作を開動作と呼ぶ。

挟み込み防止機能を持つドアは，可動部の閉動作中に挟み込みを検知すると，閉動作を停止し，開動作を開始する。挟み込み検知後にどの位置まで開くかは，車両依存である。

可動部が停止中の間も，マニュアル操作にて現在位置が変化することがあり得る。

#### 4.6.1.2. ドアに対する制御の調停

ドアに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，ドアを閉めるサービスコールが呼び出され，ドアが閉じる動作中に，開くサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，ドアは開く動作を始める。

また，ドアは標準的なロック機能を持つLockableObjectである。ドアには，機械的なロック機構は無いのが一般的であり，ソフトウェアでロック機能を実現することを想定している。

ドアが故障または挟み込みを検知した場合，ロックは解除され，故障中と挟み込み回避中は，ロックを取得することができない。

## 4.6.2. 機能クラス

Doorは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワードアであり，APIにより移動させることができる。ドアロックアンラッチ機能のみを有する者も含む |
| ClosedDetectable | ドアが全閉状態であることを検出することができる |

## 4.6.3. リスククラス

Doorは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | ドアを閉めることに伴う挟み込みのリスク |
| RiskOpen | ドアを開けることに伴う諸々のリスク |

**【今後の検討課題】**

RiskOpenをさらに分類するかは，今後の課題である。

## 4.6.4. 構成情報

Doorの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |

| | | |
|---|---|---|
| capabilities | enumSet (DoorCapabilityType) | そのドアが持つ機能 |
| riskClasses | enumSet(DoorRiskType) | そのドアのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.6.5. 状態

Doorの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | DoorMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |

ドアの主状態（mainStatus）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。また，位置のパーセンテージが大きくなる方へ移動中を開動作中，位置のパーセンテージが小さくなる方へ移動中を閉動作中と呼ぶ（図4-Y-1）。

[[Fig-図4-Y-1. ドアの主状態の状態遷移]] .図4-Y-1. ドアの主状態の状態遷移 image::../images/image/image1.png[width=50%]

ドアの現在位置（position）と目標位置（targetPosition）は，MovableObjectの現在位置と目標位置に関する規定に従う。

ドアロック/アンラッチのみの機能の場合は、開要求のみを受け付け、閉要求は受け付けないものとする。また、アンラッチ完了時をもって目標位置到達と考える。

## 4.6.6. サービスコール一覧

Doorに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのドアの構成情報の参照 |
| getStatus | ドアの状態の参照 |
| startMove | ドアの開閉の開始 |
| stopMove | ドアの開閉の停止 |
| lock | ドアのロックの取得 |
| unlock | ドアのロックの解除 |

| | |
|---|---|
| notify | ドアイベントの通知要求 |
| unnotify | ドアイベントの通知停止 |
| getEvent | ドアイベントの取得 |

### 4.6.7. イベント

ドアに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Doorに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

### 4.6.8. データ型の定義

#### 4.6.8.1. ドアの機能クラス

```
type DoorCapabilityType = enum {
    Movable,
    ClosedDetectable
};
```

#### 4.6.8.2. ドアに対するリスククラス

```
type DoorRiskType = enum {
    RiskPinch,
    RiskOpen
};
```

#### 4.6.8.3. ドアの構成情報（getConfigAllが返すデータ型）

```
type DoorConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(DoorCapabilityType),
    riskClasses : enumSet(DoorRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.6.8.4. ドア状態

```
type DoorMainStatusType = enum {
    FullyStopped,   // 停止中（全閉）
    UnlatchedStopped,   // 停止中（半ドア  or  ドア開）
    Opening,    // 開動作中
    Closing,    // 閉動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type DoorStatusType = struct {
    mainStatus : DoorMainStatusType,
    lockStatus : LockStatusType,
    position : PositionType,
    targetPosition : PositionType
};
```

#### 4.6.8.5. ドアイベント

```
type DoorEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
```

```
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type DoorEventType = struct {
    instanceId : IdType,
    eventInfo : variant(DoorEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.6.9. サービスコールの詳細規定

### getConfigAll

すべてのドアの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | DoorConfigType[] | すべてのドアに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのドアに関する構成情報を返す。

### getStatus

ドアの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のドアのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | DoorStatusType | 対象ドアの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のドアに関するすべての状態を返す。

## startMove

ドアの開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のドアのID |
| targetPosition | PositionType | 目標位置（開閉度） |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のドアが故障中 |
| E_OBJECT_STATUS | 操作対象のドアが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のドアが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し，操作対象のドアの開閉動作を開始する。

targetPositionに0を指定すれば閉動作の開始，100を指定すれば開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，ドアを開く方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskOpenに対応できていない場合でも，ドアを閉じる方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

ドアの開閉の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のドアのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のドアが故障中 |
| E_OBJECT_STATUS | 操作対象のドアが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のドアが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のドアの各次元方向への開閉動作の停止を要求する。

## lock

ドアのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のドアのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のドアが，priorityと同じかそれより高い優先度でロックされている |
| E_OBJECT_FAULT | ロック対象のドアが故障中 |
| E_OBJECT_STATUS | ロック対象のドアが挟み込み回避中 |

**【機能】**

ロック対象のドアを，指定した操作優先度でロックする。

## unlock

ドアのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のドアのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象のドアをロックしていない |

**【機能】**

ロック解除対象のドアを，ロック解除する。

## notify

ドアイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のドアのID |
| eventFilter | enumSet (DoorEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のドアで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

ドアイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

ドアイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | DoorEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.7. Vehicle.Cabin.DoorSwitch（略称：DoorSwitch）

## 4.7.1. 位置付けと機能

Doorを制御する物理的なスイッチを扱う。スイッチとDoorは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.7.1.1. Doorスイッチの位置付けと性質

Doorスイッチとは、各Doorに配置された乗員が物理操作できるDoor制御デバイスである。

スイッチはレバー形および2段階式である。

Doorスイッチは各Doorに配置され独立した系統を持つ。

スイッチとDoorは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、Door制御の主体にはならない。

### 4.7.1.2. Doorスイッチの機能

Doorスイッチは乗員の操作によるスイッチ入力をOSへ提供し、Doorオブジェクトへ制御要求を伝達する役割を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

### 4.7.1.3. Doorスイッチに対する制御の調停

Doorスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、Doorオブジェクトへルーティングされる。

調停はDoor側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、Door側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

### 4.7.2. 機能クラス

DoorSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | Door操作入力を提供する基本機能 |

### 4.7.3. 構成情報

DoorSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | DoorSwitchCategoryType | スイッチ系統（FrontRight　FrontLeft　RearRight　RearLeft） |
| capabilities | enumSet (DoorSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（OPEM側のみ、CLOSE側のみ、IsDoorChildLockEngagedの機能ON/OFFのスイッチかなど、OEM依存） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

### 4.7.4. 状態

DoorSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | DoorSwitchMainStatusType | スイッチの入力状態（OPEN側のみ、CLOSE側のみ、IsDoorChildLockEngagedの機能ON/OFFのスイッチかなど） |

### 4.7.5. サービスコール一覧

DoorSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのDoorスイッチの構成情報の参照 |
| getStatus | Doorスイッチの状態参照 |
| notify | Doorスイッチイベントの通知要求 |
| unnotify | Doorスイッチイベントの通知停止 |
| getEvent | Doorスイッチイベントの取得 |

## 4.7.6. イベント

Doorスイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

DoorSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | 各ドアに対応するスイッチのmainStatus入力状態が変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | DoorSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.7.7. データ型の定義

### 4.7.7.1. Doorスイッチの機能クラス

```
type DoorSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.7.7.2. スイッチ系統

```
type DoorSwitchCategoryType = enum {
    FrontRight, // FrontRightDoorスイッチ系統
    FrontLeft,  // FrontLeftDoorスイッチ系統
    RearRight,  // RearRightDoorスイッチ系統
    RearLeft    // RearLeftDoorスイッチ系統
};
```

### 4.7.7.3. Doorsスイッチの入力状態

```
type DoorSwitchMainStatusType = enum {
    False,  // OFF - Door停止
    OPEN,   // OPEN - DoorOPEN
```

```
    CLOSE,  // Close - DoorClose
    ChildLockON,    // IsDoorChildLockEngagedの機能ON
    ChildLockOFF    // IsDoorChildLockEngagedの機能OFF
};
```

### Doorスイッチ構成情報（getConfigAll が返すデータ型）

```
type DoorSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : DoorSwitchCategoryType,
    capabilities : enumSet(DoorSwitchCapabilityType),
    switchType : String,
    displayName : String
};
```

### 4.7.7.4. Doorスイッチ状態

```
type DoorSwitchStatusType = struct {
    mainStatus : DoorSwitchMainStatusType
};
```

Doorスイッチの状態情報

### 4.7.7.5. Doorスイッチイベント種別

```
type DoorSwitchEventKindType = enum {
    MainStatusChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type DoorSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : DoorSwitchCategoryType,
    eventInfo : variant(DoorSwitchEventKindType) {
        MainStatusChanged {
            newState : DoorSwitchMainStatusType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
```

```
};
```

## 4.7.8. サービスコールの詳細規定

### getConfigAll

すべてのDoorスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | DoorSwitchConfigType[] | すべてのDoorスイッチに関する構成情報 |

**【機能】**

すべてのDoorスイッチに関する構成情報を返す。

### getStatus

Doorスイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のDoorスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | DoorSwitchStatusType | 対象のDoorスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のDoorスイッチに関する状態を返す。


status には mainStatus（各Door系統のスイッチ入力状態）を含む。

### notify

Doorスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のDoorスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (DoorSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のDoorスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

Doorスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

Doorスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | DoorSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.8. Vehicle.Cabin.Horn（略称：Horn）

## 4.8.1. 位置付けと機能

Vehicle.Cabin.Horn（略称：Horn）は、車両のクラクション（警音器）を吹鳴させるためのシングルインスタンスオブジェクトである。

### 4.8.1.1. クラクションの位置付けと性質

クラクションとは、車両外部に対して可聴音を発する警報装置であり、車両の走行状態に関わらず吹鳴可能なものと定義する。

### 4.8.1.2. Hornの機能

Hornは、吹鳴状態と非吹鳴状態を持つオブジェクトである。Hornは、吹鳴開始要求を受けると吹鳴状態に遷移し、吹鳴停止要求を受けると非吹鳴状態に遷移する。

吹鳴中は、要求が継続している間、吹鳴状態を維持する。Hornは、警告状態を持つことができる。警告状態のHornは、アラート用途として断続的または規定パターンで吹鳴する。

警告状態が設定されている間は、吹鳴要求の有無に関わらず吹鳴を継続し、警告状態が解除されると非吹鳴状態に遷移する。

### 4.8.1.3. クラクションに対する制御の調停

クラクションに対する制御の調停には、標準的な調停機能（後勝ちの原則）が適用される。

すなわち、クラクションを吹鳴するサービスコールが呼び出され吹鳴中に、停止するサービスコールが呼び出されると、最初のサービスコールによる制御はキャンセルされ、クラクションは停止する。。

警告状態の設定要求または解除要求も、同様に後勝ちで調停される。

### 4.8.2. 機能クラス

Hornは，以下の機能クラスを持つ。

| | |
|---|---|
| VolumeControllable | 音量（Volume）を制御できる |

### 4.8.3. リスククラス

Hornは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskBlow | ユーザの意図しない吹鳴に伴う諸々のリスク |
| RiskUnblow | ユーザの意図した吹鳴をしないことに伴う諸々のリスク |
| RiskStop | 吹鳴が止まってしまうことに伴う諸々のリスク |
| RiskUnstop | 吹鳴が止まらないことに伴う諸々のリスク |

**【今後の検討課題】**

RiskHornとRiskUnstopに関して、現状は別のリスクとして検討しているが、今後の作業で分ける必要がないと判断した場合、統合する。

### 4.8.4. 構成情報

Hornの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| capabilities | enumSet (HornCapabilityType) | クラクションが持つ機能 |
| riskClasses | enumSet(HornRiskType) | クラクションのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は、YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは、この構成情報に対して、次の変換を行った構成情報を返す。

- mountedOnは、物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては、それを元に現在のロケールに対応する呼称の文字列を取得し、その文字列を返す。

### 4.8.5. 状態

Hornの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | HornMainStatusType | 主状態(吹鳴状態、非吹鳴状態、警告状態、故障状態を想定) |

| | | |
|---|---|---|
| time | TimeType | 吹鳴時間[ms] |
| priority | PriorityType? | 操作優先度 |
| volume | VolumeType | 音量[%] |
| pattern | enumSet(PatternType) | 吹鳴パターンの集合 盗難アラートでの使用を想定。 |

クラクションの主状態は、吹鳴状態もしくは非吹鳴状態に加えて、故障中の状態をとる。また、盗難アラートが作動している状態をを警告状態と呼ぶ。

### 4.8.6. サービスコール一覧

Hornに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | クラクションの構成情報の参照 |
| getStatus | クラクションの状態の参照 |
| StartHorn | 吹鳴開始 |
| StopHorn | 吹鳴停止 |
| SetVolume | 音量設定 |
| AlartHorn | 警告吹鳴 |
| notify | クラクションイベントの通知要求 |
| unnotify | クラクションイベントの通知停止 |
| getEvent | クラクションイベントの取得 |

### 4.8.7. イベント

クラクションに対するイベント種別には、どのアプリが動作しているのかを示すイベントと吹鳴完了を示すイベントに加えて、故障検知と故障から回復がある。

Hornに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetcountReached | 指定された回数分の吹鳴が完了 |
| targettimeReached | 指定された時間分の吹鳴が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplicatio n | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.8.8. データ型の定義

### 4.8.8.1. クラクションの機能クラス

```
type HornCapabilityType = enum {
    VolumeControllable
};
```

### 4.8.8.2. 吹鳴パターン種別

```
type PatternType = enum {
    BlowingTime,    // 吹鳴時間
    UnblowingTime,  // 吹鳴停止時間
    BlowingCount    // 吹鳴を繰り返す回数
};
```

### 4.8.8.3. クラクションに対するリスククラス

```
type HornRiskType = enum {
    RiskBlow,
    RiskUnblow,
    RiskStop,
    RiskUnstop
};
```

### 4.8.8.4. クラクションの構成情報（getConfigが返すデータ型）

```
type HornConfigType = struct {
    instanceId : IdType,
    capabilities : enumSet(HornCapabilityType),
    riskClasses : enumSet(HornRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.8.8.5. クラクションの状態

```
type HornMainStatusType = enum {
    Unblow, // 非吹鳴中
    Blowing,    // 吹鳴中
    Alerting,   // 警告中
    Fault   // 故障中
};
```

#### 4.8.8.6. クラクションイベント種別

```
type HornEventKindsType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetcountReached,
    targettimeReached,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type HornEventType = struct {
    instanceId : IdType,
    eventInfo : variant(HornEvenKindsType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.8.9. サービスコールの詳細規定

#### getConfig

クラクションの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | HornConfigType[] | クラクションに関する構成情報 |

**【エラーコード】**

| | |
|---|---|
| E_OK | 成功 |
| E_NOT_SUPPORTED | 未サポート（実装により返却されない場合あり） |

**【機能】**

クラクションに関する構成情報を返す。

## getStatus

クラクションの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | クラクションのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | HornStatusType | クラクションの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

クラクションに関するすべての状態を返す。

## StartHorn

吹鳴開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | クラクションのID |
| time | TimeType | 時間[ms] |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが、アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のクラクションが故障中 |
| E_OBJECT_LOCKED | 操作対象のクラクションが、他のアプリケーションにより、priorityと同じかそれより高い優先度でロックされている |

**【機能】**

非吹鳴中のクラクションに対して、指定された時間吹鳴することを要求する

## StopHorn

吹鳴停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | クラクションのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが、アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のクラクションが故障中 |
| E_OBJECT_LOCKED | 操作対象クラクションが、他のアプリケーションにより、priorityと同じかそれより高い優先度でロックされている |

**【機能】**

吹鳴中のクラクションに対して吹鳴停止を要求する。

## SetVolume

音量設定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | クラクションのID |
| volume | VolumeType | 音量[%] 法規対応のため要検討箇所。現状はどの%でも一定の音量とする。 |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | volume、priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | volume、priorityが、アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のクラクションが故障中 |
| E_OBJECT_LOCKED | 操作対象のクラクションが、他のアプリケーションにより、priorityと同じかそれより高い優先度でロックされている |

**【機能】**

クラクションの音量制御を要求する

## AlartHorn

警告吹鳴

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | クラクションのID |
| pattern | enumSet(PatternType) | 吹鳴パターンの集合 盗難アラートでの使用を想定。 |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | volume、priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |

| | |
|---|---|
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | volume、priorityが、アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のクラクションが故障中 |
| E_OBJECT_LOCKED | 操作対象のクラクションが、他のアプリケーションにより、priorityと同じかそれより高い優先度でロックされている |

**【機能】**

クラクションのアラート吹鳴を要求する

## notify

クラクションイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | クラクションのID |
| eventType | HornEventKindsType? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventTypeが有効範囲外 |

**【機能】**

イベント通知対象のクラクションで発生したeventTypeで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し、その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合、すべてのインスタンスで発生したイベントの通知を要求する。eventTypeを省略した場合、すべての種類のイベントの通知を要求する。

## unnotify

クラクションイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

クラクションイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |
| eventDetail | EventDetailType | イベント詳細情報 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから、先頭のイベントを取り出す。取り出したイベントは、イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは、用意する必要がない。

# 4.9. Vehicle.Cabin.InteriorLightSwitch（略称：InteriorLightSwitch）

## 4.9.1. 位置付けと機能

室内ライトを調整する物理的なスイッチを扱う。スイッチとライトは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。角度調整・折り畳み・ヒーティング・防眩など多様なスイッチ種別を包含可能。

### 4.9.1.1. 室内ライトスイッチの位置付けと性質

室内ライトスイッチとは、車両キャビン内に設置され、乗員が室内ライトの点灯・消灯や各種機能を物理操作できる入力デバイスである。

スイッチとライトは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは物理的な操作子を持つが、ライトの点灯状態は室内ライトオブジェクトが制御する。スイッチは入力源として機能し、ライト制御主体にはならない。

スイッチには角度調整、防眩、ヒーティングなどライト機能に対応する複数種類が存在し、どのライト種別に対応するかを構成情報として持つ。

#### 4.9.1.2. 室内ライトスイッチの機能

室内ライトスイッチは乗員の押下・操作による点灯／消灯入力（getSwitchIn）をOSへ提供し、室内ライトオブジェクトへ制御要求を伝達する役割を持つ。

スイッチ種別（switchType）に応じて、角度調整・折り畳み・防眩・ヒーティングなど特定ライト機能に対応する入力を生成し、capabilitiesで対応する機能範囲を示す。

スイッチの状態はmainStatusとして参照でき、正常・停止・故障などの状態監視が可能である。

#### 4.9.1.3. 室内ライトスイッチに対する制御の調停

室内ライトスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、室内ライトオブジェクトへルーティングされる。

調停は室内ライト側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、ライト側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

### 4.9.2. 機能クラス

InteriorLightSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchOnOffControl | 室内ライトをスイッチ入力により点灯または消灯させる |

### 4.9.3. 構成情報

InteriorLightSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchType | InteriorLightSwitchType | スイッチ種別 |
| lightType | InteriorLightType | 対応するライト種別（AmbientLight, InteractiveLightBar, DomeLight, GloveBoxLight など） |
| capabilities | enumSet (InteriorLightSwitchCapabilityType) | このスイッチが扱うライト機能の集合 |
| effect | String | エフェクト種別（OEM依存） |
| mountedOn | RelatedObjectType | 関連するオブジェクト（設置対象） |

### 4.9.4. 状態

InteriorLightSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| getSwitchIn | SwitchInputType | スイッチ入力（点灯/消灯など） |
| mainStatus | InteriorLightSwitchMainStatusType | スイッチの主状態（正常/停止/故障/電気故障など） |

### 4.9.5. サービスコール一覧

InteriorLightSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべての室内ライトスイッチの構成情報の参照 |
| getStatus | スイッチの状態参照 |
| getErrorString | エラー情報取得 |
| notify | 室内ライトスイッチイベントの通知要求 |
| unnotify | 室内ライトスイッチイベントの通知停止 |
| getEvent | 室内ライトスイッチイベントの取得 |

### 4.9.6. イベント

室内ライトスイッチに関するイベントとして、制御主体、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

InteriorLightSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

### 4.9.7. データ型の定義

#### 4.9.7.1. 室内ライトスイッチの機能クラス

```
type InteriorLightSwitchCapabilityType = enum {
    SwitchOnOffControl
};
```

#### 4.9.7.2. スイッチ状態

```
type InteriorLightSwitchMainStatusType = enum {
    Normal, // 正常状態
    Suspended,  // 停止状態（室内ライトスイッチでは通常想定しない）
    Fault,  // 故障状態（復帰しないことを想定）
    ElectricFault   // 電気故障（短絡/断線などのDTCリストを返す想定）
};
```

#### 4.9.7.3. スイッチ入力種別

```
type SwitchInputType = enum {
    LightOn,    // 点灯
    LightOff    // 消灯
};
```

#### 4.9.7.4. スイッチ種別

```
type InteriorLightSwitchType = enum {
    AngleAdjust,    // 角度調整スイッチ
    FoldSwitch, // 折り畳みスイッチ
    HeatingSwitch,  // ヒーティングスイッチ
    AntiGlareSwitch // 防眩機能スイッチ
};
```

#### 4.9.7.5. 室内ライト種別

```
type InteriorLightType = enum {
    AmbientLight,   // アンビエントライト
    InteractiveLightBar,    // インタラクティブライトバー
    DomeLight,  // ドームライト
    GloveBoxLight   // グローブボックスライト
};
```

#### 室内ライトスイッチ構成情報（getConfigAll が返すデータ型）

```
type InteriorLightSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchType : InteriorLightSwitchType,
    lightType : InteriorLightType,
    capabilities : enumSet(InteriorLightSwitchCapabilityType),
    effect : String,
    mountedOn : RelatedObjectType
};
```

#### 4.9.7.6. 室内ライトスイッチ状態

```
type InteriorLightSwitchStatusType = struct {
    getSwitchIn : SwitchInputType,
    mainStatus : InteriorLightSwitchMainStatusType
};
```

#### 4.9.7.7. 室内ライトスイッチイベント種別

```
type InteriorLightSwitchEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type InteriorLightEventType = struct {
    instanceId : IdType,
    eventInfo : variant(InteriorLightSwitchEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.9.8. サービスコールの詳細規定

## getConfigAll

すべての室内ライトスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | InteriorLightSwitchConfigType[] | 静的な構成値の参照。 |

**【機能】**

すべての室内ライトスイッチに関する構成情報を返す。

## getStatus

スイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | InteriorLightSwitchStatusType | 対象の室内ライトスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象の室内ライトに関するすべての状態を返す。


status には getSwitchIn（点灯/消灯）等を含む。

## getErrorString

エラー情報取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| errorStatus | String | 故障状態を表す文字列（DTC リスト想定） |

**【機能】**

操作対象の室内ライトに関するエラー情報を返す。

本APIはオプション扱い。詳細はOEM依存。

## notify

室内ライトスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象の室内ライトスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (InteriorLightSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象の室内ライトスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

室内ライトスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

室内ライトスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | InteriorLightEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントはFIFOで取り出され、取り出し後はキューから削除される。

# 4.10. Vehicle.Cabin.InteriorMirror（略称：InteriorMirror）

## 4.10.1. 位置付けと機能

Vehicle.Cabin.InteriorMirror（略称：InteriorMirror）は，車室内の防眩ミラーを操作するためのマルチインスタンスオブジェクトである。

### 4.10.1.1. 室内ミラーの位置付けと性質

室内ミラーとは，車両の乗員室内に装着されたミラーで，防眩機能（アンチグレア機能）を備えたものである。防眩機能により，夜間走行時にヘッドライトの光による輻射眩花を軽減することができる。

室内ミラーはルームミラーとして運転席の上部に装着されるのが一般的である。

室内ミラーの物理スイッチは，室内ミラーとは別のオブジェクトとして扱う。

#### 4.10.1.2. InteriorMirrorの機能

InteriorMirrorは，防眩機能により鏡面の反射率を制御するオブジェクトであり，MovableObjectの特性を持つ。

防眩モード時は，防眩レベルを段階的に制御できる。防眩レベルは0〜100のパーセンテージで表現され，0は無反射（暗）から100は全反射（明）となる。

室内ミラーの初期位置，目標位置となる端点は固定であるため，室内ミラーの目標位置はAPIで指定，参照しない設計とする。

室内ミラーが故障を検知した場合，制御を受け付けない。

#### 4.10.1.3. 室内ミラーに対する制御の調停

室内ミラーに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，室内ミラーの制御サービスコールが呼び出された室内ミラーが動作中に，別の制御サービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，新しい制御が開始される。

また，室内ミラーは標準的なロック機能を持つLockableObjectである。室内ミラーのロック機能はソフトウェアで実現される。

室内ミラーが故障を検知した場合，ロックは解除され，故障中は，ロックを取得することができない。

### 4.10.2. 機能クラス

InteriorMirrorは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーミラーであり，APIにより防眩レベルを制御できる |

### 4.10.3. リスククラス

InteriorMirrorは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskDimmingOnTrip | 走行中に防眩機能が動作し，走行中に視界を妨げるリスク |

### 4.10.4. 構成情報

InteriorMirrorの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | 車両内での室内ミラーの位置 |
| capabilities | enumSet (InteriorMirrorCapabilityType) | その室内ミラーが持つ機能 |

| | | |
|---|---|---|
| riskClasses | enumSet (InteriorMirrorRiskType) | その室内ミラーのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（フロントウィンドウなど） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.10.5. 状態

InteriorMirrorの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | InteriorMirrorMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| dimmingLevel | UInt32 | 現在の防眩レベル（0%：無反射，100%：全反射） |

室内ミラーの主状態（mainStatus）は，各操作状態（停止中，防眩中）と故障状態をとる。

- **停止中（Stopped）**：デフォルト状態，またはstartMoveによって設定された目標防眩度に到達して停止している状態。この状態では室内ミラーは動作していない。
- **防眩中（Dimming）**：startMoveによって設定された目標防眩度に向かって防眩レベルを調整している状態。この状態では，指定された目標防眩レベルへの制御が進行中である。
- **故障中（Fault）**：室内ミラーが故障を検知した状態。この状態では制御を受け付けず，ロック機能も取得できない。

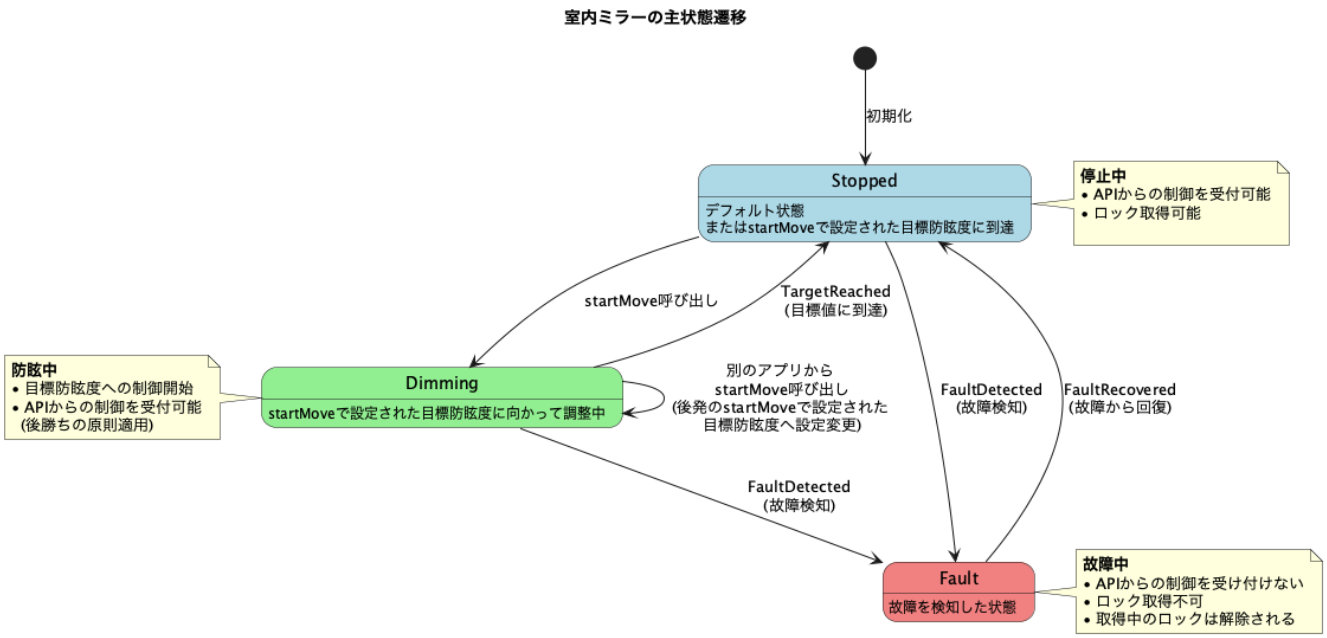



*Figure 10.* 室内ミラーの主状態の状態遷移

## 4.10.6. サービスコール一覧

InteriorMirrorに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべての室内ミラーの構成情報の参照 |
| getStatus | 室内ミラーの状態の参照 |
| startMove | 室内ミラーの防眩レベルの設定 |
| stopMove | 室内ミラーの防眩動作の停止 |
| lock | 室内ミラーのロックの取得 |
| unlock | 室内ミラーのロックの解除 |
| notify | 室内ミラーイベントの通知要求 |
| unnotify | 室内ミラーイベントの通知停止 |
| getEvent | 室内ミラーイベントの取得 |

## 4.10.7. イベント

室内ミラーに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，故障検知がある。

InteriorMirrorに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された防眩レベルに到達して停止，または，動作停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |

| | |
|---|---|
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.10.8. データ型の定義

### 4.10.8.1. 室内ミラーの機能クラス

```
type InteriorMirrorCapabilityType = enum {
    Movable
};
```

### 4.10.8.2. 室内ミラーに対するリスククラス

```
type InteriorMirrorRiskType = enum {
    RiskDimmingOnTrip
};
```

### 4.10.8.3. 室内ミラーの構成情報（getConfigAllが返すデータ型）

```
type InteriorMirrorConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(InteriorMirrorCapabilityType),
    riskClasses : enumSet(InteriorMirrorRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

### 4.10.8.4. 室内ミラー状態

```
type InteriorMirrorMainStatusType = enum {
    Stopped,    // 停止中。デフォルト状態またはstartMoveによって設定された目標防眩度に
```

```
到達した停止状態。
この状態では室内ミラーは動作していない。
    Dimming,    // 防眩中。startMoveによって設定された目標防眩度へ防眩レベルを調整して
いる状態。
指定された目標防眩レベルへの制御が進行中である。
    Fault    // 故障中。室内ミラーが故障を検知した状態。
この状態では制御を受け付けず，ロック機能も取得できない。
};
```

```
type InteriorMirrorStatusType = struct {
    mainStatus : InteriorMirrorMainStatusType,
    lockStatus : LockStatusType,
    dimmingLevel : UInt32
};
```

室内ミラーの状態情報

### 4.10.8.5. 室内ミラーイベント

```
type InteriorMirrorEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type InteriorMirrorEventType = struct {
    instanceId : IdType,
    eventInfo : variant(InteriorMirrorEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.10.9. サービスコールの詳細規定

## getConfigAll

すべての室内ミラーの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | InteriorMirrorConfigType[] | すべての室内ミラーに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべての室内ミラーに関する構成情報を返す。

## getStatus

室内ミラーの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | InteriorMirrorStatusType | 対象室内ミラーの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象の室内ミラーに関するすべての状態を返す。

## startMove

室内ミラーの防眩レベルの設定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラーのID |
| dimmingLevel | UInt32 | 防眩レベル（0〜100） |

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | dimmingLevel，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象の室内ミラーが故障中 |
| E_OBJECT_LOCKED | 操作対象の室内ミラーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

防眩レベルをdimmingLevelに設定し，操作対象の室内ミラーの防眩動作を開始する。

dimmingLevelに0を指定すれば無反射，100を指定すれば全反射の意味になる。

リスククラスRiskDimmingOnTripに対応できていない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

室内ミラーの防眩動作の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象の室内ミラーが故障中 |

| | |
|---|---|
| E_OBJECT_LOCKED | 操作対象の室内ミラーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

操作対象の室内ミラーの防眩動作の停止を要求する。

## lock

室内ミラーのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象の室内ミラーが，priorityと同じかそれより高い優先度でロックされている |
| E_OBJECT_FAULT | ロック対象の室内ミラーが故障中 |

**【機能】**

ロック対象の室内ミラーを，指定した操作優先度でロックする。

## unlock

室内ミラーのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象の室内ミラーをロックしていない |

**【機能】**

ロック解除対象の室内ミラーを，ロック解除する。

## notify

室内ミラーイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象の室内ミラーのID |
| eventFilter | enumSet (InteriorMirrorEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象の室内ミラーで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

室内ミラーイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

室内ミラーイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | InteriorMirrorEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.11. Vehicle.Cabin.InteriorMirrorSwitch（略称：InteriorMirrorSwitch）

## 4.11.1. 位置付けと機能

室内ミラーの防眩度を変更する物理的なスイッチを扱う。スイッチとミラーは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

#### 4.11.1.1. 室内ミラー防眩度変更スイッチの位置付けと性質

室内ミラー防眩度変更スイッチとは、車両内に設置された、乗員が防眩度を物理操作できる入力デバイスである。

スイッチとミラーは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは物理的な操作子を持つが、防眩機能の制御は室内ミラーオブジェクトが管理する。スイッチは入力源として機能し、ミラー制御主体にはならない。

スイッチの操作方式は2Way（ON/OFF）または3Way（ON/OFF/LEVEL）であり、どの方式に対応するかを構成情報として持つ。

#### 4.11.1.2. 室内ミラー防眩度変更スイッチの機能

室内ミラー防眩度変更スイッチは乗員の操作による防眩度変更入力（ON/OFF/LEVEL）をOSへ提供し、室内ミラーオブジェクトへ制御要求を伝達する役割を持つ。

スイッチの状態はmainStatusとして参照でき、現在の入力状態（OFF/ON/LEVEL）の監視が可能である。

capabilitiesで対応する機能範囲を示す。

#### 4.11.1.3. 室内ミラー防眩度変更スイッチに対する制御の調停

室内ミラー防眩度変更スイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、室内ミラーオブジェクトへルーティングされる。

調停は室内ミラー側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、ミラー側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

### 4.11.2. 機能クラス

InteriorMirrorSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | スイッチ入力による防眩度変更機能 |

### 4.11.3. 構成情報

InteriorMirrorSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchType | InteriorMirrorSwitchType | スイッチ操作方式（2Way/3Way） |
| capabilities | enumSet (InteriorMirrorSwitchCapabilityType) | このスイッチが扱う防眩機能の集合 |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

### 4.11.4. 状態

InteriorMirrorSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | InteriorMirrorSwitchMainStatusType | スイッチの入力状態（OFF/ON/LEVEL） |

## 4.11.5. サービスコール一覧

InteriorMirrorSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべての室内ミラー防眩度変更スイッチの構成情報の参照 |
| getStatus | 室内ミラー防眩度変更スイッチの状態参照 |
| notify | 室内ミラー防眩度変更スイッチイベントの通知要求 |
| unnotify | 室内ミラー防眩度変更スイッチイベントの通知停止 |
| getEvent | 室内ミラー防眩度変更スイッチイベントの取得 |

## 4.11.6. イベント

室内ミラー防眩度変更スイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

InteriorMirrorSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| SwitchStateChanged | スイッチ入力状態が変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がSwitchStateChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | InteriorMirrorSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.11.7. データ型の定義

### 4.11.7.1. 室内ミラー防眩度変更スイッチの機能クラス

```
type InteriorMirrorSwitchCapabilityType = enum {
    SwitchControl
};
```

#### 4.11.7.2. スイッチ操作方式

```
type InteriorMirrorSwitchType = enum {
    TwoWay, // 2Way操作（OFF/ON）
    ThreeWay    // 3Way操作（OFF/ON/LEVEL中間位置）
};
```

#### 4.11.7.3. スイッチ入力状態

```
type InteriorMirrorSwitchMainStatusType = enum {
    False,  // OFF - 防眩度変更停止
    True,   // ON - 防眩度を最大（全反射）に変更
    Level   // LEVEL - 中間位置（3Way操作時）で防眩度を段階的に調整
};
```

### 室内ミラー防眩度変更スイッチ構成情報（getConfigAll が返すデータ型）

```
type InteriorMirrorSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchType : InteriorMirrorSwitchType,
    capabilities : enumSet(InteriorMirrorSwitchCapabilityType),
    displayName : String
};
```

#### 4.11.7.4. 室内ミラー防眩度変更スイッチ状態

```
type InteriorMirrorSwitchStatusType = struct {
    mainStatus : InteriorMirrorSwitchMainStatusType
};
```

室内ミラー防眩度変更スイッチの状態情報

#### 4.11.7.5. 室内ミラー防眩度変更スイッチイベント種別

```
type InteriorMirrorSwitchEventKindType = enum {
    SwitchStateChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type InteriorMirrorSwitchEventType = struct {
    instanceId : IdType,
    eventInfo : variant(InteriorMirrorSwitchEventKindType) {
        SwitchStateChanged {
            newState : InteriorMirrorSwitchMainStatusType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.11.8. サービスコールの詳細規定

### getConfigAll

すべての室内ミラー防眩度変更スイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | InteriorMirrorSwitchConfigType[] | すべての室内ミラー防眩度変更スイッチに関する構成情報 |

**【機能】**

すべての室内ミラー防眩度変更スイッチに関する構成情報を返す。

### getStatus

室内ミラー防眩度変更スイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ミラー防眩度変更スイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | InteriorMirrorSwitchStatusType | 対象の室内ミラー防眩度変更スイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象の室内ミラー防眩度変更スイッチに関する状態を返す。

status には mainStatus（スイッチ入力状態）を含む。

## notify

室内ミラー防眩度変更スイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象の室内ミラー防眩度変更スイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (InteriorMirrorSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象の室内ミラー防眩度変更スイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

室内ミラー防眩度変更スイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

室内ミラー防眩度変更スイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | InteriorMirrorSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.12. Vehicle.Cabin.InteriorLight（略称：InteriorLight）

## 4.12.1. 位置付けと機能

Vehicle.Cabin.InteriorLight（略称:　InteriorLight）は、車両の車室内に取り付けられた照明装置を扱うマルチインスタンスオブジェクトである。乗員の視認性向上のため、点灯・消灯、輝度、色、エフェクト（効果）などを制御・参照する。

### 4.12.1.1. 室内ライトの位置付けと性質

室内ライトとは、車両の車室内に取り付けられた照明装置であり、乗員が車内の状況を確認しやすくすることで視認性の向上を図るものである。

複数の室内ライトを個別に扱うことを前提としたマルチインスタンスオブジェクトである。

対象となるライトには、アンビエントライト、インタラクティブライトバー、ドームライト、グローブボックスライトなどが含まれる。

これらは特に乗降時や夜間走行時に重要な役割を果たし、車内環境の安全性および快適性の向上に寄与する。

また、必要に応じて足元照明やドアカーテシランプといったオプションの室内ライトを追加することも可能である。

#### 4.12.1.2. 室内ライトの機能

室内ライトは、車両内部に設置された照明装置の点灯および消灯を制御する機能を有する。

これに加え、輝度をパーセンテージで設定する機能、RGBカラーコードによる色彩設定機能、さらに文字列によって指定されるライトエフェクトの設定機能を備える。

エフェクト機能の内容はOEM依存であり、主にインタラクティブライトバーなどの特定のライト種別で有効となる。

また、環境光量を参照可能なライトも存在し、周囲の明るさを考慮した制御が可能である。

これらの機能は、輝度制御、色制御、エフェクト制御、環境光参照といった機能クラスとして整理されている。

さらに、現在の輝度、色、エフェクト、環境光量といった状態情報の参照や、構成情報の取得、各種設定変更、ならびに制御状態や故障検知、効果開始・終了などのイベント通知を行う機能を備えている。

#### 4.12.1.3. 室内ライトに対する制御の調停

室内ライトに対する制御の調停には、標準的な調停機能（後勝ちの原則）が適用され、後から発行された制御要求が優先される性質を持つ。

すなわち、室内ライトを点灯するサービスコールが呼び出され、室内ライト点灯中に、室内ライトを消灯するサービスコールが呼び出されると、最初のサービスコールによる制御はキャンセルされ、室内ライトは消灯する。

### 4.12.2. 機能クラス

InteriorLightは，以下の機能クラスを持つ。

| | |
|---|---|
| Dimmable | 輝度（Intensity）を制御できる |
| ColorControllable | 色（Color）を制御できる（#000000〜#FFFFFF） |
| EffectControllable | エフェクト（effect）文字列による効果設定ができる（OEM依存） |
| AmbientAware | 環境光量（ambientLightLevel）を参照できる |

### 4.12.3. リスククラス

InteriorLightは，以下のリスククラスを持つ。

| | |
|---|---|
| LightTempOver | ライト温度上限超過 : 温度上昇に伴う発火リスクとする |
| LightBrightnessOver | 高輝度異常 : 運転中の視界不良に繋がるリスク |
| LightBrightnessUnder | 低輝度異常 : 運転中の視界不良に繋がるリスク |
| LightColorIllegal | 色味異常 : 法規に繋がるリスク |
| LightBlinkIllegal | 点滅異常 : 運転中の視界不良に繋がるリスク |

| | |
|---|---|
| LightBatteryLongOn | 長時間点灯によるバッテリ上がり：バッテリ上がりはライト以外でも該当するので、共通化するのがよいかも |
| LightPrivacyVisible | 室内が視認されやすい：プライバシー観点ではあり |

### 4.12.4. 構成情報

InteriorLightの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| lightType | InteriorLightType | ライト種別（AmbientLight, InteractiveLightBar, DomeLight, GloveBoxLight など） |
| capabilities | enumSet (InteriorLightCapabilityType) | そのライトが持つ機能 |
| riskClasses | enumSet (InteriorLightRiskType) | そのライトのリスククラスの集合 |
| effect | String | ライトエフェクト種別（OEMによるため標準は規定しない） |
| mountedOn | RelatedObjectType | 関連するオブジェクト（設置対象） |

### 4.12.5. 状態

InteriorLightの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| intensity | IntensityType | 輝度（%）。1%=最大減衰、100%=減衰なし |
| color | ColorType | 色彩（#000000～#FFFFFF） |
| effect | String | エフェクト文字列（OEM依存） |
| ambientLightLevel | PercentageType | 環境光量（%）。0%=環境光なし、100%=最大の明るさ |
| eventEnable | Bool | イベント通知の許可/不許可参照 |

### 4.12.6. サービスコール一覧

InteriorLightに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべての室内ライトの構成情報の参照 |
| getStatus | 室内ライトの状態の参照 |
| getErrorString | エラー情報取得 |
| lightOn | 点灯 |
| lightOff | 消灯 |

| | |
|---|---|
| setIntensity | 輝度設定 |
| setColor | 色彩設定 |
| setEffect | 室内ライトエフェクト設定 |
| notify | 室内ライトイベントの通知要求 |
| unnotify | 室内ライトイベントの通知停止 |
| getEvent | 室内ライトイベントの取得 |

## 4.12.7. イベント

室内ライトに関するイベントとして、制御主体やターゲット到達、効果開始/終了、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。InteractiveLightBar　などの一部機能はエフェクト設定時のみ有効。

InteriorLightに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された設定値（輝度、色味）に到達 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EffectStart | ライトエフェクト開始 |
| EffectEnd | ライトエフェクト終了 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.12.8. データ型の定義

### 4.12.8.1. 室内ライトの機能クラス

```
type InteriorLightCapabilityType = enum {
    Dimmable,
    ColorControllable,
    EffectControllable,
```

```
    AmbientAware
};
```

#### 4.12.8.2. 室内ライトに対するリスククラス

```
type InteriorLightRiskType = enum {
    LightTempOver,
    LightBrightnessOver,
    LightBrightnessUnder,
    LightColorIllegal,
    LightBlinkIllegal,
    LightBatteryLongOn,
    LightPrivacyVisible
};
```

#### 室内ライトの構成情報（getConfigAll が返すデータ型）

```
type InteriorLightConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    lightType : InteriorLightType,
    capabilities : enumSet(InteriorLightCapabilityType),
    riskClasses : enumSet(InteriorLightRiskType),
    effect : String,
    mountedOn : RelatedObjectType
};
```

#### 4.12.8.3. 室内ライト状態

```
type InteriorLightStatusType = struct {
    intensity : IntensityType,
    color : ColorType,
    effect : String,
    ambientLightLevel : PercentageType,
    eventEnable : Bool
};
```

#### 4.12.8.4. 室内ライト種別

```
type InteriorLightType = enum {
    AmbientLight,   // アンビエントライト
    InteractiveLightBar,    // インタラクティブライトバー
    DomeLight,  // ドームライト
    GloveBoxLight   // グローブボックスライト
```

```
};
```

#### 4.12.8.5. 室内ライトイベント種別

```
type InteriorLightEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    EffectStart,
    EffectEnd,
    EventOverflow,
    ConfigChanged
};
```

```
type InteriorLightEventType = struct {
    instanceId : IdType,
    eventInfo : variant(InteriorLightEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.12.9. サービスコールの詳細規定

#### getConfigAll

すべての室内ライトの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | InteriorLightConfigType[] | すべての室内ライトに関する構成情報 |

**【エラーコード】**

| | |
|---|---|
| E_OK | 成功 |

| | |
|---|---|
| E_NOT_SUPPORTED | 未サポート（実装により返却されない場合あり） |

**【機能】**

すべての室内ライトに関する構成情報を返す。

## getStatus

室内ライトの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | InteriorLightStatusType | 対象の室内ライトの状態（Intensity, Color, effect, ambientLightLevel など） |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象の室内ライトに関するすべての状態を返す。

## getErrorString

エラー情報取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| errorStatus | String | 故障状態を表す文字列（DTC リスト想定） |

**【機能】**

"操作対象の室内ライトに関するエラー情報を返す。"


"本APIはオプション扱い。詳細はOEM依存。"

## lightOn

点灯

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | E_OK またはエラー |

**【機能】**

設定された輝度・色で点灯させる。

## lightOff

消灯

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | E_OK またはエラー |

**【機能】**

消灯させる。

## setIntensity

輝度設定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |
| intensity | IntensityType | 設定する輝度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | E_OK またはエラー |

**【機能】**

輝度（%）を設定する。

## setColor

色彩設定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |

| | | |
|---|---|---|
| color | ColorType | 設定する色彩 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | E_OK またはエラー |

**【機能】**

色彩（#000000〜#FFFFFF）を設定する。

## setEffect

室内ライトエフェクト設定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象の室内ライトのID |
| effect | String | 設定するライトエフェクト |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | E_OK またはエラー |

**【機能】**

ライトエフェクトを文字列で設定。InteractiveLightBar のみ有効。

## notify

室内ライトイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象の室内ライトID（省略時は全インスタンス） |
| eventFilter | enumSet (InteriorLightEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象の室内ライトで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

室内ライトイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

室内ライトイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | InteriorLightEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

NOTE

イベントはFIFOで取り出され、取り出し後はキューから削除される。

# 4.13. Vehicle.Cabin.MirrorSwitch（略称：MirrorSwitch）

## 4.13.1. 位置付けと機能

Mirrorを制御する物理的なスイッチを扱う。スイッチとMirrorは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.13.1.1. Mirrorスイッチの位置付けと性質

Mirrorスイッチとは、運転席に配置され乗員が物理操作できるMirror制御デバイスである。

Mirrorスイッチは各Mirrorに配置され独立した系統を持つ。

スイッチとMirrorは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、Mirror制御の主体にはならない。

### 4.13.1.2. Mirrorスイッチの機能

Mirrorスイッチは乗員の操作によるスイッチ入力をOSへ提供し、Mirrorオブジェクトへ制御要求を伝達する役割を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

### 4.13.1.3. Mirrorスイッチに対する制御の調停

Mirrorスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、Mirrorオブジェクトへルーティングされる。

調停はMirror側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、Mirror側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

## 4.13.2. 機能クラス

MirrorSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | Mirror操作入力を提供する基本機能 |

## 4.13.3. 構成情報

MirrorSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | MirrorSwitchCategoryType | スイッチ系統（ミラー調整スイッチ、ミラー折り畳みスイッチ、ミラーヒーティングスイッチ、ミラー防眩スイッチ） |
| capabilities | enumSet (MirrorSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（ミラー調整スイッチ、ミラー折り畳みスイッチ、ミラーヒーティングスイッチ、ミラー防眩スイッチかなど、OEM依存） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.13.4. 状態

MirrorSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | MirrorSwitchMainStatusType | スイッチの入力状態（MirrorAdjust系統の場合：OPEN/CLOSE/LEFT/RIGHT） |
| mirrorFoldingStatus | MirrorSwitchMirrorFoldingStatusType | ミラー格納スイッチの入力状態。（ON、OFF） |
| mirrorHeatingStatus | MirrorSwitchMirrorHeatingStatusType | ミラーヒーティングスイッチの入力状態。（ON、OFF） |
| heaterLevel | UInt32 | ヒーティングON場合のレベル。0〜100。ダイアルで調整可能。 |
| mirrorAntiglareStatus | MirrorSwitchMirrorAntiglareStatusType | ミラー防眩スイッチの入力状態。（ON、OFF） |
| antiglareLevel | UInt32 | 防眩がONの場合ののレベル。0〜100。ダイアルで調整可能。 |

## 4.13.5. サービスコール一覧

MirrorSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのMirrorスイッチの構成情報の参照 |
| getStatus | Mirrorスイッチの状態参照 |
| notify | Mirrorスイッチイベントの通知要求 |
| unnotify | Mirrorスイッチイベントの通知停止 |

| | |
|---|---|
| getEvent | Mirrorスイッチイベントの取得 |

### 4.13.6. イベント

Mirrorスイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

MirrorSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | ミラー調整に対応するスイッチのmainStatus入力状態が変化 |
| FoldingStatusChanged | ミラー格納に対応するスイッチの入力状態が変化 |
| HeatingStatusChanged | ミラーヒーティングに対応するスイッチの入力状態が変化 |
| HeaterLevelChanged | ヒーターモードON時のヒーターレベルが変化 |
| AntiglareStatusChanged | ミラー防眩に対応するスイッチの入力状態が変化 |
| AntiglareLevelChanged | 防眩モードON時の防眩レベルが変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | MirrorSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がFoldingStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | MirrorSwitchMirrorFoldingStatusType | Foldingスイッチの新しい入力状態 |

また，イベント種別がHeatingStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | MirrorSwitchMirrorHeatingStatusType | Heatingスイッチの新しい入力状態 |

また，イベント種別がHeaterLevelChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newHeaterLevel | UInt32 | 新しいレベル |

また，イベント種別がAntiglareStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | MirrorSwitchMirrorAntiglareStatusType | Heatingスイッチの新しい入力状態 |

また，イベント種別がAntiglareLevelChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newAntiglareLevel | UInt32 | 新しいレベル |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.13.7. データ型の定義

### 4.13.7.1. Mirrorスイッチの機能クラス

```
type MirrorSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.13.7.2. スイッチ系統

```
type MirrorSwitchCategoryType = enum {
    MirrorAdjust,   // MirrorAdjustスイッチ系統
    MirrorFolding,  // MirrorFoldingスイッチ系統
    MirrorHeating,  // MirrorHeatingスイッチ系統
    MirrorAntiglare // MirrorAntiglareスイッチ系統
};
```

### 4.13.7.3. Mirrorsスイッチの入力状態

```
type MirrorSwitchMainStatusType = enum {
    False,  // OFF - Mirror停止
    Up, // UP - MirrorUP
    Down,   // Down - MirrorDown
    Right,  // Right - MirrorRight
    Left    // Left - MirrorLeft
};
```

### 4.13.7.4. Mirror格納スイッチの入力状態

```
type MirrorSwitchMirrorFoldingStatusType = enum {
    False,  // OFF - ミラー展開
    True    // ON - ミラー格納
};
```

#### 4.13.7.5. Mirrorヒーティングの入力状態

```
type MirrorSwitchMirrorHeatingStatusType = enum {
    False,  // OFF - 機能停止
    True    // ON - 機能有効
};
```

#### 4.13.7.6. Mirror防眩スイッチの入力状態

```
type MirrorSwitchMirrorAntiglareStatusType = enum {
    False,  // OFF - 機能停止
    True    // ON - 機能有効
};
```

### Mirrorスイッチ構成情報（getConfigAll が返すデータ型）

```
type MirrorSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : MirrorSwitchCategoryType,
    capabilities : enumSet(MirrorSwitchCapabilityType),
    switchType : String,
    displayName : String
};
```

#### 4.13.7.7. Mirrorスイッチ状態

```
type MirrorSwitchStatusType = struct {
    mainStatus : MirrorSwitchMainStatusType,
    mirrorFoldingStatus : MirrorSwitchMirrorFoldingStatusType,
    mirrorHeatingStatus : MirrorSwitchMirrorHeatingStatusType,
    heaterLevel : UInt32,
    mirrorAntiglareStatus : MirrorSwitchMirrorAntiglareStatusType,
    antiglareLevel : UInt32
};
```

Mirrorスイッチの状態情報

#### 4.13.7.8. Mirrorスイッチイベント種別

```
type MirrorSwitchEventKindType = enum {
    MainStatusChanged,
    FoldingStatusChanged,
    HeatingStatusChanged,
```

```
    HeaterLevelChanged,
    AntiglareStatusChanged,
    AntiglareLevelChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type MirrorSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : MirrorSwitchCategoryType,
    eventInfo : variant(MirrorSwitchEventKindType) {
        MainStatusChanged {
            newState : MirrorSwitchMainStatusType
        },
        FoldingStatusChanged {
            newState : MirrorSwitchMirrorFoldingStatusType
        },
        HeatingStatusChanged {
            newState : MirrorSwitchMirrorHeatingStatusType
        },
        HeaterLevelChanged {
            newHeaterLevel : UInt32
        },
        AntiglareStatusChanged {
            newState : MirrorSwitchMirrorAntiglareStatusType
        },
        AntiglareLevelChanged {
            newAntiglareLevel : UInt32
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.13.8. サービスコールの詳細規定

### getConfigAll

すべてのMirrorスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | MirrorSwitchConfigType[] | すべてのMirrorスイッチに関する構成情報 |

**【機能】**

すべてのMirrorスイッチに関する構成情報を返す。

## getStatus

Mirrorスイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のMirrorスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | MirrorSwitchStatusType | 対象のMirrorスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のMirrorスイッチに関する状態を返す。


status には mainStatus（各Mirror系統のスイッチ入力状態）を含む。

## notify

Mirrorスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のMirrorスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (MirrorSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のMirrorスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

Mirrorスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

Mirrorスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | MirrorSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

【機能】

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.14. Vehicle.Cabin.Seat（略称：Seat）

## 4.14.1. 位置付けと機能

Vehicle.Cabin.Seat（略称：Seat）は，車両のシートを操作するためのマルチインスタンスオブジェクトである。

### 4.14.1.1. シートの位置付けと性質

シートとは，人が座るための場所、複数の部位に分かれていると定義する。シートを開閉するための物理スイッチは，シートとは別のオブジェクトとして扱う。

#### Seatの機能

Seatは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉，100：全開）で表す。また，0に近づく動作を閉動作，100に近づく動作を開動作と呼ぶ。

挟み込み防止機能を持つシートは，可動部の閉動作中に挟み込みを検知すると，閉動作を停止し，開動作を開始する。挟み込み検知後にどの位置まで開くかは，車両依存である。

可動部が停止中の間は，現在位置が変化することはない。可動部が強制的に移動されたなどの理由で現在位置が変化した場合には，シートが一時的に故障し，すぐに回復したものと扱う。

### 4.14.1.2. シートに対する制御の調停

シートに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，シートを閉めるサービスコールが呼び出され，シートが閉じる動作中に，開くサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，シートは開く動作を始める。

## 4.14.2. 機能クラス

Seatは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーシートであり，APIにより移動させることができる |

## 4.14.3. リスククラス

Seatは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | シートを作動させることに伴う挟み込みのリスク |
| RiskDriveMove | 運転中にシートを作動させることに伴うリスク |

### 4.14.4. 構成情報

Seatの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| capabilities | enumSet (SeatCapabilityType) | そのシートが持つ機能 |
| riskClasses | enumSet(SeatRiskType) | そのシートのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

### 4.14.5. 状態

Seatの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | SeatMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| isOccupied | boolean | 着座状況 True:着座　False:未着座 |
| isBelted | boolean | シートベルト状況　True：装着　Flase：未装着 |
| seatBeltHeight | SeatBeltHeightType | シートベルト高さ |
| massageLevel | MassageLevelType | マッサージ機能強さ　単位：% |
| heatingCooling | HeatingCoolingType | シート温度 |
| seatPosition | structOf (Vehicle.Cabin.Seat.SeatpositionType) | シートの現在位置 |
| targetPosition | structOf (Vehicle.Cabin.Seat.SeatpositionType) | 目標シート位置 |

シートの主状態（mainStatus）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。

```
[image path=./images/image/image1.png, width=50%, title="図4-Y-1. シートの主状態の状態遷移"]
```

シートの現在位置（seatPosition）と目標位置（targetPosition），MovableObjectの現在位置と目標位置に関する規定に従う。

## 4.14.6. サービスコール一覧

Seatに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのシートの構成情報の参照 |
| getStatus | シートの状態の参照 |
| changeSeatBeltHeight | シートベルト高さ変更 |
| changeMassageLevel | マッサージ機能強さ変更 |
| changeHeatingCooling | シート温度変更 |
| changeTargetPosition | シート位置変更 |
| stopMove | シートの開閉の停止 |
| notify | シートイベントの通知要求 |
| unnotify | シートイベントの通知停止 |
| getEvent | シートイベントの取得 |

## 4.14.7. イベント

シートに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Seatに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplicatio n | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.14.8. データ型の定義

### 4.14.8.1. シートの機能クラス

```
type SeatCapabilityType = enum {
    Movable
};
```

### 4.14.8.2. シートに対するリスククラス

```
type SeatRiskType = enum {
    RiskPinch,
    RiskDriveMove
};
```

### 4.14.8.3. シートの構成情報（getConfigAllが返すデータ型）

```
type SeatConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(SeatCapabilityType),
    riskClasses : enumSet(SeatRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

### 4.14.8.4. シート状態

```
type SeatMainStatusType = enum {
    Stopped,    // 停止中
    Operating,  // シート動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type SeatStatusType = struct {
```

```
    mainStatus : SeatMainStatusType,
    lockStatus : LockStatusType,
    isOccupied : boolean,
    isBelted : boolean,
    seatBeltHeight : SeatBeltHeightType,
    massageLevel : MassageLevelType,
    heatingCooling : HeatingCoolingType,
    seatPosition : structOf(Vehicle.Cabin.Seat.SeatpositionType),
    targetPosition : structOf(Vehicle.Cabin.Seat.SeatpositionType)
};
```

#### 4.14.8.5. シートイベント

```
type SeatEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type SeatEventType = struct {
    instanceId : IdType,
    eventInfo : variant(SeatEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

#### 4.14.8.6. シート位置

```
type SeatpositionType = struct {
    posiotion : UInt32, // 前後位置[mm] 0=もっとも前
    height : UInt32,    // シート高さ[mm] 0=最も低い
    tile : UInt32,  // シート角度[°] 0=シートが最も倒れていない
    recline : UInt32,   // 背もたれ角度[°] 0=背もたれが最も倒れていない
    lumbarSupport : UInt32, //   ランバーサポートの左右（外内）[%] 0:最も内側 100:もっ
```

```
とも外側
    lumbarHeight : UInt32,  // ランバーサポートの高さ[%] 0=最も低い
    sideBolesterSupport : UInt32,   // サイドボルスターのサポート度合い[%]  0:最もサポートが弱い（外側に開く）100:最もサポートが強い（狭い）
    seatingLength : UInt32, // 座面の前後位置[mm] 0=最も後ろ位置
    headrestHeight : UInt32 // ヘッドレストの高さ[mm] 0=最も低い
};
```

## 4.14.9. サービスコールの詳細規定

### getConfigAll

すべてのシートの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | SeatConfigType[] | すべてのシートに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのシートに関する構成情報を返す。

### getStatus

シートの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | SeatStatusType | 対象シートの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のシートに関するすべての状態を返す。

## changeSeatBeltHeight

シートベルト高さ変更

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |
| targetSeatBeltHeight | SeatBeltHeight | 目標シートベルト高さ |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetSeatBeltHeight，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシートが故障中 |
| E_OBJECT_STATUS | 操作対象のシートが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシートが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標シートベルト高さをtype:　SeatBeltHeightに設定し，操作対象のシートのシートベルト高さ調整を開始する。

SeatBeltHeightの単位はmmであり、0は車両として最も低い位置を示す。

## changeMassageLevel

マッサージ機能強さ変更

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |
| targetMassageLevel | massageLevel | 目標マッサージ強度 |

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetMassageLevel，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシートが故障中 |
| E_OBJECT_STATUS | 操作対象のシートが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシートが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標シートベルト高さをtype: type: massageLevelに設定し，操作対象のシートのマッサージ機能強さ調整を行う。

massageLevelの単位は%であり、0はOFF、100は最も強いマッサージレベルを示す。

## changeHeatingCooling

シート温度変更

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |
| targetMassageLevel | massageLevel | 目標マッサージ強度 |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetMassageLevel，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |

| | |
|---|---|
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシートが故障中 |
| E_OBJECT_STATUS | 操作対象のシートが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシートが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標シートベルト高さをtype: type: massageLevelに設定し，操作対象のシートのマッサージ機能強さ調整を行う。

massageLevelの単位は%であり、0はOFF、100は最も強いマッサージレベルを示す。

## changeTargetPosition

シート位置変更

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |
| targetPosition | structOf (Vehicle.Cabin.Seat.SeatpositionType) | 目標シート位置 構造体 |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシートが故障中 |
| E_OBJECT_STATUS | 操作対象のシートが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシートが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標シートベルト高さを type: targetPositionに設定し，操作対象のシートの位置調整を行う。

Seatpotisionと同じ要素をメンバーに持つ構造体を返す。

## stopMove

シートの開閉の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシートのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシートが故障中 |
| E_OBJECT_STATUS | 操作対象のシートが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシートが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のシートの各次元方向への開閉動作の停止を要求する。

## notify

シートイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のシートのID |
| eventFilter | enumSet (SeatEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のシートで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

シートイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

シートイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | SeatEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意

する必要がない。

# 4.15. Vehicle.Cabin.SeatSwitch（略称：SeatSwitch）

## 4.15.1. 位置付けと機能

Vehicle.Cabin.SeatSwitch（略称：SeatSwitch）シートを動作させる物理的なスイッチを扱う。すべてON/OFFの2-Wayスイッチとする。

## 4.15.2. 構成情報

SeatSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |

## 4.15.3. 状態

SeatSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | SeatSwMainStatusType | スイッチの主状態（正常/停止/故障/電気故障など） |

## 4.15.4. サービスコール一覧

SeatSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | シート動作スイッチの構成情報の参照 |
| getStatus | シート動作スイッチの状態の参照 |

## 4.15.5. サービスコールの詳細規定

### getConfigAll

シート動作スイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | SeatSwConfigType[] | シート動作スイッチに関する構成情報 |

**【エラーコード】**

| | |
|---|---|
| E_OK | 成功 |

| | |
|---|---|
| E_NOT_SUPPORTED | 未サポート（実装により返却されない場合あり） |

**【機能】**

シート動作スイッチに関する構成情報を返す。

・識別名

・IdType（API呼び出しに使う）

・位置

・紐づいている機能

・サポートする機能（オプション）

すべてON/OFFのスイッチを仮定するため

スイッチとしてサポートする機能はオプション扱いとする。

### getStatus

シート動作スイッチの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | シート動作スイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | SeatSwStatusType | シート動作スイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

シート動作スイッチに関するすべての状態を返す。

# 4.16. Vehicle.Cabin.Shade（略称：Shade）

## 4.16.1. 位置付けと機能

Vehicle.Cabin.Shade（略称：Shade）は，車両のシェイドを操作するためのマルチインスタンスオブジェクトである。

### 4.16.1.1. シェイドの位置付けと性質

シェイドとは，開閉できる車室の開口部に付属するもので，車両の走行中も開けておいて差し支えないものと定義する。シェイドを開閉するための物理スイッチは，シェイドとは別のオブジェクトとして扱う。

シェイドは，閉じていると光を通さず，開いていると光を通す。

### Shadeの機能

Shadeは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉，100：全開）で表す。また，0に近づく動作を閉動作，100に近づく動作を開動作と呼ぶ。

挟み込み防止機能を持つシェイドは，可動部の閉動作中に挟み込みを検知すると，閉動作を停止し，開動作を開始する。挟み込み検知後にどの位置まで開くかは，車両依存である。

可動部が停止中の間は，現在位置が変化することはない。可動部が強制的に移動されたなどの理由で現在位置が変化した場合には，シェイドが一時的に故障し，すぐに回復したものと扱う。

#### 4.16.1.2. シェイドに対する制御の調停

シェイドに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，シェイドを閉めるサービスコールが呼び出され，シェイドが閉じる動作中に，開くサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，シェイドは開く動作を始める。

## 4.16.2. 機能クラス

Shadeは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーシェイドであり，APIにより移動させることができる |
| ClosedDetectable | シェイドが全閉状態であることを検出することができる |

## 4.16.3. リスククラス

Shadeは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | シェイドを閉めることに伴う挟み込みのリスク |

## 4.16.4. 構成情報

Shadeの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| capabilities | enumSet (ShadeCapabilityType) | そのシェイドが持つ機能 |
| riskClasses | enumSet(ShadeRiskType) | そのシェイドのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.16.5. 状態

Shadeの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | ShadeMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |

シェイドの主状態（mainStatus）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。また，位置のパーセンテージが大きくなる方へ移動中を開動作中，位置のパーセンテージが小さくなる方へ移動中を閉動作中と呼ぶ（図4-Y-1）。

```
[image path=./images/image/image1.png, width=50%, title="図4-Y-1. シェイドの主状態の状態遷移"]
```

```
シェイドの現在位置（position）と目標位置（targetPosition），MovableObjectの現在位置と目標位置に関する規定に従う。
```

## 4.16.6. サービスコール一覧

Shadeに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのシェイドの構成情報の参照 |
| getStatus | シェイドの状態の参照 |
| startMove | シェイドの開閉の開始 |
| stopMove | シェイドの開閉の停止 |
| notify | シェイドイベントの通知要求 |
| unnotify | シェイドイベントの通知停止 |
| getEvent | シェイドイベントの取得 |

## 4.16.7. イベント

シェイドに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Shadeに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.16.8. データ型の定義

### 4.16.8.1. シェイドの機能クラス

```
type ShadeCapabilityType = enum {
    Movable,
    ClosedDetectable
};
```

### 4.16.8.2. シェイドに対するリスククラス

```
type ShadeRiskType = enum {
    RiskPinch
};
```

### 4.16.8.3. シェイドの構成情報（getConfigAllが返すデータ型）

```
type ShadeConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
```

```
    capabilities : enumSet(ShadeCapabilityType),
    riskClasses : enumSet(ShadeRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.16.8.4. シェイド状態

```
type ShadeMainStatusType = enum {
    Stopped,    // 停止中
    Opening,    // 開動作中
    Closing,    // 閉動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type ShadeStatusType = struct {
    mainStatus : ShadeMainStatusType,
    lockStatus : LockStatusType,
    position : PositionType,
    targetPosition : PositionType
};
```

#### 4.16.8.5. シェイドイベント

```
type ShadeEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type ShadeEventType = struct {
    instanceId : IdType,
    eventInfo : variant(ShadeEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
```

```
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.16.9. サービスコールの詳細規定

### getConfigAll

すべてのシェイドの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | ShadeConfigType[] | すべてのシェイドに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのシェイドに関する構成情報を返す。

### getStatus

シェイドの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシェイドのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | ShadeStatusType | 対象シェイドの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のシェイドに関するすべての状態を返す。

## startMove

シェイドの開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシェイドのID |
| targetPosition | PositionType | 目標位置（開閉度） |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシェイドが故障中 |
| E_OBJECT_STATUS | 操作対象のシェイドが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシェイドが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し，操作対象のシェイドの開閉動作を開始する。

targetPositionに0を指定すれば閉動作の開始，100を指定すれば開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，シェイドを開く方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskOpenに対応できていない場合でも，シェイドを閉じる方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

シェイドの開閉の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のシェイドのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のシェイドが故障中 |
| E_OBJECT_STATUS | 操作対象のシェイドが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のシェイドが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のシェイドの各次元方向への開閉動作の停止を要求する。

## notify

シェイドイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のシェイドのID |
| eventFilter | enumSet (ShadeEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のシェイドで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

### unnotify

シェイドイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

シェイドイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | ShadeEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

## 4.17. Vehicle.Cabin.ShadeSwitch（略称：ShadeSwitch）

## 4.17.1. 位置付けと機能

Shadeを制御する物理的なスイッチを扱う。スイッチとShadeは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.17.1.1. Shadeスイッチの位置付けと性質

Shadeスイッチとは、各Shadeに配置された乗員が物理操作できるShade制御デバイスである。

スイッチはレバー形および2段階式である。

Shadeスイッチは各Shadeに配置され独立した系統を持つ。

スイッチとShadeは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、Shade制御の主体にはならない。

### 4.17.1.2. Shadeスイッチの機能

Shadeスイッチは乗員の操作によるスイッチ入力をOSへ提供し、Shadeオブジェクトへ制御要求を伝達する役割を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

### 4.17.1.3. Shadeスイッチに対する制御の調停

Shadeスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、Shadeオブジェクトへルーティングされる。

調停はShade側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、Shade側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

## 4.17.2. 機能クラス

ShadeSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | Shade操作入力を提供する基本機能 |

## 4.17.3. 構成情報

ShadeSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | ShadeSwitchCategoryType | スイッチ系統（FrontRight　FrontLeft　RearRight　RearLeft　Roof） |

| | | |
|---|---|---|
| capabilities | enumSet (ShadeSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（UP側のみ、DONW側のみ、２段階スイッチかなど、OEM依存） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.17.4. 状態

ShadeSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | ShadeSwitchMainStatusType | スイッチの入力状態（UP側のみ、DONW側のみ、２段階スイッチ、など） |

## 4.17.5. サービスコール一覧

ShadeSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのShadeスイッチの構成情報の参照 |
| getStatus | Shadeスイッチの状態参照 |
| notify | Shadeスイッチイベントの通知要求 |
| unnotify | Shadeスイッチイベントの通知停止 |
| getEvent | Shadeスイッチイベントの取得 |

## 4.17.6. イベント

Shadeスイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

ShadeSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | Front/RearスイッチのmainStatus入力状態が変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | ShadeSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.17.7. データ型の定義

### 4.17.7.1. Shadeスイッチの機能クラス

```
type ShadeSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.17.7.2. スイッチ系統

```
type ShadeSwitchCategoryType = enum {
    FrontRight, // FrontRightShadeスイッチ系統
    FrontLeft,  // FrontLeftShadeスイッチ系統
    RearRight,  // RearRightShadeスイッチ系統
    RearLeft,   // RearLeftShadeスイッチ系統
    Roof    // Roofスイッチ系統
};
```

### 4.17.7.3. Shadesスイッチの入力状態

```
type ShadeSwitchMainStatusType = enum {
    False,  // OFF - Shade停止
    Up, // UP - ShadeUP
    UpAUuto,    // UPAuto - ShadeUP AUTO
    Down,   // Down - ShadeUP
    DownAuto    // DownAuto - Shades Down Auto
};
```

### Shadeスイッチ構成情報（getConfigAll が返すデータ型）

```
type ShadeSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : ShadeSwitchCategoryType,
    capabilities : enumSet(ShadeSwitchCapabilityType),
    switchType : String,
    displayName : String
```

```
};
```

#### 4.17.7.4. Shadeスイッチ状態

```
type ShadeSwitchStatusType = struct {
    mainStatus : ShadeSwitchMainStatusType
};
```

Shadeスイッチの状態情報

#### 4.17.7.5. Shadeスイッチイベント種別

```
type ShadeSwitchEventKindType = enum {
    MainStatusChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type ShadeSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : ShadeSwitchCategoryType,
    eventInfo : variant(ShadeSwitchEventKindType) {
        MainStatusChanged {
            newState : ShadeSwitchMainStatusType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.17.8. サービスコールの詳細規定

#### getConfigAll

すべてのShadeスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | ShadeSwitchConfigType[] | すべてのShadeスイッチに関する構成情報 |

**【機能】**

すべてのShadeスイッチに関する構成情報を返す。

## getStatus

Shadeスイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のShadeスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | ShadeSwitchStatusType | 対象のShadeスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のShadeスイッチに関する状態を返す。


status には mainStatus（各Shade系統のスイッチ入力状態）を含む。

## notify

Shadeスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のShadeスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (ShadeSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のShadeスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

Shadeスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

Shadeスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | ShadeSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

【機能】

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.18. Vehicle.Cabin.TrunkSwitch（略称：TrunkSwitch）

## 4.18.1. 位置付けと機能

Trunkを制御する物理的なスイッチを扱う。スイッチとTrunkは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.18.1.1. Trunkスイッチの位置付けと性質

Trunkスイッチとは、各Trunkに配置された乗員が物理操作できるTrunk制御デバイスである。

また、ふっとセンサによって足で操作し、自動で開閉するケースもある。

スイッチとTrunkは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、Trunk制御の主体にはならない。

### 4.18.1.2. Trunkスイッチの機能

Trunkスイッチは乗員の操作によるスイッチ入力をOSへ提供し、Trunkオブジェクトへ制御要求を伝達する役割を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

### 4.18.1.3. Trunkスイッチに対する制御の調停

Trunkスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、Trunkオブジェクトへルーティングされる。

調停はTrunk側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、Trunk側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

## 4.18.2. 機能クラス

TrunkSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | Trunk操作入力を提供する基本機能 |

### 4.18.3. 構成情報

TrunkSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | TrunkSwitchCategoryType | スイッチ系統（Fron　Rear） |
| capabilities | enumSet (TrunkSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（Open、Close、など、OEM依存） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

### 4.18.4. 状態

TrunkSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | TrunkSwitchMainStatusType | スイッチの入力状態（Open、Closなど） |

### 4.18.5. サービスコール一覧

TrunkSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのTrunkスイッチの構成情報の参照 |
| getStatus | Trunkスイッチの状態参照 |
| notify | Trunkスイッチイベントの通知要求 |
| unnotify | Trunkスイッチイベントの通知停止 |
| getEvent | Trunkスイッチイベントの取得 |

### 4.18.6. イベント

Trunkスイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

TrunkSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | Front/RearスイッチのmainStatus入力状態が変化 |

| | |
|---|---|
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | TrunkSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.18.7. データ型の定義

### 4.18.7.1. Trunkスイッチの機能クラス

```
type TrunkSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.18.7.2. スイッチ系統

```
type TrunkSwitchCategoryType = enum {
    Front,  // FrontTrunkスイッチ系統
    Rear    // RearTrunkスイッチ系統
};
```

### 4.18.7.3. Trunksスイッチの入力状態

```
type TrunkSwitchMainStatusType = enum {
    False,  // OFF - Trunk停止
    Open,   // Open - TrunkOpen
    Close   // Close - TrunkClose
};
```

### Trunkスイッチ構成情報（getConfigAll が返すデータ型）

```
type TrunkSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : TrunkSwitchCategoryType,
```

```
    capabilities : enumSet(TrunkSwitchCapabilityType),
    switchType : String,
    displayName : String
};
```

#### 4.18.7.4. Trunkスイッチ状態

```
type TrunkSwitchStatusType = struct {
    mainStatus : TrunkSwitchMainStatusType
};
```

Trunkスイッチの状態情報

#### 4.18.7.5. Trunkスイッチイベント種別

```
type TrunkSwitchEventKindType = enum {
    MainStatusChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type TrunkSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : TrunkSwitchCategoryType,
    eventInfo : variant(TrunkSwitchEventKindType) {
        MainStatusChanged {
            newState : TrunkSwitchMainStatusType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.18.8. サービスコールの詳細規定

#### getConfigAll

すべてのTrunkスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | TrunkSwitchConfigType[] | すべてのTrunkスイッチに関する構成情報 |

**【機能】**

すべてのTrunkスイッチに関する構成情報を返す。

## getStatus

Trunkスイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のTrunkスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | TrunkSwitchStatusType | 対象のTrunkスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のTrunkスイッチに関する状態を返す。


status には mainStatus（Front/Rear系統のスイッチ入力状態）を含む。

## notify

Trunkスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のTrunkスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (TrunkSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のTrunkスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

Trunkスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

Trunkスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | TrunkSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

【機能】

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.19. Vehicle.Cabin.Window（略称：Window）

## 4.19.1. 位置付けと機能

Vehicle.Cabin.Window（略称：Window）は，車両のウィンドウ（窓）を操作するためのマルチインスタンスオブジェクトである。

### 4.19.1.1. ウィンドウの位置付けと性質

ウィンドウとは，開閉できる車室の開口部で，車両の走行中も開けておいて差し支えないものと定義する。サンルーフやコンバーチブルトップ（オープンカーの開閉する屋根）もウィンドウと扱う。それに対して，フロントウィンドウ，リアウィンドウは，開閉できないため，ウィンドウとは扱わない（ガラスと扱い，フロントガラス，リアガラスと呼ぶことにする）。ウィンドウを開閉するための物理スイッチは，ウィンドウとは別のオブジェクトとして扱う。

ウィンドウは，閉じていると気体・液体を通さず，開いていると気体・液体を通す。光は，開閉に関わらず通すが，光を遮るためのサンシェードを備えている場合がある。人が通ることや大きい荷物を出し入れすることは想定していない。

常に光を遮るウィンドウは，現時点では想定していない。

### 4.19.1.2. Windowの機能

Windowは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

チルト機構を持つサンルーフのように，可動部を二次元に移動できるウィンドウを扱うために，Windowは，オプションで，可動部を第二次元方向に移動させる機能を持つ。この場合，第二次元方向の移動に対してもMovableObjectと同等の機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉，100：全開）で表す。また，0に近づく動作を閉動作，100に近づく動作を開動作と呼ぶ。

挟み込み防止機能を持つWindowは，可動部の閉動作中に挟み込みを検知すると，閉動作を停止し，開動作を開始する。挟み込み検知後にどの位置まで開くかは，車両依存である。

可動部が停止中の間は，現在位置が変化することはない。可動部が強制的に移動されたなどの理由で現在位置が変化した場合には，Windowが一時的に故障し，すぐに回復したものと扱う。

### 4.19.1.3. Windowに対する制御の調停

Windowに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，Windowを閉めるサービスコールが呼び出され，Windowが閉じる動作中に，開くサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，Windowは開く動作を始める。

また，Windowは標準的なロック機能を持つLockableObjectである。ウィンドウには，機械的なロック機

構は無いのが一般的であり，ソフトウェアでロック機能を実現することを想定している。

Windowが故障または挟み込みを検知した場合，ロックは解除され，故障中と挟み込み回避中は，ロックを取得することができない。

## 4.19.2. 機能クラス

Windowは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーウィンドウであり，APIにより移動させることができる |
| ClosedDetectable | ウィンドウが全閉状態であることを検出することができる |
| HasMove2 | 第二次元方向に移動する機能を持つ |
| MoveProfileSupported | 移動プロファイルをサポートしている |

## 4.19.3. リスククラス

Windowは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | ウィンドウを閉めることに伴う挟み込みのリスク |
| RiskOpen | ウィンドウを開けることに伴う諸々のリスク |

RiskOpenをさらに分類するかは，今後の課題である。

## 4.19.4. 構成情報

Windowの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスID |
| placement | PlacementType | 車室内でのウィンドウの位置 |
| capabilities | enumSet (WindowCapabilityType) | 備えている機能クラスの集合 |
| riskClasses | enumSet (WindowRiskType) | 対策できていないリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayNameKey | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を以下の構成情報として返す。

| | | |
|---|---|---|
| displayName | String | ユーザに表示する呼称 |

## 4.19.5. 状態

Windowの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | WindowMainStatusType | 主状態 |
| mainStatus2 | WindowMainStatusType? | 第二次元方向の主状態 |
| lockStatus | LockStatusType | ロック状態 |
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |
| position2 | PositionType? | 第二次元方向の現在位置 |
| targetPosition2 | PositionType? | 第二次元方向の目標位置 |

mainStatus2，position2，targetPosition2は，機能クラスHasMove2を備えている場合にのみ有効である。

Windowの主状態（mainStatusとmainStatus2）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。また，位置のパーセンテージが大きくなる方へ移動中を開動作中，位置のパーセンテージが小さくなる方へ移動中を閉動作中と呼ぶ（Figure 11）。

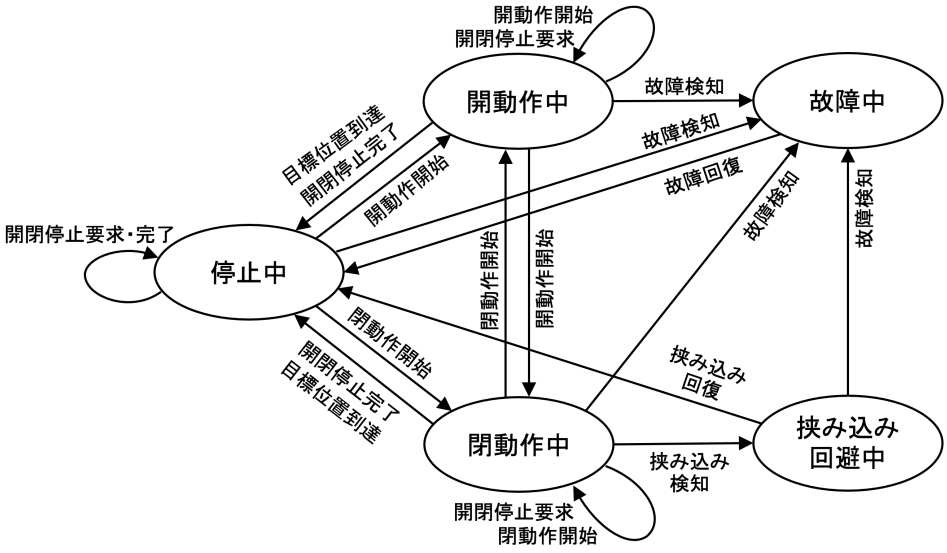



*Figure 11. Windowの主状態の状態遷移*

Windowの現在位置（position）と目標位置（targetPosition），第二次元方向の現在位置（position2）と目標位置（targetPosition2）は，MovableObjectの現在位置と目標位置に関する規定に従う。

## 4.19.6. サービスコール一覧

Windowに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | Windowのすべてのインスタンスの構成情報の参照 |
| getStatus | Windowの状態の参照 |
| startMove | Windowの開閉の開始 |
| startMove2 | Windowの第二次元方向の開閉の開始 |

| | |
|---|---|
| stopMove | Windowの開閉の停止 |
| lock | Windowのロックの取得 |
| unlock | Windowのロックの解除 |
| notify | Windowイベントの通知要求 |
| unnotify | Windowイベントの通知停止 |
| getEvent | Windowイベントの取得 |

### 4.19.7. イベント

Windowに対するイベント種別には，すべてのオブジェクトが持つイベント，MovableObjectが持つイベント，LockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Windowに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がTargetReachedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.19.8. データ型の定義

### (1) Windowの機能クラス

```
type WindowCapabilityType = enum {
    Movable,
    ClosedDetectable,
    HasMove2,
    MoveProfileSupported
};
```

### (2) Windowに対するリスククラス

```
type WindowRiskType = enum {
    RiskPinch,
    RiskOpen
};
```

### (3) Windowの構成情報（getConfigAllが返すデータ型）

```
type WindowConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(WindowCapabilityType),
    riskClasses : enumSet(WindowRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

### (4) Window状態

```
type WindowMainStatusType = enum {
    Stopped,    // 停止中
    Opening,    // 開動作中
    Closing,    // 閉動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type WindowStatusType = struct {
    mainStatus : WindowMainStatusType,
    mainStatus2 : WindowMainStatusType?,
    lockStatus : LockStatusType,
    position : PositionType,
    targetPosition : PositionType,
```

```
    position2 : PositionType?,
    targetPosition2 : PositionType?
};
```

mainStatus2，position2，targetPosition2は，機能クラスHasMove2を備えている場合にのみ値を持つ。

### (5) Windowの移動プロファイル

```
type WindowMoveProfileType = enum {
    Standard,   // 標準的な移動
    Fast,   // 高速で移動
    Slow    // 低速で移動
};
```

### (6) Windowイベント

```
type WindowEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type WindowEventType = struct {
    instanceId : IdType,
    eventInfo : variant(WindowEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        TargetReached {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.19.9. サービスコールの詳細規定

### getConfigAll

Windowのすべてのインスタンスの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | WindowConfigType[] | Windowのすべてのインスタンスに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

Windowのすべてのインスタンスのすべての構成情報を返す。

### getStatus

Windowの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | WindowStatusType | 対象Windowの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

状態参照対象のWindowに関するすべての状態を返す。

### startMove

Windowの開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |
| targetPosition | PositionType | 目標位置（開閉度） |
| moveProfile | WindowMoveProfileType? | 移動プロファイル |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，moveProfile，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のWindowが故障中 |
| E_OBJECT_STATUS | 操作対象のWindowが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のWindowが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し，操作対象のWindowの開閉動作を開始する。必要な場合には，第二次元方向への開閉動作も開始する（例えば，第二次元方向に閉まっている時しか，第一次元方向に開くことができない場合）。

targetPositionに0を指定すれば閉動作の開始，100を指定すれば開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，Windowを開く方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskOpenに対応できていない場合でも，Windowを閉じる方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## startMove2

Windowの第二次元方向の開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |
| targetPosition2 | PositionType | 目標位置（開閉度） |

| | | |
|---|---|---|
| moveProfile | WindowMoveProfileType? | 移動プロファイル |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition2，moveProfile，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のWindowが故障中 |
| E_OBJECT_STATUS | 操作対象のWindowが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のWindowが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_NOT_SUPPORTED | 第二次元方向に移動する機能を持たない |

**【機能】**

第二次元方向の目標位置をtargetPosition2に設定し，操作対象のWindowの第二次元方向への開閉動作を開始する。必要な場合には，第一次元方向への開閉動作も開始する（例えば，第一次元方向に閉まっている時しか，第二次元方向に開くことができない場合）。

targetPosition2に0を指定すれば閉動作の開始，100を指定すれば開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，Windowを開く方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskOpenに対応できていない場合でも，Windowを閉じる方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

Windowの開閉の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のWindowが故障中 |
| E_OBJECT_STATUS | 操作対象のWindowが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のWindowが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のWindowの各次元方向への開閉動作の停止を要求する。

1つの次元方向への開閉動作を停止させる機能は用意していない。

## lock

Windowのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のWindowが，priorityと同じかそれより高い優先度でロックされている（自アプリケーションが，priorityと同じかそれより高い優先度でロックしている場合を含む） |
| E_OBJECT_FAULT | ロック対象のWindowが故障中 |
| E_OBJECT_STATUS | ロック対象のWindowが挟み込み回避中 |

**【機能】**

ロック対象のWindowを，指定した操作優先度でロックする。

## unlock

Windowのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象のWindowをロックしていない |

**【機能】**

ロック解除対象のWindowを，ロック解除する。

## notify

Windowイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のWindowのID |
| eventFilter | enumSet (WindowEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のWindowで発生したeventFilterで指定した種別のイベントの通知を要求する。イベントを格納するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種別のイベントの通知を要求する。

## unnotify

Windowイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。イベントを格納するためのイベントキューを削除する。

## getEvent

Windowイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントを取得するイベントキューの通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | WindowEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.20. Vehicle.Cabin.WindowSwitch（略称：WindowSwitch）

## 4.20.1. 位置付けと機能

Windowを制御する物理的なスイッチを扱う。スイッチとWindowは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.20.1.1. Windowスイッチの位置付けと性質

Windowスイッチとは、各Windowに配置された乗員が物理操作できるWindow制御デバイスである。

スイッチはレバー形および2段階式である。

Windowスイッチは各Windowに配置され独立した系統を持つ。

スイッチとWindowは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、Window制御の主体にはならない。

### 4.20.1.2. Windowスイッチの機能

Windowスイッチは乗員の操作によるスイッチ入力をOSへ提供し、Windowオブジェクトへ制御要求を伝達する役割を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

### 4.20.1.3. Windowスイッチに対する制御の調停

Windowスイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、Windowオブジェクトへルーティングされる。

調停はWindow側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、Window側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

## 4.20.2. 機能クラス

WindowSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | Window操作入力を提供する基本機能 |

## 4.20.3. 構成情報

WindowSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | WindowSwitchCategoryType | スイッチ系統（FrontRight　FrontLeft　RearRight　RearLeft　Roof） |
| capabilities | enumSet (WindowSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（開側、閉側、２段階スイッチ、IsWindowChildLockEngagedの機能ON/OFFのスイッチかなど、OEM依存） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.20.4. 状態

WindowSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | WindowSwitchMainStatusType | スイッチの入力状態（開側、閉側、２段階スイッチ） |
| windowChildLockStatus | WindowChildLockStatusType | IsWindowChildLockEngagedの機能ON/OFFのスイッチの入力状態。（ON、OFF） |

## 4.20.5. サービスコール一覧

WindowSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのWindowスイッチの構成情報の参照 |
| getStatus | Windowスイッチの状態参照 |
| notify | Windowスイッチイベントの通知要求 |
| unnotify | Windowスイッチイベントの通知停止 |
| getEvent | Windowスイッチイベントの取得 |

## 4.20.6. イベント

Windowスイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

WindowSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | ドア付属スイッチのmainStatus入力状態が変化 |
| ChildLockStatusChanged | IsWindowChildLockEngagedの機能ON/OFFのスイッチの入力状態が変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | WindowSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がChildLockStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | WindowSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.20.7. データ型の定義

### 4.20.7.1. Windowスイッチの機能クラス

```
type WindowSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.20.7.2. スイッチ系統

```
type WindowSwitchCategoryType = enum {
    FrontRight, // FrontRightWindowスイッチ系統
    FrontLeft,  // FrontLeftWindowスイッチ系統
    RearRight,  // RearRightWindowスイッチ系統
    RearLeft,   // RearLeftWindowスイッチ系統
    Roof    // Roofスイッチ系統
};
```

### 4.20.7.3. Windowsスイッチの入力状態

```
type WindowSwitchMainStatusType = enum {
```

```
    NoAction,   // NoAction（操作なし）
    ManualOpen, // ManualOpen（開1段階目）
    AutoOpen,   // AutoOpen（開2段階目）
    ManualClose,    // ManualClose（閉1段階目）
    AutoClose   // AutoClose（閉2段階目）
};
```

#### 4.20.7.4. IsWindowChildLockEngagedの機能ON/OFFのスイッチの入力状態

```
type WindowChildLockStatusType = enum {
    False,  // IsWindowChildLockEngagedの機能OFF
    True    // IsWindowChildLockEngagedの機能ON
};
```

#### Windowスイッチ構成情報（getConfigAll が返すデータ型）

```
type WindowSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : WindowSwitchCategoryType,
    capabilities : enumSet(WindowSwitchCapabilityType),
    switchType : String,
    displayName : String
};
```

#### 4.20.7.5. Windowスイッチ状態

```
type WindowSwitchStatusType = struct {
    mainStatus : WindowSwitchMainStatusType,
    windowChildLockStatus : WindowChildLockStatusType
};
```

Windowスイッチの状態情報

#### 4.20.7.6. Windowスイッチイベント種別

```
type WindowSwitchEventKindType = enum {
    MainStatusChanged,
    ChildLockStatusChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type WindowSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : WindowSwitchCategoryType,
    eventInfo : variant(WindowSwitchEventKindType) {
        MainStatusChanged {
            newState : WindowSwitchMainStatusType
        },
        ChildLockStatusChanged {
            newState : WindowSwitchMainStatusType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.20.8. サービスコールの詳細規定

### getConfigAll

すべてのWindowスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | WindowSwitchConfigType[] | すべてのWindowスイッチに関する構成情報 |

**【機能】**

すべてのWindowスイッチに関する構成情報を返す。

### getStatus

Windowスイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のWindowスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | WindowSwitchStatusType | 対象のWindowスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のWindowスイッチに関する状態を返す。

status には mainStatus（各Window系統のスイッチ入力状態）を含む。

## notify

Windowスイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のWindowスイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (WindowSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のWindowスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

Windowスイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

Windowスイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | WindowSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.21. Vehicle.Cabin.WiperSwitch（略称：WiperSwitch）

## 4.21.1. 位置付けと機能

ワイパーを制御する物理的なスイッチを扱う。スイッチとワイパーは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

### 4.21.1.1. ワイパースイッチの位置付けと性質

ワイパースイッチとは、車両のハンドル周辺に設置された、乗員が物理操作できるワイパー制御デバイスである。

スイッチはレバー形およびダイアル式であり、運転席のハンドルに装着されている。

ワイパースイッチはFront（フロント用）、Rear（リア用）、Washer（ウォッシャー用）の3つの独立した系統を持つ。

スイッチとワイパーは直接接続されず、制御要求はビークルOSが受け取る。

スイッチは入力源として機能し、ワイパー制御の主体にはならない。

#### 4.21.1.2. ワイパースイッチの機能

ワイパースイッチは乗員の操作によるスイッチ入力をOSへ提供し、ワイパーオブジェクトへ制御要求を伝達する役割を持つ。

Front/Rearスイッチは、OFF、MIST、INT、LOW、HIGH、AUTOの6つの状態を持つ。INTモードでは、ダイアルにより0〜30秒の間欠時間を調整することができる。

Washerスイッチは、ON/OFFの2つの状態を持つ。

各スイッチの系統はconfigData内のswitchCategoryによって識別される。

#### 4.21.1.3. ワイパースイッチに対する制御の調停

ワイパースイッチは入力イベントの発生源であり、自身が制御競合を調停することはない。

入力はOSが一元管理し、ワイパーオブジェクトへルーティングされる。

調停はワイパー側の後勝ち原則に従う。スイッチ操作が他アプリケーションの制御要求と競合した場合、ワイパー側の調停ロジックにより採用または却下が判断される。

スイッチ自身は制御主体ではなく、アプリケーションや自動制御と同様に「入力の一つ」として扱われる。

### 4.21.2. 機能クラス

WiperSwitchは，以下の機能クラスを持つ。

| | |
|---|---|
| SwitchControl | ワイパー操作入力を提供する基本機能 |

### 4.21.3. 構成情報

WiperSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| switchCategory | WiperSwitchCategoryType | スイッチ系統（Front、Rear、Washer） |
| capabilities | enumSet (WiperSwitchCapabilityType) | このスイッチがサポートする操作機能の集合 |
| switchType | String | スイッチ種別（Lever形またはDial形など、OEM依存） |

| | | |
|---|---|---|
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.21.4. 状態

WiperSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | WiperSwitchMainStatusType | スイッチの入力状態（Front/Rear系統の場合：OFF、MIST、INT、LOW、HIGH、AUTO） |
| washerStatus | WiperSwitchWasherStatusType | Washerスイッチの入力状態。switchCategoryがWasherの場合に有効（ON、OFF） |
| intervalTime | UInt32 | スイッチがINT状態の場合の間欠時間（秒）。0～30。ダイアルで調整可能。 |

## 4.21.5. サービスコール一覧

WiperSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのワイパースイッチの構成情報の参照 |
| getStatus | ワイパースイッチの状態参照 |
| notify | ワイパースイッチイベントの通知要求 |
| unnotify | ワイパースイッチイベントの通知停止 |
| getEvent | ワイパースイッチイベントの取得 |

## 4.21.6. イベント

ワイパースイッチに関するイベントとして、スイッチ状態変化、故障検出/回復、構成変更、イベントキューのオーバーフローなどを定義する。

WiperSwitchに対するイベント種別は以下の通り。

| | |
|---|---|
| MainStatusChanged | Front/RearスイッチのmainStatus入力状態が変化 |
| WasherStatusChanged | Washerスイッチの入力状態が変化 |
| IntervalTimeChanged | Intermittent（INT）モード時の間欠時間が変化 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がMainStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | WiperSwitchMainStatusType | スイッチの新しい入力状態 |

また，イベント種別がWasherStatusChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newState | WiperSwitchWasherStatusType | Washerスイッチの新しい入力状態 |

また，イベント種別がIntervalTimeChangedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| newIntervalTime | UInt32 | 新しい間欠時間（秒） |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.21.7. データ型の定義

### 4.21.7.1. ワイパースイッチの機能クラス

```
type WiperSwitchCapabilityType = enum {
    SwitchControl
};
```

### 4.21.7.2. スイッチ系統

```
type WiperSwitchCategoryType = enum {
    Front,  // フロントワイパースイッチ系統
    Rear,   // リアワイパースイッチ系統
    Washer  // ウォッシャースイッチ系統
};
```

### 4.21.7.3. Front/Rearスイッチの入力状態

```
type WiperSwitchMainStatusType = enum {
    False,  // OFF - ワイパー停止
    Mist,   // MIST - ミスト動作（短時間ワイパー動作）
    Intermittent,   // INT - 間欠動作。ダイアルでintervalTimeを調整可能。
    Low,    // LOW - 低速連続動作
    High,   // HIGH - 高速連続動作
    Auto    // AUTO - 自動動作（雨量センサー連動）
};
```

#### 4.21.7.4. Washerスイッチの入力状態

```
type WiperSwitchWasherStatusType = enum {
    False,  // OFF - ウォッシャー停止
    True    // ON - ウォッシャー液噴射動作
};
```

### ワイパースイッチ構成情報（getConfigAll が返すデータ型）

```
type WiperSwitchConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    switchCategory : WiperSwitchCategoryType,
    capabilities : enumSet(WiperSwitchCapabilityType),
    switchType : String,
    displayName : String
};
```

#### 4.21.7.5. ワイパースイッチ状態

```
type WiperSwitchStatusType = struct {
    mainStatus : WiperSwitchMainStatusType,
    washerStatus : WiperSwitchWasherStatusType,
    intervalTime : UInt32
};
```

ワイパースイッチの状態情報

#### 4.21.7.6. ワイパースイッチイベント種別

```
type WiperSwitchEventKindType = enum {
    MainStatusChanged,
    WasherStatusChanged,
    IntervalTimeChanged,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
};
```

```
type WiperSwitchEventType = struct {
    instanceId : IdType,
    switchCategory : WiperSwitchCategoryType,
    eventInfo : variant(WiperSwitchEventKindType) {
        MainStatusChanged {
```

```
            newState : WiperSwitchMainStatusType
        },
        WasherStatusChanged {
            newState : WiperSwitchWasherStatusType
        },
        IntervalTimeChanged {
            newIntervalTime : UInt32
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.21.8. サービスコールの詳細規定

### getConfigAll

すべてのワイパースイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | WiperSwitchConfigType[] | すべてのワイパースイッチに関する構成情報 |

**【機能】**

すべてのワイパースイッチに関する構成情報を返す。

### getStatus

ワイパースイッチの状態参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパースイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | WiperSwitchStatusType | 対象のワイパースイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |

**【機能】**

操作対象のワイパースイッチに関する状態を返す。

status　には　mainStatus（Front/Rear系統のスイッチ入力状態）、washerStatus（Washer系統のスイッチ入力状態）、intervalTime（INT時の間欠時間）を含む。

## notify

ワイパースイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のワイパースイッチID（省略時は全インスタンス） |
| eventFilter | enumSet (WiperSwitchEventKindType)? | 通知するイベント種別の集合（省略時は全種） |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceId が有効範囲外 |
| E_INVALID_PARAMETER | eventFilter が有効範囲外 |

**【機能】**

イベント通知対象のワイパースイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

ワイパースイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

ワイパースイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別する通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | WiperSwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandle が不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.22. Vehicle.CargoSpace（略称：CargoSpace）

＜TBD＞

# 4.23. Vehicle.CargoSpace.Trunk（略称：Trunk）

## 4.23.1. 位置付けと機能

Vehicle.CargoSpace.Trunk（略称：Trunk）は，車両のトランク（荷室）を操作するためのマルチインスタンスオブジェクトである。

### 4.23.1.1. トランクの位置付けと性質

開閉できる車室の開口部の中で、車両の走行中は通常閉じておくもの。または、大きい荷物を出し入れすることを想定しているもの。人が通ることは想定していない。

### Trunkの機能

Trunkは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉，100：全開）で表す。また，0に近づく動作を閉動作，100に近づく動作を開動作と呼ぶ。

挟み込み防止機能を持つトランクは，可動部の閉動作中に挟み込みを検知すると，閉動作を停止し，開動作を開始する。挟み込み検知後にどの位置まで開くかは，車両依存である。

可動部が停止中の間も，マニュアル操作にて現在位置が変化することがあり得る。

#### 4.23.1.2. トランクに対する制御の調停

トランクに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，トランクを閉めるサービスコールが呼び出され，トランクが閉じる動作中に，開くサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，トランクは開く動作を始める。

また，トランクは標準的なロック機能を持つLockableObjectである。トランクには，機械的なロック機構は無いのが一般的であり，ソフトウェアでロック機能を実現することを想定している。

トランクが故障または挟み込みを検知した場合，ロックは解除され，故障中と挟み込み回避中は，ロックを取得することができない。

## 4.23.2. 機能クラス

Trunkは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワートランクであり，APIにより移動させることができる。トランクロックアンラッチ機能のみを有する者も含む |
| ClosedDetectable | トランクが全閉状態であることを検出することができる |
| MoveProfileSupported | 移動プロファイルをサポートしている |

## 4.23.3. リスククラス

Trunkは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | トランクを閉めることに伴う挟み込みのリスク |
| RiskOpen | トランクを開けることに伴う諸々のリスク |

**【今後の検討課題】**

RiskOpenをさらに分類するかは，今後の課題である。

## 4.23.4. 構成情報

Trunkの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |

| | | |
|---|---|---|
| capabilities | enumSet (TrunkCapabilityType) | そのトランクが持つ機能 |
| riskClasses | enumSet(TrunkRiskType) | そのトランクのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.23.5. 状態

Trunkの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | TrunkMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |

トランクの主状態（mainStatus）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。また，位置のパーセンテージが大きくなる方へ移動中を開動作中，位置のパーセンテージが小さくなる方へ移動中を閉動作中と呼ぶ。

トランクの現在位置（position）と目標位置（targetPosition）は，MovableObjectの現在位置と目標位置に関する規定に従う。

トランクロック/アンラッチのみの機能の場合は、開要求のみを受け付け、閉要求は受け付けないものとする。また、アンラッチ完了時をもって目標位置到達と考える。

## 4.23.6. サービスコール一覧

Trunkに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのトランクの構成情報の参照 |
| getStatus | トランクの状態の参照 |
| startMove | トランクの開閉の開始 |
| stopMove | トランクの開閉の停止 |
| lock | トランクのロックの取得 |
| unlock | トランクのロックの解除 |
| notify | トランクイベントの通知要求 |
| unnotify | トランクイベントの通知停止 |

| | |
|---|---|
| getEvent | トランクイベントの取得 |

## 4.23.7. イベント

トランクに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Trunkに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplicatio n | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.23.8. データ型の定義

### 4.23.8.1. トランクの機能クラス

```
type TrunkCapabilityType = enum {
    Movable,
    ClosedDetectable,
    MoveProfileSupported
};
```

### 4.23.8.2. トランクに対するリスククラス

```
type TrunkRiskType = enum {
    RiskPinch,
    RiskOpen
};
```

#### 4.23.8.3. トランクの構成情報（getConfigAllが返すデータ型）

```
type TrunkConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(TrunkCapabilityType),
    riskClasses : enumSet(TrunkRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.23.8.4. トランク状態

```
type TrunkMainStatusType = enum {
    FullyStopped,   // 停止中（全閉）
    UnlatchedStopped,   // 停止中（半ドア　or　ドア開）
    Opening,    // 開動作中
    Closing,    // 閉動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type TrunkStatusType = struct {
    mainStatus : TrunkMainStatusType,
    lockStatus : LockStatusType,
    position : PositionType,
    targetPosition : PositionType
};
```

#### 4.23.8.5. 移動プロファイル

```
type TrunkMoveProfileType = enum {
    TrunkMoveProfileStandard,
    TrunkMoveProfileFast,
    TrunkMoveProfileSlow
};
```

#### 4.23.8.6. ウィンドウイベント

```
type TrunkEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type TrunkEventType = struct {
    instanceId : IdType,
    eventInfo : variant(TrunkEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.23.9. サービスコールの詳細規定

#### getConfigAll

すべてのトランクの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | TrunkConfigType[] | すべてのトランクに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのトランクに関する構成情報を返す。

## getStatus

トランクの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のトランクのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | TrunkStatusType | 対象トランクの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のトランクに関するすべての状態を返す。

## startMove

トランクの開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のトランクのID |
| targetPosition | PositionType | 目標位置（開閉度） |
| moveProfile | TrunkMoveProfileType | 移動プロファイル |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のトランクが故障中 |

| | |
|---|---|
| E_OBJECT_STATUS | 操作対象のトランクが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のトランクが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し，操作対象のトランクの開閉動作を開始する。

targetPositionに0を指定すれば閉動作の開始，100を指定すれば開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，トランクを開く方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskOpenに対応できていない場合でも，トランクを閉じる方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

トランクの開閉の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のトランクのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のトランクが故障中 |
| E_OBJECT_STATUS | 操作対象のトランクが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のトランクが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のトランクの各次元方向への開閉動作の停止を要求する。

## lock

トランクのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のトランクのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のトランクが，priorityと同じかそれより高い優先度でロックされている |
| E_OBJECT_FAULT | ロック対象のトランクが故障中 |
| E_OBJECT_STATUS | ロック対象のトランクが挟み込み回避中 |

**【機能】**

ロック対象のトランクを，指定した操作優先度でロックする。

## unlock

トランクのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のトランクのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象のトランクをロックしていない |

**【機能】**

ロック解除対象のトランクを，ロック解除する。

## notify

トランクイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のトランクのID |
| eventFilter | enumSet (TrunkEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のトランクで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

トランクイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

トランクイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | TrunkEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.24. Vehicle.Body（略称：Body）

＜TBD＞

# 4.25. Vehicle.Body.Hood（略称：Hood）

## 4.25.1. 位置付けと機能

Vehicle.Body.Hood（略称：Hood）は、車両のボンネットを操作するためのシングルトンオブジェクトである。

### 4.25.1.1. ボンネットの位置付けと性質

ボンネットとは、エンジンルーム(フロントコンパートメント)を覆う外板であり、車両の走行中は閉じていることを前提として定義する。ボンネットを開けるためのオープナーは、ボンネットとは別のオブジェクトとして扱う。

ボンネットは、閉じている状態ではエンジン等の内部機器を外部環境から保護するとともに、衝突時の衝撃緩和、空力性能、熱対策、騒音低減および車体構造の補助といった車両性能・安全性能の一部を担う役割を持つ。

開いている状態では、内部の点検・整備、部品交換および診断作業を行うためのアクセスを可能にし、車両の保守性および維持管理性を確保する役割を持つ。

### 4.25.1.2. Hoodの機能

Hoodの機能は、可動部を一次元に移動できるオブジェクトであり、MovableObjectの機能を持つ。

ラッチを解除し、閉状態からポップアップ状態に一次元に移動するものである。そのため、ポップアップ状態から全開状態、全開状態から閉状態への移行は手動で行うことを想定している。

可動部の位置は、開方向の移動に対して、どれだけ開いているかのパーセンテージ（開閉度、0：全閉、100：ポップアップ完了）で表す。

## 4.25.2. 機能クラス

Hoodは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーボンネットではないが、APIにより移動させることができる |
| ClosedDetectable | ボンネットが全閉状態であることを検出することができる |
| HasMove | 第一次元方向に移動する機能を持つ |

## 4.25.3. リスククラス

Hoodは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskOpen | ボンネットを開ける／開いていることに伴う諸々のリスク |

**【今後の検討課題】**

RiskOpenをさらに分類するかは、今後の課題である。

## 4.25.4. 構成情報

Hoodの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| capabilities | enumSet (HoodCapabilityType) | ボンネットが持つ機能 |
| riskClasses | enumSet(HoodRiskType) | ボンネットのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車室） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は、YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは、この構成情報に対して、次の変換を行った構成情報を返す。

- mountedOnは、物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては、それを元に現在のロケールに対応する呼称の文字列を取得し、その文字列を返す。

## 4.25.5. 状態

Hoodの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | HoodMainStatusType | 主状態(全閉状態、ポップアップ状態、故障状態を想定) |

| | | |
|---|---|---|
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |
| priority | PriorityType? | 操作優先度 |

ボンネットの主状態は、MovableObjectの主状態に加えて、故障中の状態をとる。また、位置のパーセンテージが大きくなる方へ移動中をポップアップ動作中と呼ぶ。

ウィンドウの現在位置（position）と目標位置（targetPosition）は、MovableObjectの現在位置と目標位置に関する規定に従う。

## 4.25.6. サービスコール一覧

Hoodに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | ボンネットの構成情報の参照 |
| getStatus | ボンネットの状態の参照 |
| StartMove | ボンネットの開閉の開始 |
| notify | ボンネットイベントの通知要求 |
| unnotify | ボンネットイベントの通知停止 |
| getEvent | ボンネットイベントの取得 |

## 4.25.7. イベント

ボンネットに対するイベント種別には、MovableObjectの持つイベントに加えて、ポップアップ完了、ラッチ解除の検知、故障検知と故障から回復がある。

Hoodに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| PopupReached | 指定された目標位置に到達して停止、または、移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

### 4.25.8. データ型の定義

#### 4.25.8.1. ボンネットの機能クラス

```
type HoodCapabilityType = enum {
    Movable,
    ClosedDetectable,
    HasMove
};
```

#### 4.25.8.2. ボンネットに対するリスククラス

```
type HoodRiskType = enum {
    RiskOpen
};
```

#### 4.25.8.3. ボンネットの構成情報（getConfigが返すデータ型）

```
type HoodConfigType = struct {
    instanceId : IdType,
    capabilities : enumSet(HoodCapabilityType),
    riskClasses : enumSet(HoodRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.25.8.4. ボンネットの状態

```
type HoodMainStatusType = enum {
    Closed, // 全閉中
    PoppingUp,  // ポップアップ中
    Fault   // 故障中
};
```

#### 4.25.8.5. ボンネットイベント種別

```
type HoodEventKindsType = enum {
    ControlledBySelf,
    ControlledByOther,
    PopupReached,
    FaultDetected,
    FaultRecovered,
    EventOverflow,
    ConfigChanged
```

```
};
```

```
type HoodEventType = struct {
    instanceId : IdType,
    eventInfo : variant(HoodEvenKindsType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.25.9. サービスコールの詳細規定

### getConfig

ボンネットの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | HoodConfigType[] | ボンネットに関する構成情報 |

**【エラーコード】**

| | |
|---|---|
| E_OK | 成功 |
| E_NOT_SUPPORTED | 未サポート（実装により返却されない場合あり） |

**【機能】**

ボンネットに関する構成情報を返す。

### getStatus

ボンネットの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | ボンネットのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | HoodStatusType | ボンネットの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

ボンネットに関するすべての状態を返す。

## StartMove

ボンネットの開閉の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | ボンネットのID |
| targetPosition | PositionType | 目標位置 |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition、priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが、アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のボンネットが故障中 |
| E_OBJECT_STATUS | 操作対象のボンネットが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のボンネットが、他のアプリケーションにより、priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し、ボンネットのポップアップ動作を開始する。

## notify

ボンネットイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | ボンネットのID |
| eventType | HoodEventKindsType? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventTypeが有効範囲外 |

**【機能】**

イベント通知対象のボンネットで発生したeventTypeで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し、その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合、すべてのインスタンスで発生したイベントの通知を要求する。eventTypeを省略した場合、すべての種類のイベントの通知を要求する。

## unnotify

ボンネットイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

ボンネットイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

| | | |
|---|---|---|
| eventDetail | EventDetailType | イベント詳細情報 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから、先頭のイベントを取り出す。取り出したイベントは、イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは、用意する必要がない。

# 4.26. Vehicle.Body.HoodSwitch（略称：HoodSwitch）

## 4.26.1. 位置付けと機能

Vehicle.Body.HoodSwitch（略称：HoodSwitch）ボンネットをポップアップさせる物理的なオープナーを扱う。オープナーとボンネットは直結せず、制御はOS側（ビークルOS）で行うことを仮定する。

## 4.26.2. 構成情報

HoodSwitchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |

## 4.26.3. 状態

HoodSwitchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | HoodSwMainStatusType | スイッチの主状態（正常/停止/故障/電気故障など） |

## 4.26.4. サービスコール一覧

HoodSwitchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | ボンネットポップアップスイッチの構成情報の参照 |
| getStatus | ボンネットポップアップスイッチの状態の参照 |

### 4.26.5. サービスコールの詳細規定

#### getConfig

ボンネットポップアップスイッチの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | HoodSwConfigType[] | ボンネットポップアップスイッチに関する構成情報 |

**【エラーコード】**

| | |
|---|---|
| E_OK | 成功 |
| E_NOT_SUPPORTED | 未サポート（実装により返却されない場合あり） |

**【機能】**

ボンネットポップアップスイッチに関する構成情報を返す。

#### getStatus

ボンネットポップアップスイッチの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | ボンネットポップアップスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | HoodSwStatusType | ボンネットポップアップスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

ボンネットポップアップスイッチに関するすべての状態を返す。

## 4.27. Vehicle.Body.Mirror（略称：Mirror）

## 4.27.1. 位置付けと機能

Vehicle.Body.Mirror（略称：Mirror）は，車両のミラーを操作するためのマルチインスタンスオブジェクトである。

### 4.27.1.1. ミラーの位置付けと性質

車両の外部に設置され、運転者の視界を補助するための装置。走行中の安全性を確保するために重要な役割を果たす。

### ミラーの機能

ミラーは，可動部を一次元に移動できるオブジェクトであり，MovableObjectの機能を持つ。

可動部の位置は，いずれの方向の移動に対しても，どれだけ開いているかのパーセンテージ（開閉度，0：全閉（格納），100：全開（展開））で表す。また，0に近づく動作を格納動作，100に近づく動作を展開動作と呼ぶ。

挟み込み防止機能を持つミラーは，可動部の格納動作中に挟み込みを検知すると，格納動作を停止し，展開動作を開始する。挟み込み検知後にどの位置まで展開するかは，車両依存である

可動部が停止中の間も，マニュアル操作にて現在位置が変化することがあり得る。

### 4.27.1.2. ミラーに対する制御の調停

ミラーに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，ミラーを格納するサービスコールが呼び出され，ミラーが格納動作中に，展開するサービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，ミラーは展開動作を始める。

また，ミラーは標準的なロック機能を持つLockableObjectである。ミラーには，機械的なロック機構は無いのが一般的であり，ソフトウェアでロック機能を実現することを想定している。

ミラーが故障または挟み込みを検知した場合，ロックは解除され，故障中と挟み込み回避中は，ロックを取得することができない。

## 4.27.2. 機能クラス

Mirrorは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーミラーであり，APIにより移動させることができる。 |
| ClosedDetectable | ミラーが全閉（格納）状態であることを検出することができる |

## 4.27.3. リスククラス

Mirrorは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskPinch | ミラーを閉めることに伴う挟み込みのリスク |
| RiskCollision | ミラーを開けることに伴う，歩行者・隣接車両・構造物等への接触のリスク |
| RiskVisibility | ミラーが正規位置まで展開しきれないことに伴う，後方視界不良のリスク |

## 4.27.4. 構成情報

Mirrorの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | インスタンスがある場所 |
| capabilities | enumSet (MirrorCapabilityType) | そのミラーが持つ機能 |
| riskClasses | enumSet(MirrorRiskType) | そのミラーのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ドアまたは車体） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。
- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

## 4.27.5. 状態

Mirrorの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | MirrorMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| position | PositionType | 現在位置 |
| targetPosition | PositionType | 目標位置 |

ミラーの主状態（mainStatus）は，MovableObjectの主状態に加えて，挟み込み回避中の状態をとる。また，位置のパーセンテージが大きくなる方へ移動中を展開動作中，位置のパーセンテージが小さくなる方へ移動中を格納動作中と呼ぶ（図4-Y-1）。

[[Fig-図4-Y-1. ミラーの主状態の状態遷移]] .図4-Y-1. ミラーの主状態の状態遷移 image::../images/image/image1.png[width=50%]

ミラーの現在位置（position）と目標位置（targetPosition）は，MovableObjectの現在位置と目標位置に関する規定に従う。

ミラーロック機能のみの場合は，展開要求のみを受け付け，格納要求は受け付けないものとする。また，ロック解除完了時をもって目標位置到達と考える。

## 4.27.6. サービスコール一覧

Mirrorに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのミラーの構成情報の参照 |

| | |
|---|---|
| getStatus | ミラーの状態の参照 |
| startMove | ミラーの格納・展開の開始 |
| stopMove | ミラーの格納・展開の停止 |
| lock | ミラーのロックの取得 |
| unlock | ミラーのロックの解除 |
| notify | ミラーイベントの通知要求 |
| unnotify | ミラーイベントの通知停止 |
| getEvent | ミラーイベントの取得 |

## 4.27.7. イベント

ミラーに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，挟み込み検知と挟み込みから回復がある。

Mirrorに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| PinchDetected | 挟み込み検知 |
| PinchRecovered | 挟み込みから回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.27.8. データ型の定義

#### 4.27.8.1. ミラーの機能クラス

```
type MirrorCapabilityType = enum {
    Movable,
    ClosedDetectable
};
```

#### 4.27.8.2. ミラーに対するリスククラス

```
type MirrorRiskType = enum {
    RiskPinch,
    RiskCollision,
    RiskVisibility
};
```

#### 4.27.8.3. ミラーの構成情報（getConfigAllが返すデータ型）

```
type MirrorConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(MirrorCapabilityType),
    riskClasses : enumSet(MirrorRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.27.8.4. ミラー状態

```
type MirrorMainStatusType = enum {
    FullyStopped,   // 停止中（全閉（格納））
    UnlatchedStopped,   // 停止中（展開途中）
    Opening,    // 開動作中
    Closing,    // 格納動作中
    Fault,  // 故障中
    PinchAvoiding   // 挟み込み回避中
};
```

```
type MirrorStatusType = struct {
    mainStatus : MirrorMainStatusType,
    lockStatus : LockStatusType,
    position : PositionType,
    targetPosition : PositionType
};
```

#### 4.27.8.5. ミラーイベント

```
type MirrorEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    PinchDetected,
    PinchRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type MirrorEventType = struct {
    instanceId : IdType,
    eventInfo : variant(MirrorEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

### 4.27.9. サービスコールの詳細規定

#### getConfigAll

すべてのミラーの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | MirrorConfigType[] | すべてのミラーに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのミラーに関する構成情報を返す。

## getStatus

ミラーの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のミラーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | MirrorStatusType | 対象ミラーの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のミラーに関するすべての状態を返す。

## startMove

ミラーの格納・展開の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のミラーのID |
| targetPosition | PositionType | 目標位置（開閉度） |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | targetPosition，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のドアが故障中 |
| E_OBJECT_STATUS | 操作対象のドアが挟み込み回避中 |

| | |
|---|---|
| E_OBJECT_LOCKED | 操作対象のドアが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

目標位置をtargetPositionに設定し，操作対象のミラーの格納・展開動作を開始する。

targetPositionに0を指定すれば格納動作の開始，100を指定すれば展開動作の開始の意味になる。

リスククラスRiskPinchに対応できていない場合でも，ミラーを展開する方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

リスククラスRiskCollisionに対応できていない場合でも，ミラーを格納する方向に動作を開始しようとした場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

ミラーの格納・展開の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のミラーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のミラーが故障中 |
| E_OBJECT_STATUS | 操作対象のミラーが挟み込み回避中 |
| E_OBJECT_LOCKED | 操作対象のミラーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

各次元方向の目標位置を現在位置に設定して，操作対象のミラーの各次元方向への格納・展開動作の停止を要求する。

## lock

ミラーのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のミラーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のミラーが，priorityと同じかそれより高い優先度でロックされている |
| E_OBJECT_FAULT | ロック対象のミラーが故障中 |
| E_OBJECT_STATUS | ロック対象のミラーが挟み込み回避中 |

**【機能】**

ロック対象のミラーを，指定した操作優先度でロックする。

## unlock

ミラーのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のミラーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象のミラーをロックしていない |

**【機能】**

ロック解除対象のミラーを，ロック解除する。

## notify

ミラーイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のミラーのID |
| eventFilter | enumSet (MirrorEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のミラーで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

ミラーイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

ミラーイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | MirrorEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.28. Vehicle.Body.Wiper（略称：Wiper）

## 4.28.1. 位置付けと機能

Vehicle.Body.Wiper（略称：Wiper）は，自動車に付着した雨水，雪，汚れなどを払拭する装置を操作するためのマルチインスタンスオブジェクトである。

### 4.28.1.1. ワイパーの位置付けと性質

ワイパーとは，自動車に付着した雨水，雪，汚れなどを払拭する装置で，払拭ブレード，アーム，それを駆動させるモーター，リンク機構を備えている。ワイパーはフロントウィンドウ，リアウィンドウ，ヘッドライト，サイドミラー，カメラなど複数の場所に装着される可能性がある。

また，ワイパーに併設されるウォッシャー機能（洗浄液による払拭）も本APIで扱う。

ワイパーの物理スイッチは，ワイパーとは別のオブジェクトとして扱う。

### 4.28.1.2. Wiperの機能

Wiperは，払拭動作を一次元的に移動するオブジェクトであり，MovableObjectと同等の機能を持つ。

ワイパーは,ブレードを往復動作させてフロントガラスを払拭する「払拭動作」を基本動作とする。

ワイパーの制御モードには,払拭動作を連続して実行する連続モード,および払拭動作を一定時間間隔で実行する間欠モードがあるが,これらの動作はAPIで指定する動作速度レベルと一時停止時間により制御される。

ワイパーの初期位置，目標位置となる端点は固定であるため，ワイパーの目標位置はAPIで指定，参照しない設計とする。

ワイパーが故障または凍結を検知した場合，制御を受け付けない。

#### 4.28.1.3. ワイパーに対する制御の調停

ワイパーに対する制御の調停には，標準的な調停機能（後勝ちの原則）が適用される。すなわち，ワイパーの動作サービスコールが呼び出されたワイパーが動作中に，別の制御サービスコールが呼び出されると，最初のサービスコールによる制御はキャンセルされ，新しい制御が開始される。

また，ワイパーは標準的なロック機能を持つLockableObjectである。ワイパーのロック機能はソフトウェアで実現される。

ワイパーが故障または凍結を検知した場合，ロックは解除され，故障中と凍結中は，ロックを取得することができない。

### 4.28.2. 機能クラス

Wiperは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | パワーワイパーであり，APIにより動作させることができる |
| HasWasher | ウォッシャー機能を備えている |

### 4.28.3. リスククラス

Wiperは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskWipingOnTrip | 走行中にワイパーが作動し，走行中に視界を妨げるリスク |
| RiskWiperWearing | ワイパーが摩耗した状態での操作により，窓に傷がつき視界不良となるリスク |
| RiskWiperBatteryDrain | ワイパーの連続操作によりバッテリーが消費し,走行不能となるリスク |
| RiskWiperHighFrequency | ワイパーの高周期操作により，ワイパー及びモータの早期摩耗・故障リスク |

### 4.28.4. 構成情報

Wiperの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| placement | PlacementType | 車両側面でのワイパーの位置（例：フロント，リア，サイドなど） |
| capabilities | enumSet (WiperCapabilityType) | そのワイパーが持つ機能 |
| riskClasses | enumSet(WiperRiskType) | そのワイパーのリスククラスの集合 |
| mountedOn | RelatedObjectType | 設置対象物（ウィンドウガラスなど） |
| displayName | String | ユーザに表示する呼称を取得するためのキー |

上記は，YAML形式で記述した構成記述ファイル中に記述する形式である。構成情報を参照するサービスコールは，この構成情報に対して，次の変換を行った構成情報を返す。

- mountedOnは，物理APIで定められたデータ型に変換して返す。

- displayNameKeyに対しては，それを元に現在のロケールに対応する呼称の文字列を取得し，その文字列を返す。

### 4.28.5. 状態

Wiperの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| mainStatus | WiperMainStatusType | 主状態 |
| lockStatus | LockStatusType | ロック状態 |
| frequencyLevel | UInt32 | 現在の動作速度レベル（1分間の往復回数,0〜100） |
| intervalLevel | UInt32 | 現在の一時停止時間（間欠動作による一時停止時間(秒),0〜30秒） |
| wiperWearLevel | UInt32 | ワイパーの摩耗レベル（0%：摩耗無し，100%：摩耗，要交換,0〜100） |
| washerStatus | WiperWasherStatusType | ウォッシャーの状態（OFF：停止，ON：作動中） |
| washerWaterLevel | UInt32 | ウォッシャー液の残量レベル（0%：残量無し，100%：満タン,0〜100） |

ワイパーの主状態（mainStatus）は，高頻度の状態遷移およびイベントの発生を防ぐため，「停止中」，「払拭動作中」，「故障中」，「凍結中」の状態をとる。

- **停止中（Stopped）**：デフォルト状態で，払拭動作が行われていない状態。この状態ではAPIからの制御が受け付けられ，ロック取得が可能である。
- **払拭動作中（Moving）**：startMoveによって設定された速度レベルで払拭ブレードが往復動作している状態。指定された速度レベルでの払拭制御が進行中である。APIからの新しい制御が呼び出された場合，後勝ちの原則により速度レベルが変更される。
- **故障中（Fault）**：ワイパーが故障を検知した状態。この状態ではAPIからの制御を受け付けず，ロック機能も取得できない。
- **凍結中（Frozen）**：ワイパーの払拭ブレードが凍結により動作できない状態を検知した状態。この状態ではAPIからの制御を受け付けず，ロック機能も取得できない。

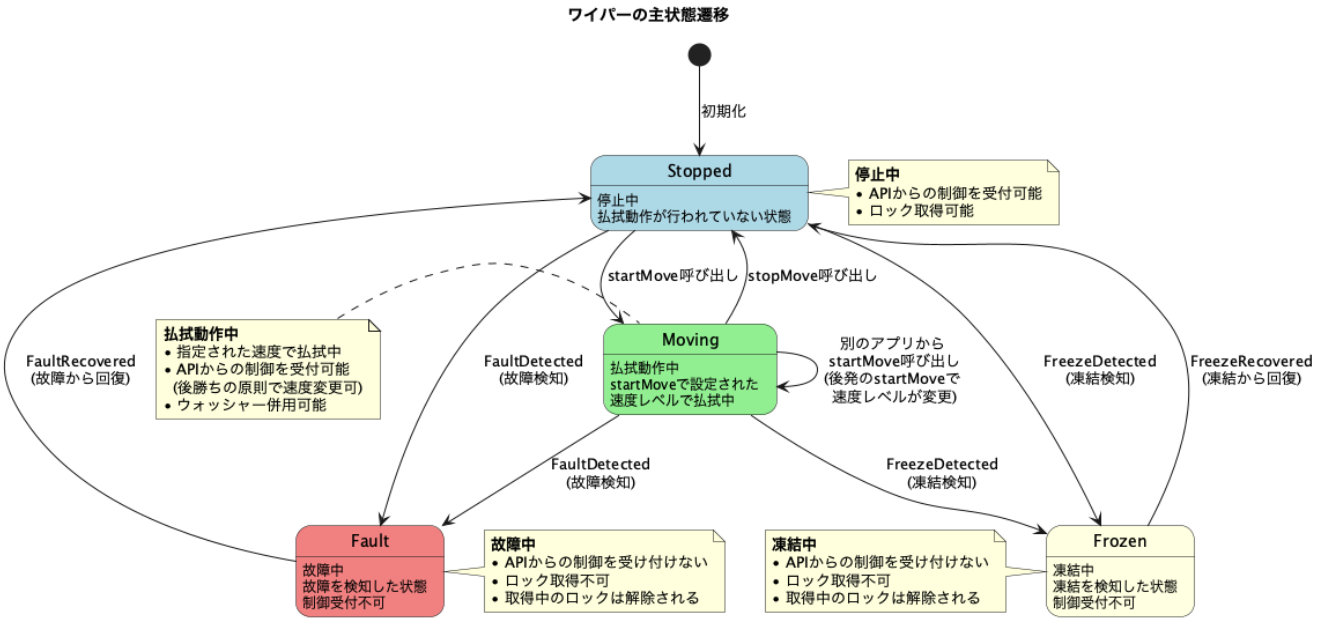



*Figure 12.* ワイパーの主状態の状態遷移

## 4.28.6. サービスコール一覧

Wiperに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのワイパーの構成情報の参照 |
| getStatus | ワイパーの状態の参照 |
| startMove | ワイパーの動作の開始 |
| stopMove | ワイパーの動作の停止 |
| startWasher | ウォッシャーの動作の開始 |
| stopWasher | ウォッシャーの動作の停止 |
| lock | ワイパーのロックの取得 |
| unlock | ワイパーのロックの解除 |
| notify | ワイパーイベントの通知要求 |
| unnotify | ワイパーイベントの通知停止 |
| getEvent | ワイパーイベントの取得 |

## 4.28.7. イベント

ワイパーに対するイベント種別には，MovableObjectの持つイベントとLockableObjectが持つイベントに加えて，故障検知と凍結検知がある。

Wiperに対するイベント種別は以下の通り。

| | |
|---|---|
| ControlledBySelf | 自アプリケーションが制御 |
| ControlledByOther | 他のアプリケーションが制御 |
| TargetReached | 指定された目標位置に到達して停止，または，移動停止が完了 |

| | |
|---|---|
| FaultDetected | 故障検知 |
| FaultRecovered | 故障から回復 |
| FreezeDetected | 凍結検知 |
| FreezeRecovered | 凍結から回復 |
| OwnLockRevoked | 自アプリケーションのロックの強制解除 |
| OtherLockReleased | 他のアプリケーション等によるロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

この内，イベント種別がControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.28.8. データ型の定義

### 4.28.8.1. ワイパーの機能クラス

```
type WiperCapabilityType = enum {
    Movable,
    HasWasher
};
```

### 4.28.8.2. ワイパーに対するリスククラス

```
type WiperRiskType = enum {
    RiskWipingOnTrip,
    RiskWiperWearing,
    RiskWiperBatteryDrain,
    RiskWiperHighFrequency
};
```

### 4.28.8.3. ワイパーの構成情報（getConfigAllが返すデータ型）

```
type WiperConfigType = struct {
    instanceId : IdType,
    placement : PlacementType,
    capabilities : enumSet(WiperCapabilityType),
```

```
    riskClasses : enumSet(WiperRiskType),
    mountedOn : RelatedObjectType,
    displayName : String
};
```

#### 4.28.8.4. ワイパー状態

```
type WiperMainStatusType = enum {
    Stopped,    // 停止中。払拭動作が行われていない状態。
この状態ではAPIからの制御が受け付けられ，ロック取得が可能である。
    Moving, // 払拭動作中。startMoveによって設定された速度レベルで払拭ブレードが往復動作している状態。
指定された速度レベルでの払拭制御が進行中である。
    Fault,  // 故障中。ワイパーが故障を検知した状態。
この状態ではAPIからの制御を受け付けず，ロック機能も取得できない。
    Frozen  // 凍結中。ワイパーの払拭ブレードが凍結により動作できない状態を検知した状態。
この状態ではAPIからの制御を受け付けず，ロック機能も取得できない。
};
```

```
type WiperWasherStatusType = enum {
    False,  // 停止
    True    // 作動中
};
```

```
type WiperStatusType = struct {
    mainStatus : WiperMainStatusType,
    lockStatus : LockStatusType,
    frequencyLevel : UInt32,
    intervalLevel : UInt32,
    wiperWearLevel : UInt32,
    washerStatus : WiperWasherStatusType,
    washerWaterLevel : UInt32
};
```

ワイパーの状態情報

#### 4.28.8.5. ワイパーイベント

```
type WiperEventKindType = enum {
    ControlledBySelf,
    ControlledByOther,
    TargetReached,
    FaultDetected,
    FaultRecovered,
    FreezeDetected,
```

```
    FreezeRecovered,
    OwnLockRevoked,
    OtherLockReleased,
    EventOverflow,
    ConfigChanged
};
```

```
type WiperEventType = struct {
    instanceId : IdType,
    eventInfo : variant(WiperEventKindType) {
        ControlledByOther {
            sourceApplication : ApplicationIdType
        },
        EventOverflow {
            noLostEvents : UInt32
        }
    }
};
```

## 4.28.9. サービスコールの詳細規定

### getConfigAll

すべてのワイパーの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | WiperConfigType[] | すべてのワイパーに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのワイパーに関する構成情報を返す。

### getStatus

ワイパーの状態の参照

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | WiperStatusType | 対象ワイパーの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のワイパーに関するすべての状態を返す。

## startMove

ワイパーの動作の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |
| freqencyLevel | UInt32 | 動作速度レベル（1分間の往復回数,0～100） |
| intervalLevel | UInt32? | 一時停止時間（間欠動作による一時停止時間(秒),0～30秒） |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | freqencyLevel，intervalLevel，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のワイパーが故障中 |
| E_OBJECT_STATUS | 操作対象のワイパーが凍結中 |
| E_OBJECT_LOCKED | 操作対象のワイパーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

動作速度レベルをfreqencyLevelに設定し，intervalLevelを指定した場合には間欠動作モードで一時停止時間をintervalLevelに設定して，操作対象のワイパーの動作を開始する。

intervalLevelを指定しない場合には，連続動作モードで動作する。

リスククラスRiskWipingOnTrip，RiskWiperWearing，RiskWiperBatteryDrainに対応できていない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopMove

ワイパーの動作の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のワイパーが故障中 |
| E_OBJECT_STATUS | 操作対象のワイパーが凍結中 |
| E_OBJECT_LOCKED | 操作対象のワイパーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |

**【機能】**

操作対象のワイパーの動作停止を要求する。

## startWasher

ウォッシャーの動作の開始

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |

| | |
|---|---|
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | 操作対象のワイパーが故障中 |
| E_OBJECT_STATUS | 操作対象のワイパーに十分なウォッシャー液がない |
| E_OBJECT_LOCKED | 操作対象のワイパーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_NOT_SUPPORTED | ウォッシャー機能を持たない |

**【機能】**

操作対象のワイパーのウォッシャー機能を動作させる。

## stopWasher

ウォッシャーの動作の停止

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | 操作対象のワイパーが，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_NOT_SUPPORTED | ウォッシャー機能を持たない |

**【機能】**

操作対象のワイパーのウォッシャー機能の動作停止を要求する。

## lock

ワイパーのロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | ロック対象のワイパーが，priorityと同じかそれより高い優先度でロックされている |
| E_OBJECT_FAULT | ロック対象のワイパーが故障中 |
| E_OBJECT_STATUS | ロック対象のワイパーが凍結中 |

**【機能】**

ロック対象のワイパーを，指定した操作優先度でロックする。

## unlock

ワイパーのロックの解除

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のワイパーのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが，操作対象のワイパーをロックしていない |

**【機能】**

ロック解除対象のワイパーを，ロック解除する。

## notify

ワイパーイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のワイパーのID |
| eventFilter | enumSet (WiperEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のワイパーで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

ワイパーイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

## getEvent

ワイパーイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | WiperEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。

イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.29. Vehicle.Infotainment（略称：Infotainment）

＜TBD＞

# 4.30. Vehicle.HMI（略称：HMI）

＜TBD＞

# 4.31. Vehicle.HMI.Display（略称：Display）

## 4.31.1. 位置付けと機能

Vehicle.HMI.Display (略称: Display) は，車両内に搭載された物理的なディスプレイを表すマルチインスタンスオブジェクトである。

Displayは、液晶ディスプレイ(LCD)，ヘッドアップディスプレイ(HUD)，電子サイドミラー等，様々な種類の表示装置を抽象化する。 DisplayはLockableObjectの機能を持つ。

### 4.31.1.1. Displayの位置付けと性質

Displayは，車内に向けて情報を表示する物理的な画面である。各Displayは固有のIDを持ち，その種別，視認可能な人物，物理サイズ，解像度などの構成情報を持つ。

Displayはほぼ平面であると仮定する。ディスプレイの形状はBoundingBox(外接矩形)のサイズで表現する。

Displayは一つ以上のViewを含むことができる。アプリケーションはDisplayに直接描画するのではなく，Display上に定義されたViewを通じて描画を行う。

フリップダウンディスプレイのように，視認性が状態によって変化するDisplayも存在する。

## 4.31.2. 機能クラス

Displayは，以下の機能クラスを持つ。

| | |
|---|---|
| Movable | Displayの位置を変えることができる(例:可動式モニタ) |
| Openable | Displayを開閉できる(例:フリップダウンモニタ) |

### 4.31.3. リスククラス

Displayは，以下のリスククラスを持つ。

| | |
|---|---|
| None | None |

### 4.31.4. 構成情報

Displayの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | ディスプレイのID |
| kind | DisplayKindType | ディスプレイの種別 |
| displayName | String | ディスプレイの名称 |
| visibleTo | enumSet(VisibleToType) | このディスプレイを視認できる人物の集合 |
| boundingBoxWidth | Float | ディスプレイのBoundingBoxの幅(単位:メートル) |
| boundingBoxHeight | Float | ディスプレイのBoundingBoxの高さ(単位:メートル) |
| shape | DisplayShapeType | ディスプレイの形状 |
| resolutionX | Uint32 | 水平方向の解像度(単位:px) |
| resolutionY | Uint32 | 垂直方向の解像度(単位:px) |
| refreshRate | Uint32 | リフレッシュレート(単位:Hz) |
| displayTechnology | DisplayTechnologyType | ディスプレイの表示技術の種別 |
| capabilities | enumSet (DisplayCapabilityType) | このディスプレイが持つ機能クラスの集合 |

### 4.31.5. 状態

Displayの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| availability | DisplayAvailabilityType | ディスプレイの使用可否状態 |

### 4.31.6. サービスコール一覧

Displayに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのディスプレイの構成情報の取得 |

| | |
|---|---|
| getStatus | ディスプレイの状態の取得 |
| notify | ディスプレイイベントの通知要求 |
| unnotify | ディスプレイイベントの通知停止 |
| getEvent | ディスプレイイベントの取得 |

## 4.31.7. イベント

Displayに対するイベント種別

Displayに対するイベント種別は以下の通り。

| | |
|---|---|
| AvailabilityChanged | ディスプレイの使用可否状態の変化 |

## 4.31.8. データ型の定義

### 4.31.8.1. ディスプレイの種別

```
type DisplayKindType = enum {
    CenterDisplay,  // センターディスプレイ
    InstrumentPanel,    // インパネ(ドライバーの目の前)
    HUD,    // ヘッドアップディスプレイ
    SideMirror, // サイドミラー(電子ミラー)
    RearMirror, // リアミラー
    RearMonitor,    // 後席モニタ
    PassengerDisplay,   // 助手席ディスプレイ
    Other   // その他
};
```

### 4.31.8.2. ディスプレイの形状

```
type DisplayShapeType = enum {
    Rectangle,  // 矩形
    Circle, // 円形
    Other   // その他
};
```

### 4.31.8.3. ディスプレイの表示技術

```
type DisplayTechnologyType = enum {
    LCD,    // 液晶ディスプレイ
    HUD // ヘッドアップディスプレイ(投影型)
};
```

#### 4.31.8.4. ディスプレイの機能クラス

```
type DisplayCapabilityType = enum {
    Movable,
    Openable
};
```

#### 4.31.8.5. ディスプレイを視認できる人物の種別

```
type VisibleToType = enum {
    Driver, // 運転者
    FrontPassenger, // 助手席乗員
    RearLeft,   // 後部座席左側乗員
    RearRight,  // 後部座席右側乗員
    Outside // 車外の人(今後の課題)
};
```

#### 4.31.8.6. ディスプレイの使用可否状態

```
type DisplayAvailabilityType = enum {
    Available,  // 使用可能
    Unavailable // 使用不可(故障等)
};
```

#### 4.31.8.7. ディスプレイの構成情報

```
type DisplayConfigType = struct {
    instanceId : IdType,
    kind : DisplayKindType,
    displayName : String,
    visibleTo : enumSet(VisibleToType),
    boundingBoxWidth : Float,
    boundingBoxHeight : Float,
    shape : DisplayShapeType,
    resolutionX : Uint32,
    resolutionY : Uint32,
    refreshRate : Uint32,
    displayTechnology : DisplayTechnologyType,
    capabilities : enumSet(DisplayCapabilityType)
};
```

#### 4.31.8.8. ディスプレイの状態

```
type DisplayStatusType = struct {
```

```
    availability : DisplayAvailabilityType
};
```

## 4.31.9. サービスコールの詳細規定

### getConfigAll

すべてのディスプレイの構成情報の取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| configs | DisplayConfigType[] | 車内のすべてのディスプレイの構成情報 |

**【エラーコード】**

なし

**【機能】**

車内のすべてのディスプレイの構成情報を返す。

### getStatus

ディスプレイの状態の取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 対象ディスプレイのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | DisplayStatusType | 対象ディスプレイの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

指定したディスプレイの使用可否状態を返す。

### notify

ディスプレイイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のディスプレイのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

指定したディスプレイで発生したイベントの通知を要求する。instanceIdを省略した場合，すべてのインスタンスのイベントを通知する。

## unnotify

ディスプレイイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。

## getEvent

ディスプレイイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | DisplayEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。

# 4.32. Vehicle.HMI.View（略称：View）

## 4.32.1. 位置付けと機能

Vehicle.HMI.View(略称:View)は，物理ディスプレイ(Vehicle.HMI.Display)上に定義された仮想的な描画領域を表すマルチインスタンスオブジェクトである。

アプリケーションはViewを通じてGraphicsObjectを描画することで，ディスプレイに映像・UI等を表示する。 ViewはLockableObjectの機能を持つ。

### 4.32.1.1. Viewの位置付けと性質

ViewはDisplayのBoundingBox左上を原点とするピクセル座標系で位置と大きさを定義する。Viewは矩形であると仮定する。

各ViewはひとつのDisplayに属し，複数のアプリケーションが同じViewを利用しようとした場合，優先度に基づく使用権(ロック)の仕組みにより排他制御が行われる。

アプリケーションがViewに描画するには，まずlock()を呼び出して使用権を取得し，その後attach()でGraphicsObjectを指定する。

矩形以外の形状のViewへの拡張は今後の課題である。

### 4.32.1.2. Viewの機能クラスと状態監視

Viewは機能クラスとして FreezeDetection を持つことができる。FreezeDetection 機能クラスを持つ View は、当該 View 上での描画処理が所定の時間内に進行していない状態（フリーズ）を検知する仕組みを備える。

## 4.32.2. 機能クラス

Viewは，以下の機能クラスを持つ。

| | |
|---|---|
| FreezeDetection | フリーズ検出が可能 |

## 4.32.3. リスククラス

Viewは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskDistractionHud | ヘッドアップディスプレイと同等のドライバディストラクションのリスク（運転者が見るために，視線の移動がほとんど必要ない。運転中は，運転に直結する情報のみ表示して良い） |
| RiskDistractionIP | インパネ内のディスプレイと同等のドライバディストラクションのリスク（運転者が見るために，少しの視線移動が必要である。運転中は，運転に関連する情報のみ表示して良い） |
| RiskDistractionCid | センターディスプレイと同等のドライバディストラクションのリスク（運転者が見るために，大きい視線移動が必要である） |

## 4.32.4. 構成情報

Viewの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | ViewのID |
| displayId | IdType | このViewを含むDisplayのID |
| positionX | Uint32 | DisplayのBoundingBox左上からのX座標(単位:px) |
| positionY | Uint32 | DisplayのBoundingBox左上からのY座標(単位:px) |
| width | Uint32 | Viewの幅(単位:px) |
| height | Uint32 | Viewの高さ(単位:px) |
| visibleTo | enumSet(VisibleToType) | このViewを視認できる人物の集合 |
| riskClasses | enumSet(ViewRiskType) | このViewのリスククラスの集合 |

## 4.32.5. 状態

Viewの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| status | ViewStatusType | Viewの現在の状態 |

## 4.32.6. サービスコール一覧

Viewに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | すべてのViewの構成情報の取得 |
| getStatus | Viewの状態の取得 |
| lock | Viewの使用権の確保 |
| unlock | Viewの使用権の破棄 |
| attach | GraphicsObjectツリーのViewへのアタッチ |
| setBackground | 背景テクスチャの指定 |
| notify | Viewイベントの通知要求 |
| unnotify | Viewイベントの通知停止 |

| | |
|---|---|
| getEvent | Viewイベントの取得 |
| detach | GraphicsObjectツリーのViewからのデタッチ |
| setVariableParameter | I/O変数への値の設定 |
| getVariableParameter | I/O変数の現在値の取得 |

## 4.32.7. イベント

Viewに対するイベント種別には，状態変化がある。

Viewに対するイベント種別は以下の通り。

| | |
|---|---|
| StatusChanged | Viewの状態変化 |

## 4.32.8. データ型の定義

### 4.32.8.1. Viewのリスククラス

```
type ViewRiskType = enum {
    RiskDistractionHud,
    RiskDistractionIP,
    RiskDistractionCid
};
```

### 4.32.8.2. Viewの状態種別

```
type ViewStatusType = enum {
    UNAVAILABLE,    // 警告またはDisplayがオフになった等の理由によりアプリが使用不可
    AVAILABLE,  // 使用可能
    IN_USE, // アプリが使用中
    FAULT   // 何等かの理由でViewが正常に表示できない
};
```

### 4.32.8.3. テクスチャのフィットタイプ

```
type FitType = enum {
    FILL,   // Viewのサイズに合うようテクスチャを引き延ばす
    CONTAIN,    // アスペクト比を維持したままテクスチャがViewに収まるよう拡大縮小する
    COVER   // アスペクト比を維持したままテクスチャがViewを埋めるよう拡大縮小する
};
```

### 4.32.8.4. Viewの構成情報

```
type ViewConfigType = struct {
```

```
    instanceId : IdType,
    displayId : IdType,
    positionX : Uint32,
    positionY : Uint32,
    width : Uint32,
    height : Uint32,
    visibleTo : enumSet(VisibleToType),
    riskClasses : enumSet(ViewRiskType)
};
```

#### 4.32.8.5. Viewの状態

```
type ViewStatusDataType = struct {
    status : ViewStatusType
};
```

#### 4.32.8.6. GraphicsObject共通型

```
type Vec2 = struct {
    x : Float32,
    y : Float32
};
```

```
type Vec3 = struct {
    x : Float32,
    y : Float32,
    z : Float32
};
```

```
type Color = struct {
    r : Float32,
    g : Float32,
    b : Float32,
    a : Float32
};
```

```
type UvArea = struct {
    origin : Vec2,
    size : Vec2
};
```

#### 4.32.8.7. I/O変数の値型

```
type IoVariantKind = enum {
    Sint32, // 32ビット符号付き整数
    Float32,    // 32ビット浮動小数点数
    String, // 文字列
    Bool,   // 真偽値
    Color,  // RGBAカラー
    Vec2,   // 2次元ベクトル
    Vec3,   // 3次元ベクトル
    UvArea  // UV座標系矩形領域
};
```

```
type IoVariant = variant(IoVariantKind) {
    Sint32 {
        value : Int32
    },
    Float32 {
        value : Float32
    },
    String {
        value : String
    },
    Bool {
        value : Bool
    },
    Color {
        value : Color
    },
    Vec2 {
        value : Vec2
    },
    Vec3 {
        value : Vec3
    },
    UvArea {
        value : UvArea
    }
};
```

#### 4.32.8.8. Bindable型（直接値またはI/O変数バインド）

```
type IoVariableId = Uint32; // I/O変数のID
```

```
type BindableKind = enum {
    Value,  // 直接値
```

```
    Bind    // I/O変数へのバインド参照
};
```

```
type BindableBool = variant(BindableKind) {
    Value {
        value : Bool
    },
    Bind {
        bindId : IoVariableId
    }
};
```

```
type BindableFloat32 = variant(BindableKind) {
    Value {
        value : Float32
    },
    Bind {
        bindId : IoVariableId
    }
};
```

```
type BindableString = variant(BindableKind) {
    Value {
        value : String
    },
    Bind {
        bindId : IoVariableId
    }
};
```

```
type BindableColor = variant(BindableKind) {
    Value {
        value : Color
    },
    Bind {
        bindId : IoVariableId
    }
};
```

```
type BindableVec2 = variant(BindableKind) {
    Value {
        value : Vec2
    },
    Bind {
```

```
        bindId : IoVariableId
    }
};
```

```
type BindableVec3 = variant(BindableKind) {
    Value {
        value : Vec3
    },
    Bind {
        bindId : IoVariableId
    }
};
```

```
type BindableUvArea = variant(BindableKind) {
    Value {
        value : UvArea
    },
    Bind {
        bindId : IoVariableId
    }
};
```

### 4.32.8.9. GraphicsObjectパラメータ型

```
type GraphicsObjectParams = struct {
    visible : BindableBool?,    // デフォルト: true
    anchor_point : BindableVec2?,   // 回転・拡縮の中心。デフォルト: 描画サイズの中心
    position : BindableVec3?,   // anchor_pointの親座標系での配置位置。デフォルト: {x:
0, y:0, z:0}
    area : Vec2?,   // BoundingBoxサイズのオーバーライド。デフォルト: 描画サイズから自
動算出
    rotation : BindableFloat32?,    // 度数法。デフォルト: 0
    scale : BindableVec2?   // デフォルト: {x:1.0 y:1.0}
};
```

```
type TextureParams = struct {
    base : GraphicsObjectParams?,
    texture_id : BindableSint32?,   // 未設定時は何も描画しない
    color : BindableColor?, // 乗算ブレンド。デフォルト: {r:1,g:1,b:1,a:1}
    uv_area : BindableUvArea?   // デフォルト: テクスチャ全体
};
```

```
type CircleParams = struct {
    base : GraphicsObjectParams?,
```

```
    color : BindableColor?, // デフォルト: {r:1,g:1,b:1,a:1}
    radius : BindableFloat32?   // デフォルト: 50
};
```

```
type RectangleParams = struct {
    base : GraphicsObjectParams?,
    color : BindableColor?, // デフォルト: {r:1,g:1,b:1,a:1}
    width : BindableFloat32?,   // デフォルト: 100
    height : BindableFloat32?   // デフォルト: 100
};
```

```
type TriangleParams = struct {
    base : GraphicsObjectParams?,
    color : BindableColor?, // デフォルト: {r:1,g:1,b:1,a:1}
    side_length : BindableFloat32?  // 正三角形の1辺。デフォルト: 100
};
```

```
type LineParams = struct {
    base : GraphicsObjectParams?,
    color : BindableColor?, // デフォルト: {r:1,g:1,b:1,a:1}
    length : BindableFloat32?,  // 線の方向はrotationで指定。デフォルト: 100
    width : BindableFloat32?    // デフォルト: 1
};
```

```
type TextParams = struct {
    base : GraphicsObjectParams?,
    text_string : BindableString?,  // デフォルト: ""
    color : BindableColor?, // デフォルト: {r:1,g:1,b:1,a:1}
    font_size : BindableFloat32?,   // ピクセル単位。デフォルト: 16
    font_id : BindableSint32?   // 未設定時はシステムデフォルト
};
```

```
type ArcParams = struct {
    base : GraphicsObjectParams?,
    radius : BindableFloat32?,  // デフォルト: 50
    start_angle : BindableFloat32?, // 0度 = 12時方向、時計回り。デフォルト: 0
    end_angle : BindableFloat32?,   // start_angle < end_angle。デフォルト: 360
    line_width : BindableFloat32?,  // デフォルト: 1
    color : BindableColor?  // デフォルト: {r:1g:1b:1a:1}
};
```

```
type VideoPlayState = enum {
    Playing,    // 再生中
```

```
    Stopped // 先頭に戻る
};
```

```
type VideoParams = struct {
    base : GraphicsObjectParams?,
    video_id : BindableSint32?, // 未設定時は何も描画しない
    play_state : VideoPlayState?    // デフォルト: Stopped
};
```

```
type Alignment = enum {
    UpperLeft,  // 左上
    UpperCenter,    // 上中央
    UpperRight, // 右上
    MiddleLeft, // 左中央
    MiddleCenter,   // 中央
    MiddleRight,    // 右中央
    LowerLeft,  // 左下
    LowerCenter,    // 下中央
    LowerRight  // 右下
};
```

```
type LayoutParams = struct {
    base : GraphicsObjectParams?,
    alignment : Alignment?, // デフォルト: UpperLeft
    padding : BindableVec2? // x=左右、y=上下（片側分）。デフォルト: {x:0 y:0}
};
```

```
type DirectionalLayoutParams = struct {
    base : GraphicsObjectParams?,
    alignment : Alignment?, // デフォルト: UpperLeft
    padding : BindableVec2?,    // x=左右、y=上下（片側分）。デフォルト: {x:0, y:0}
    spacing : BindableFloat32?  // 子要素間の間隔。デフォルト: 0
};
```

#### 4.32.8.10. I/O変数定義

```
type IoVariableDef = struct {
    id : IoVariableId,  // I/O変数のID
    type : IoVariantKind,   // I/O変数の値型
    default_value : IoVariant?  // 初期値。省略時はゼロ値
};
```

### 4.32.8.11. GraphicsObjectツリー構造型

```
type GraphicsObjectKind = enum {
    Texture,    // テクスチャ画像
    Arc,    // 円弧
    Circle, // 円
    Rectangle,  // 矩形
    Triangle,   // 正三角形
    Line,   // 直線
    Text,   // テキスト
    Video,  // ビデオ
    Layout, // レイアウト
    VerticalLayout, // 縦方向レイアウト
    HorizontalLayout    // 横方向レイアウト
};
```

```
type GraphicsObjectNode = variant(GraphicsObjectKind) {
    Texture {
        params : TextureParams?,
        children : GraphicsObjectNode[]?
    },
    Arc {
        params : ArcParams?,
        children : GraphicsObjectNode[]?
    },
    Circle {
        params : CircleParams?,
        children : GraphicsObjectNode[]?
    },
    Rectangle {
        params : RectangleParams?,
        children : GraphicsObjectNode[]?
    },
    Triangle {
        params : TriangleParams?,
        children : GraphicsObjectNode[]?
    },
    Line {
        params : LineParams?,
        children : GraphicsObjectNode[]?
    },
    Text {
        params : TextParams?,
        children : GraphicsObjectNode[]?
    },
    Video {
        params : VideoParams?,
        children : GraphicsObjectNode[]?
    },
```

```
    Layout {
        params : LayoutParams?,
        children : GraphicsObjectNode[]?
    },
    VerticalLayout {
        params : DirectionalLayoutParams?,
        children : GraphicsObjectNode[]?
    },
    HorizontalLayout {
        params : DirectionalLayoutParams?,
        children : GraphicsObjectNode[]?
    }
};
```

```
type GraphicsObjectTree = struct {
    io_variables : IoVariableDef[],
    root : GraphicsObjectNode
};
```

### 4.32.8.12. GraphicsObjectエラーコードと結果型

```
type GraphicsObjectErrorCode = enum {
    E_INVALID_ID,   // IDが不正
    E_INVALID_HANDLE,   // ハンドルが不正
    E_INVALID_PARAMETER,    // パラメータが不正
    E_OBJECT_STATUS // attach()時に同一view_idに既存ツリーが存在する
};
```

```
type TreeHandle = IdType;   // attach()が返すツリーハンドル
```

```
type GraphicsObjectResultKind = enum {
    Ok, // 正常終了
    Err // エラー終了
};
```

```
type AttachResult = variant(GraphicsObjectResultKind) {
    Ok {
        tree_handle : TreeHandle
    },
    Err {
        GraphicsObjectErrorCode : GraphicsObjectErrorCode
    }
};
```

```
type VoidResult = variant(GraphicsObjectResultKind) {
    Ok,
    Err {
        GraphicsObjectErrorCode : GraphicsObjectErrorCode
    }
};
```

```
type GetParameterResult = variant(GraphicsObjectResultKind) {
    Ok {
        value : IoVariant
    },
    Err {
        GraphicsObjectErrorCode : GraphicsObjectErrorCode
    }
};
```

## 4.32.9. サービスコールの詳細規定

### getConfigAll

すべてのViewの構成情報の取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| configs | ViewConfigType[] | 車内のすべてのViewの構成情報 |

**【エラーコード】**

なし

**【機能】**

車内のすべてのViewの構成情報を返す。

### getStatus

Viewの状態の取得

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 対象ViewのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | ViewStatusType | Viewの現在の状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

指定したViewの現在の状態を返す。

## lock

Viewの使用権の確保

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 対象ViewのID |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_LOCKED | 自分より優先度の高いアプリがすでに使用権を確保している |
| E_OBJECT_STATUS | ViewがUNAVAILABLEまたはFAULT状態 |

**【機能】**

Viewの使用権の確保を試みる。

自分より優先度の高いアプリがすでに確保していたらlockに失敗する。

逆に，あるアプリが一度lockしたViewであっても，別アプリがより高い優先度でlockしたら使用権を奪われる。

例:ゲーム(優先度:3)再生中に，車両接近通報装置(優先度:4)がlockして，ゲーム画面が車両接近通報装置に切り替わる。

## unlock

Viewの使用権の破棄

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 対象ViewのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_OBJECT_STATUS | 自アプリケーションが対象Viewをロックしていないため Unlockできない |

**【機能】**

Viewの使用権を破棄する。

## attach

GraphicsObjectツリーのViewへのアタッチ

**【パラメータ】**

| | | |
|---|---|---|
| view_id | IdType | 対象ViewのID |
| tree | GraphicsObjectTree | アタッチするGraphicsObjectツリー |

**【リターンパラメータ】**

| | | |
|---|---|---|
| result | AttachResult | 成功時はtree_handle、失敗時はエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_OBJECT_STATUS | 同一view_idに既存ツリーが存在する |

**【機能】**

GraphicsObjectツリーをViewにアタッチする。返却されるtreeHandleを使ってI/O変数の操作を行う。同一view_idに既にツリーがアタッチされている場合はE_OBJECT_STATUSを返す。

## setBackground

背景テクスチャの指定

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 対象ViewのID |
| textureId | IdType | テクスチャID |
| fitType | FitType | フィットタイプ |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdまたはtextureIdが有効範囲外 |
| E_OBJECT_STATUS | 対象ViewがLock済みでない |

**【機能】**

ViewにテクスチャをfitTypeで指定した方法で背景として設定する。

## notify

Viewイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | イベント通知対象のViewのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

指定したViewで発生したイベントの通知を要求する。instanceIdを省略した場合，すべてのインスタンスのイベントを通知する。

## unnotify

Viewイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。

## getEvent

Viewイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | ViewEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。

## detach

GraphicsObjectツリーのViewからのデタッチ

**【パラメータ】**

| | | |
|---|---|---|
| tree_handle | TreeHandle | 対象ツリーのハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| result | VoidResult | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | tree_handleが不正 |

**【機能】**

Viewから描画ツリーを取り除く。

## setVariableParameter

I/O変数への値の設定

**【パラメータ】**

| | | |
|---|---|---|
| tree_handle | TreeHandle | 対象ツリーのハンドル |
| io_variable_id | IoVariableId | 設定対象のI/O変数ID |
| value | IoVariant | 設定する値。IoVariableDef.typeと一致させること |

**【リターンパラメータ】**

| | | |
|---|---|---|
| result | VoidResult | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | tree_handleが不正 |
| E_INVALID_ID | io_variable_idが存在しない |
| E_INVALID_PARAMETER | valueのkindがI/O変数の型と不一致 |

**【機能】**

I/O変数に値を設定する。バインドされているすべてのparamに即時反映される。

### getVariableParameter

I/O変数の現在値の取得

**【パラメータ】**

| | | |
|---|---|---|
| tree_handle | TreeHandle | 対象ツリーのハンドル |
| io_variable_id | IoVariableId | 取得対象のI/O変数ID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| result | GetParameterResult | 成功時はIoVariant値、失敗時はエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | tree_handleが不正 |
| E_INVALID_ID | io_variable_idが存在しない |

**【機能】**

I/O変数の現在値を取得する。

## 4.33. Vehicle.HMI.Input.Keyboard（略称：Keyboard）

## 4.33.1. 位置付けと機能

Vehicle.HMI.Input.Keyboard（略称: Keyboard）は，車載キーボード(物理ボタン含む)からの入力を受け取る機能を提供するシングルインスタンスオブジェクトである。

### 4.33.1.1. Keyboardの位置付けと性質

Keyboardは，キー押下・キーリリースイベントの通知，入力文字の取得などの機能を提供する。物理キーボードおよびステアリングスイッチなどのハードウェアボタンからの入力を統一的に扱う。

## 4.33.2. 機能クラス

## 4.33.3. リスククラス

## 4.33.4. 構成情報

## 4.33.5. 状態

## 4.33.6. サービスコール一覧

Keyboardに対するサービスコールは以下の通り。

| | |
|---|---|
| startKeyboardInput | 入力の開始(キーボードを表示) |
| completeKeyboardInput | 入力の完了(キーボードを非表示) |
| onTextInput | 文字入力を検知 |

## 4.33.7. イベント

## 4.33.8. データ型の定義

## 4.33.9. サービスコールの詳細規定

### startKeyboardInput

入力の開始(キーボードを表示)

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

NOTE ソフトウェアキーボードを想定する。

### completeKeyboardInput

入力の完了(キーボードを非表示)

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

### onTextInput

文字入力を検知

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| inputChar | String | 入力された文字 |

**【エラーコード】**

なし

**【機能】**

# 4.34. Vehicle.HMI.Input.Mouse（略称：Mouse）

## 4.34.1. 位置付けと機能

Vehicle.HMI.Input.Mouse（略称: Mouse）は，ポインティングデバイス(マウス，トラックパッド等)からの入力を受け取る機能を提供するシングルインスタンスオブジェクトである。

### 4.34.1.1. Mouseの位置付けと性質

Mouseは，ポインタ座標の取得，ボタンクリックイベントの通知，スクロール操作の検出などの機能を提

供する。

本オブジェクトは大部分が仕様未定義である。今後、詳細な定義を行う。

### 4.34.2. 機能クラス

### 4.34.3. リスククラス

### 4.34.4. 構成情報

### 4.34.5. 状態

### 4.34.6. サービスコール一覧

Mouseに対するサービスコールは以下の通り。

### 4.34.7. イベント

### 4.34.8. データ型の定義

### 4.34.9. サービスコールの詳細規定

## 4.35. Vehicle.HMI.Input.Touch（略称：Touch）

### 4.35.1. 位置付けと機能

Vehicle.HMI.Input.Touch (略称: Touch) は，タッチパネルからのタッチ入力を受け取る機能を提供するシングルインスタンスオブジェクトである。

#### 4.35.1.1. Touchの位置付けと性質

Touchは，タッチ開始・移動・終了イベントの通知，マルチタッチのサポート，タッチ座標の取得などの機能を提供する。タッチスクリーンを持つDisplayに対応した操作インタフェースを提供する。

### 4.35.2. 機能クラス

### 4.35.3. リスククラス

### 4.35.4. 構成情報

### 4.35.5. 状態

## 4.35.6. サービスコール一覧

Touchに対するサービスコールは以下の通り。

| | |
|---|---|
| getTouchNum | タッチ点数を取得 |
| getTouchId | タッチ点のIDを取得 |
| getPosition | タッチ点の位置を取得 |
| isDown | タッチ点が押されているかどうかを判定 |
| isPressed | タッチ点がそのフレームで押されたかどうかを判定 |
| isReleased | タッチ点がそのフレームで放されたかどうかを判定 |
| isTapped | タッチ点がそのフレームでタップされたかどうかを判定 |
| isDoubleTapped | タッチ点がそのフレームでダブルタップされたかどうかを判定 |
| isLongTapped | タッチ点がそのフレームでロングタップされたかどうかを判定 |
| isFlicked | タッチ点がそのフレームでフリックされたかどうかを判定 |
| getFlickPosition | フリックの始点位置を取得 |
| getFlickDirection | フリックの方向を取得 |
| invalidate | 入力の無効化 |

## 4.35.7. イベント

## 4.35.8. データ型の定義

## 4.35.9. サービスコールの詳細規定

### getTouchNum

タッチ点数を取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| touchNum | Uint32 | タッチ点数 |

**【エラーコード】**

なし

**【機能】**

## getTouchId

タッチ点のIDを取得

**【パラメータ】**

| | | |
|---|---|---|
| index | Uint32 | タッチ点の認識順番 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| touchId | Uint32 | タッチ点のID |

**【エラーコード】**

なし

**【機能】**

## getPosition

タッチ点の位置を取得

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| position | Vector2 | タッチ点の位置(2次元ベクトル) |

**【エラーコード】**

なし

**【機能】**

## isDown

タッチ点が押されているかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点が押されているかどうか |

**【エラーコード】**

なし

**【機能】**

## isPressed

タッチ点がそのフレームで押されたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームで押されたかどうか |

**【エラーコード】**

なし

**【機能】**

## isReleased

タッチ点がそのフレームで放されたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームで放されたかどうか |

**【エラーコード】**

なし

**【機能】**

## isTapped

タッチ点がそのフレームでタップされたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームでタップされたかどうか |

**【エラーコード】**

なし

**【機能】**

## isDoubleTapped

タッチ点がそのフレームでダブルタップされたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームでダブルタップされたかどうか |

**【エラーコード】**

なし

**【機能】**

## isLongTapped

タッチ点がそのフレームでロングタップされたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームでロングタップされたかどうか |

**【エラーコード】**

なし

**【機能】**

## isFlicked

タッチ点がそのフレームでフリックされたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | タッチ点がそのフレームでフリックされたかどうか |

**【エラーコード】**

なし

**【機能】**

## getFlickPosition

フリックの始点位置を取得

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| position | Vector2 | フリックの始点位置(2次元ベクトル) |

**【エラーコード】**

なし

**【機能】**

## getFlickDirection

フリックの方向を取得

**【パラメータ】**

| | | |
|---|---|---|
| touchId | Uint32 | タッチ点のID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| direction | Vector2 | フリックの方向(2次元ベクトル) |

**【エラーコード】**

なし

**【機能】**

### invalidate

入力の無効化

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

この関数実行後から次のフレーム開始までの間、

タッチの入力値の取得を無効化する

# 4.36. Vehicle.HMI.Input.Joystick（略称：Joystick）

## 4.36.1. 位置付けと機能

Vehicle.HMI.Input.Joystick（略称: Joystick）は，ジョイスティック(方向スイッチ，ボタン)からの入力を受け取る機能を提供するシングルインスタンスオブジェクトである。

### 4.36.1.1. Joystickの位置付けと性質

Joystickは，方向入力(上下左右)，決定・戻るボタン，アナログ軸の値取得などの機能を提供する。センターコンソールのコントローラやステアリングスイッチなどの操作デバイスを統一的に扱う。

## 4.36.2. 機能クラス

## 4.36.3. リスククラス

## 4.36.4. 構成情報

## 4.36.5. 状態

## 4.36.6. サービスコール一覧

Joystickに対するサービスコールは以下の通り。

| | |
|---|---|
| isDown | ボタンが押されているかどうかを判定 |
| isPressed | ボタンがそのフレームで押されたかどうかを判定 |
| isReleased | ボタンがそのフレームで放されたかどうかを判定 |
| getAnalog | アナログ軸の入力値を取得 |
| invalidate | 入力の無効化 |

## 4.36.7. イベント

## 4.36.8. データ型の定義

### 4.36.8.1. 型定義

```
type JoystickKeyKind = enum {
    Up, // 上方向
    Down,   // 下方向
    Left,   // 左方向
    Right,  // 右方向
    Decide, // 決定
    Back    // 戻る
};
```

```
type JoystickAxisKind = enum {
    X,  // 左右方向の傾き量（左: -1.0 ～ 右: +1.0）
    Y   // 上下方向の傾き量（下: -1.0 ～ 上: +1.0）
};
```

## 4.36.9. サービスコールの詳細規定

### isDown

ボタンが押されているかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| keyKind | JoystickKeyKind | キーの種類 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | ボタンが押されているかどうか |

**【エラーコード】**

なし

**【機能】**

## isPressed

ボタンがそのフレームで押されたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| keyKind | JoystickKeyKind | キーの種類 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | ボタンがそのフレームで押されたかどうか |

**【エラーコード】**

なし

**【機能】**

## isReleased

ボタンがそのフレームで放されたかどうかを判定

**【パラメータ】**

| | | |
|---|---|---|
| keyKind | JoystickKeyKind | キーの種類 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| result | Bool | ボタンがそのフレームで放されたかどうか |

**【エラーコード】**

なし

**【機能】**

## getAnalog

アナログ軸の入力値を取得

**【パラメータ】**

| | | |
|---|---|---|
| axisKind | JoystickAxisKind | 軸の種類 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| analogValue | Float32 | 軸の入力値（-1.0 ～ +1.0） |

**【エラーコード】**

なし

**【機能】**

### invalidate

入力の無効化

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

この関数実行後から次のフレーム開始までの間、

ジョイスティックの入力値の取得を無効化する

# 4.37. Vehicle.HMI.Sound.Common.SoundEffect（略称：SoundEffect）

## 4.37.1. 位置付けと機能

Vehicle.HMI.Sound.Common.SoundEffect (略称: SoundEffect) は，システム共通の効果音の再生管理機能を提供するシングルインスタンスオブジェクトである。

### 4.37.1.1. SoundEffect(Common)の位置付けと性質

Vehicle.HMI.Sound.Common.SoundEffectは，操作フィードバック音など，複数のアプリケーションで共有される共通効果音の再生・停止機能を提供する。共通効果音はシステム定義のサウンドセットから選択して使用する。

## 4.37.2. 機能クラス

## 4.37.3. リスククラス

## 4.37.4. 構成情報

## 4.37.5. 状態

## 4.37.6. サービスコール一覧

SoundEffectに対するサービスコールは以下の通り。

| | |
|---|---|
| play | 共通サウンドの再生 |
| overwrite | 共通サウンドのデータを上書き |
| reset | 共通サウンドのデータをリセット |

## 4.37.7. イベント

## 4.37.8. データ型の定義

### 4.37.8.1. 型定義

```
type CommonSoundKind = enum {
    OK, // 操作確定音
    Cancel, // キャンセル音
    NG, // 操作不可音
    CursorMove, // カーソル移動音
    Alert,  // 警告音
    Open,   // 画面オープン音
    Close   // 画面クローズ音
};
```

## 4.37.9. サービスコールの詳細規定

### play

共通サウンドの再生

**【パラメータ】**

| | | |
|---|---|---|
| soundKind | CommonSoundKind | 共通サウンドの種類 |
| volume | Uint8 | 音量(0～255) |
| priority | Uint8 ([0,100]) | 優先順位(0～100) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

Sound.App.SoundEffect.playの「ビークルOSカーネルに実装する機能等」の欄を参照

## overwrite

共通サウンドのデータを上書き

**【パラメータ】**

| | | |
|---|---|---|
| soundKind | CommonSoundKind | 共通サウンドの種類 |
| soundId | Uint32 | 置き換え先の音源ID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

デフォルトで入っていない音源にはidを割り振り、idを指定して再生や置き換えなどを行う。

## reset

共通サウンドのデータをリセット

**【パラメータ】**

| | | |
|---|---|---|
| soundKind | CommonSoundKind | 共通サウンドの種類 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

# 4.38. Vehicle.HMI.Sound.App.SoundEffect（略称：SoundEffect）

## 4.38.1. 位置付けと機能

Vehicle.HMI.Sound.App.SoundEffect（略称: SoundEffect）は，アプリケーション固有の効果音の再生管理機能を提供するシングルインスタンスオブジェクトである。

### 4.38.1.1. SoundEffect(App)の位置付けと性質

Vehicle.HMI.Sound.App.SoundEffectは，アプリケーションが独自に定義した効果音の登録・再生・停止機能を提供する。アプリケーションはサウンドリソースを登録し，任意のタイミングで再生できる。

## 4.38.2. 機能クラス

## 4.38.3. リスククラス

## 4.38.4. 構成情報

## 4.38.5. 状態

## 4.38.6. サービスコール一覧

SoundEffectに対するサービスコールは以下の通り。

| | |
|---|---|
| getStatus | SoundEffectハンドルのステータスを取得 |
| getPlaying | 再生中のSoundEffectを取得 |
| getScheduled | スケジュールされたSoundEffectを取得 |
| play | SoundEffectを再生 |
| playLoop | SoundEffectをループ再生 |
| schedulePlay | SoundEffectの再生をスケジュールする |
| pause | SoundEffectの中断 |
| unPause | SoundEffectの再開 |
| stop | SoundEffectの停止 |
| cancelSchedule | スケジュールをキャンセル |

## 4.38.7. イベント

## 4.38.8. データ型の定義

## 4.38.9. サービスコールの詳細規定

### getStatus

SoundEffectハンドルのステータスを取得

**【パラメータ】**

| | | |
|---|---|---|
| soundEffectHandle | Uint32 | 操作対象のSoundEffectハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | enum([{'enumerator': 'PLAYING', 'description': '再生中'}, {'enumerator': 'WAITING', 'description': '再生待機中'}, {'enumerator': 'STOPPED', 'description': '再生終了'}, {'enumerator': 'PAUSED', 'description': 'ポーズ中'}]) | SoundEffectハンドルのステータス |

**【エラーコード】**

なし

**【機能】**


SoundEffectの再生中はPLAYING、

SchedulePlayによる再生待機中はWAITING、

再生が完了したもの、あるいはStopやCancelScheduleによって二度と再開されなくなったものはSTOPPED、

ポーズ中のものはPAUSEDとなる。

### getPlaying

再生中のSoundEffectを取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| handles | Uint32[3] | SoundEffectハンドルの配列(長さは最大3) |

**【エラーコード】**

なし

**【機能】**

## getScheduled

スケジュールされたSoundEffectを取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| handles | Uint32[] | SoundEffectハンドルの配列 |

**【エラーコード】**

なし

**【機能】**

## play

SoundEffectを再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |
| priority | Uint8 ([0,100]) | 優先順位(0〜100) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【エラーコード】**

なし

**【機能】**

Sound.CommonとSoundEffectと合わせた同時再生数の最大値は3つ。

4つ目を再生すると残り3つのうち最も優先順位が低いものが停止する。

## playLoop

SoundEffectをループ再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |
| priority | Uint8 ([0,100]) | 優先順位(0〜100) |
| loopCount | Uint16 ([1,256]) | ループの回数(最大256回) |
| loopInterval | Uint32 | ループの間隔(ミリ秒) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【エラーコード】**

なし

**【機能】**

## schedulePlay

SoundEffectの再生をスケジュールする

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |
| priority | Uint8 ([0,100]) | 優先順位(0〜100) |
| delay | Uint32 | 遅延させる時間(ミリ秒) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【エラーコード】**

なし

**【機能】**

引数で指定した時間が経過してから再生を行う。

## pause

SoundEffectの中断

**【パラメータ】**

| | | |
|---|---|---|
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

ハンドルで指定したSoundEffectをポーズする。

下記のUnPauseメソッドをポーズしたSoundEffectに対して呼ぶことで再生が再開される。

## unPause

SoundEffectの再開

**【パラメータ】**

| | | |
|---|---|---|
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

### stop

SoundEffectの停止

**【パラメータ】**

| | | |
|---|---|---|
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

ハンドルで指定したSoundEffectを停止する。

Pauseとは異なり、再開することはない。

### cancelSchedule

スケジュールをキャンセル

**【パラメータ】**

| | | |
|---|---|---|
| soundEffectHandle | Uint32 | SoundEffectハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

再生が予定されているSoundEffectに対してキャンセル処理を行う。

# 4.39. Vehicle.HMI.Sound.App.Music（略称：Music）

## 4.39.1. 位置付けと機能

Vehicle.HMI.Sound.App.Music (略称: Music) は，アプリケーションによる音楽・BGMの再生管理機能を提供するシングルインスタンスオブジェクトである。

#### 4.39.1.1. Musicの位置付けと性質

Musicは，音楽ファイルの再生，一時停止，停止，音量調整などの再生制御機能を提供する。複数のアプリケーションが同時に音楽を再生しようとした場合，優先度に基づいて制御される。

## 4.39.2. 機能クラス

## 4.39.3. リスククラス

Musicは，以下のリスククラスを持つ。

| | |
|---|---|
| T.B.D | T.B.D |

## 4.39.4. 構成情報

## 4.39.5. 状態

## 4.39.6. サービスコール一覧

Musicに対するサービスコールは以下の通り。

| | |
|---|---|
| play | Musicの再生 |
| pause | Musicの中断 |
| unPause | Musicの再開 |
| stop | Musicの停止 |
| fadeIn | Musicのフェードイン |
| fadeOut | Musicのフェードアウト |

## 4.39.7. イベント

## 4.39.8. データ型の定義

## 4.39.9. サービスコールの詳細規定

### play

Musicの再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

同時に再生できるMusicは1つのみであり、

2つ目を再生すると現在再生中のものが停止する。

常時ループ再生され、アプリの終了または中断で自動的に停止する。

## pause

Musicの中断

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

## unPause

Musicの再開

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

## stop

Musicの停止

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

## fadeIn

Musicのフェードイン

**【パラメータ】**

| | | |
|---|---|---|
| fadeInDuration | Uint32 | フェードインにかける時間(ミリ秒) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

## fadeOut

Musicのフェードアウト

**【パラメータ】**

| | | |
|---|---|---|
| fadeOutDuration | Uint32 | フェードアウトにかける時間(ミリ秒) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

【機能】

# 4.40. Vehicle.HMI.Sound.App.Voice（略称：Voice）

## 4.40.1. 位置付けと機能

Vehicle.HMI.Sound.App.Voice （略称: Voice） は，アプリケーションによる音声(テキスト読み上げ，ナレーション等)の再生管理機能を提供するシングルインスタンスオブジェクトである。

### 4.40.1.1. Voice(App)の位置付けと性質

Vehicle.HMI.Sound.App.Voiceは，音声ファイルの再生，テキスト音声合成(TTS)による読み上げ，再生状態の監視などの機能を提供する。ナビゲーション案内やアシスタント音声など，アプリが必要とする音声出力を管理する。

## 4.40.2. 機能クラス

## 4.40.3. リスククラス

## 4.40.4. 構成情報

## 4.40.5. 状態

## 4.40.6. サービスコール一覧

Voiceに対するサービスコールは以下の通り。

| | |
|---|---|
| getStatus | voiceハンドルの状態を取得 |
| getPlaying | 再生中のボイスを取得 |
| getWaiting | 待機中のボイスを取得 |
| play | ボイスの再生 |
| playText | テキスト読み上げによるボイスの再生 |
| playBlank | 無音の再生 |
| interrupt | ボイスの割り込み再生 |
| interruptText | テキスト読み上げによるボイスの割り込み再生 |
| stop | ボイスの停止 |
| pause | ボイスの中断 |
| unPause | ボイスの再開 |
| skip | 再生中のボイスをスキップ |

## 4.40.7. イベント

## 4.40.8. データ型の定義

### 4.40.8.1. 型定義

```
type VoiceInterruptBehavior = enum {
    PAUSE,  // 途中から再開
    STOP,   // 再吹鳴なし
    REPLAY  // 頭から再度再生
};
```

## 4.40.9. サービスコールの詳細規定

### getStatus

voiceハンドルの状態を取得

**【パラメータ】**

| | | |
|---|---|---|
| voiceHandle | Uint32 | voiceハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | enum([{'enumerator': 'PLAYING', 'description': '再生中'}, {'enumerator': 'WAITING', 'description': '再生待機中'}, {'enumerator': 'STOPPED', 'description': '再生終了'}, {'enumerator': 'PAUSED', 'description': 'ポーズ中'}]) | 状態 |

**【エラーコード】**

なし

**【機能】**

Voiceの再生中はPLAYING、

バッファに積まれている間や、中断時の挙動をREPLAYにしたことによる再生待ちはWAITING、

Stopメソッドや中断時の挙動をSTOPに設定したことによる再生終了はSTOPPED、

Pauseメソッドや中断時の挙動をPAUSEに設定したことによるポーズはPAUSEDとなる。

## getPlaying

再生中のボイスを取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

## getWaiting

待機中のボイスを取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandles | Uint32[] | voiceハンドルの配列 |

**【エラーコード】**

なし

**【機能】**

## play

ボイスの再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

同時に再生できるボイスは1つのみである。

内部の振る舞いはFIFOバッファであり、

2つ目以降はバッファの末尾に積まれる。

下記のInterruptメソッドや、警告サウンドによってボイスが中断された時の挙動を引数で設定し、

PAUSEの場合は割り込み中はボイスをポーズして完了時に再開、

STOPの場合は割り込みが起きた瞬間にボイスを終了(再吹鳴なし)、

REPLAYの場合は割り込みが終わると再び最初から再生を行う。

## playText

テキスト読み上げによるボイスの再生

**【パラメータ】**

| | | |
|---|---|---|
| text | String | 読み上げるテキスト |
| volume | Uint8 | 音量 |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

音源を再生するのとは異なり、引数にテキストをとって読み上げるAPIである。

## playBlank

無音の再生

**【パラメータ】**

| | | |
|---|---|---|
| duration | Uint32 | 再生時間(ミリ秒) |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

複数のボイスを間をあけて再生したい場合に用いる。

## interrupt

ボイスの割り込み再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| volume | Uint8 | 音量 |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

現在再生中のボイスを停止し、割り込んで再生を行う。

## interruptText

テキスト読み上げによるボイスの割り込み再生

**【パラメータ】**

| | | |
|---|---|---|
| text | String | 読み上げるテキスト |
| volume | Uint8 | 音量 |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| voiceHandle | Uint32 | voiceハンドル |

**【エラーコード】**

なし

**【機能】**

interruptのテキスト読み上げ版。

## stop

ボイスの停止

**【パラメータ】**

| | | |
|---|---|---|
| voiceHandle | Uint32 | voiceハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

引数にとったボイスが再生中ならば下記のSkipと同様、バッファに積まれている場合はバッファからそのボイスを取り除く。

## pause

ボイスの中断

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

現在再生中のボイスを中断する。

下記のUnPauseメソッドを呼ぶことで再開する。

## unPause

ボイスの再開

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

## skip

再生中のボイスをスキップ

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

現在再生中のボイスをスキップし、バッファに積まれた次のボイスを再生する。

# 4.41. Vehicle.HMI.Sound.Alert.SoundEffect（略称：SoundEffect）

## 4.41.1. 位置付けと機能

Vehicle.HMI.Sound.Alert.SoundEffect (略称: SoundEffect) は，警告・通知用効果音の再生管理機能を提供するシングルインスタンスオブジェクトである。

### 4.41.1.1. SoundEffect(Alert)の位置付けと性質

Vehicle.HMI.Sound.Alert.SoundEffectは，衝突警告，車線逸脱警告などの安全に関わる警告音や，システム通知音の再生機能を提供する。アラート音は高い優先度を持ち，他の音声出力に割り込んで再生される。

## 4.41.2. 機能クラス

## 4.41.3. リスククラス

## 4.41.4. 構成情報

## 4.41.5. 状態

## 4.41.6. サービスコール一覧

SoundEffectに対するサービスコールは以下の通り。

| | |
|---|---|
| play | 警告SoundEffectの再生 |

## 4.41.7. イベント

## 4.41.8. データ型の定義

## 4.41.9. サービスコールの詳細規定

### play

警告SoundEffectの再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

警告サウンドが吹鳴するとき、他の全てのサウンドは音量が絞られ、ユーザーサウンドのボイスは停止する。

警告サウンドは1音のみで、後に再生されたものを優先し、音量は既定の最大音量で固定となる。

# 4.42. Vehicle.HMI.Sound.Alert.Voice（略称：Voice）

## 4.42.1. 位置付けと機能

Vehicle.HMI.Sound.Alert.Voice（略称: Voice）は，警告・通知用音声メッセージの再生管理機能を提供するシングルインスタンスオブジェクトである。

### 4.42.1.1. Voice(Alert)の位置付けと性質

Vehicle.HMI.Sound.Alert.Voiceは，安全警告や緊急通知のための音声メッセージの再生機能を提供する。アラート音声は最高優先度で出力され，他の音声再生に割り込む。

## 4.42.2. 機能クラス

## 4.42.3. リスククラス

## 4.42.4. 構成情報

## 4.42.5. 状態

## 4.42.6. サービスコール一覧

Voiceに対するサービスコールは以下の通り。

| | |
|---|---|
| play | 警告ボイスの再生 |
| playText | テキスト読み上げによる警告ボイスの再生 |

## 4.42.7. イベント

## 4.42.8. データ型の定義

### 4.42.8.1. 型定義

```
type VoiceInterruptBehavior = enum {
    PAUSE,  // 途中から再開
    STOP,   // 再吹鳴なし
```

```
    REPLAY  // 頭から再度再生
};
```

## 4.42.9. サービスコールの詳細規定

### play

警告ボイスの再生

**【パラメータ】**

| | | |
|---|---|---|
| soundId | Uint32 | 音源を表すid |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

上記のSound.Alert.SoundEffect.Playを参照

### playText

テキスト読み上げによる警告ボイスの再生

**【パラメータ】**

| | | |
|---|---|---|
| text | String | 読み上げるテキスト |
| interruptBehavior | VoiceInterruptBehavior | 中断された時の挙動 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

なし

**【機能】**

上記のSound.Alert.SoundEffect.Playを参照

# 4.43. Vehicle.HMI.Switch（略称：Switch）

## 4.43.1. 位置付けと機能

Vehicle.HMI.Switch は、車両の物理スイッチをアプリケーション上の論理スイッチとしてマッピングし、制御するためのマルチインスタンスオブジェクトである。

### 4.43.1.1. 本APIで扱うスイッチの範囲

ユーザーの操作に対し、物理的なストロークやフィードバックを伴うスイッチ（プッシュ、トグル、ロータリー）を対象とし、静電容量方式のタッチパネルや、ジェスチャー入力等の物理的ストロークを伴わないインターフェースは対象外とする。

## 4.43.2. 機能クラス

Switchは，以下の機能クラスを持つ。

| | |
|---|---|
| StatusMonitorable | スイッチの押下状態を取得することができる |
| LedControllable | スイッチに付随するLEDの輝度をAPIから制御することができる |
| Customiziable | 物理スイッチに対する論理機能のマッピング（用途）を変更することができる |
| ModeChangeable | スイッチの動作モード（Default/Custom/GameMode）を切り替えることができる |

## 4.43.3. リスククラス

Switchは，以下のリスククラスを持つ。

| | |
|---|---|
| T.B.D | T.B.D |

## 4.43.4. 構成情報

Switchの各インスタンスは，以下の構成情報を持つ。

| | | |
|---|---|---|
| instanceId | IdType | インスタンスが持つId |
| switchType | SwitchInteractionType | スイッチ種別 |
| ledInstallation | ComponentInstallationType | LEDの実装有無 |
| position | String | スイッチの搭載位置を返す。 搭載位置は段数を特定できるものとする。<br><br>段数内のスイッチの割り振りは運転手から違い順に割り振る |

## 4.43.5. 状態

Switchの各インスタンスは，以下の状態を持つ。

| | | |
|---|---|---|
| switchStatus | SwitchStatusType | 現在の物理スイッチの動作状態。 ※詳細は SwitchStatusType の制約事項を参照。 |

## 4.43.6. サービスコール一覧

Switchに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfigAll | コンフィグ情報取得 |
| getStatus | スイッチの押下状態 |
| getSwitchMode | スイッチの現在モード |
| getSwitchCustomize | スイッチのカスタマイズ情報 |
| getSwitchLEDBrightness | スイッチのLED輝度を取得する |
| setSwitchLEDBrightness | スイッチのLED輝度を設定する |
| setSwitchMode | スイッチの現在モードを変更する |
| setSwitchCustomize | スイッチのカスタマイズ |
| notify | スイッチイベントの通知要求 |
| unnotify | スイッチイベントの通知停止 |
| getEvent | スイッチイベントの取得 |

## 4.43.7. イベント

スイッチに対するイベント種別にはスイッチの押下、スイッチに付随するLEDの輝度変更、論理機能の変更、動作モードの変更がある。

Switchに対するイベント種別は以下の通り。

| | |
|---|---|
| SwitchStateChanged | スイッチの押下状態が変更 |
| BrightnessChanged | スイッチのLEDの輝度が変更 |
| CustomizationChanged | スイッチのカスタマイズ変更 |
| OperationModeChanged | スイッチの現在モード変更 |

## 4.43.8. データ型の定義

### 4.43.8.1. スイッチ種別

```
type SwitchInteractionType = enum {
    Momentary,  // 押している間だけONになるスイッチ
    Toggle, // 押すたびに状態が保持・反転するスイッチ
    Rotary  // 回転操作を伴うスイッチ
};
```

#### 4.43.8.2. 部品実装の有無

```
type ComponentInstallationType = enum {
    NotInstalled,   // 未実装
    Installed   // 実装済み
};
```

#### 4.43.8.3. モーメンタリスイッチの押下状態

```
type MomentarySwitchStatusType = enum {
    Released,   // リリース状態
    Pressed // 押下状態
};
```

#### 4.43.8.4. トグルスイッチの押下状態

```
type ToggleSwitchStatusType = enum {
    Inactive,   // OFF状態
    Active  // ON状態
};
```

#### 4.43.8.5. ロータリースイッチの押下状態

```
type RotarySwitchStatusType = enum {
    NoAction,   // 変化なし
    RotateRight,    // 右回転
    RotateLeft  // 左回転
};
```

#### 4.43.8.6. スイッチの押下状態

```
type SwitchStatusType = struct {
    momentarySwichiStatus : MomentarySwitchStatusType,
    toggleSwichiStatus : ToggleSwitchStatusType,
    rotarySwichiStatus : RotarySwitchStatusType
};
```

#### 4.43.8.7. スイッチのカスタマイズ状態

```
type CustomizeModeType = enum {
    Default,    // カスタマイズされていな状態を示す
    Customiz,   // スイッチをユーザー任意に設定しているモードを示す
```

```
    GameMode     // OS側の機能をバイパスし、アプリへ押下状態のみを通知するモード
};
```

#### 4.43.8.8. スイッチの構成情報(getConfigAllが返すデータ型)

```
type SwitchConfigType = struct {
    instanceId : IdType,
    switchType : SwitchInteractionType,
    ledInstallation : ComponentInstallationType,
    position : String
};
```

#### 4.43.8.9. スイッチイベント

```
type SwitchEventKindType = enum {
    SwitchStateChanged,
    BrightnessChanged,
    CustomizationChanged,
    OperationModeChanged
};
```

```
type SwitchEventType = struct {
    instanceId : IdType,
    eventInfo : SwitchEventKindType
};
```

### 4.43.9. サービスコールの詳細規定

#### getConfigAll

コンフィグ情報取得

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | SwitchConfigType[] | すべてのスイッチに関する構成情報 |

**【エラーコード】**

なし

**【機能】**

すべてのスイッチに関する構成情報を返す。

## getStatus

スイッチの押下状態

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| switchStatus | SwitchStatusType | モーメンタリ、トグル、ロータリスイッチの状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のスイッチの押下状態を返す。

返却される switchStatus の中身は、物理スイッチの種別（switchType）に依存する。

- Momentary の場合: momentaryStatus フィールドのみを参照。
- Toggle の場合: toggleStatus フィールドのみを参照。
- Rotary の場合: rotaryStatus フィールドのみを参照。


OSにてスイッチのチャタリング除去を行い通知する。チャタリングと連打は各OEMにて判断する。

## getSwitchMode

スイッチの現在モード

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| customizMode | CustomizeModeType | スイッチのカスタマイズ状態 |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

操作対象のスイッチの現在モードを返す。

割り当たっているカスタマイズの情報はgetSwitchCustomizeで取得する。

## getSwitchCustomize

スイッチのカスタマイズ情報

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| function | String | カスタマイズ後の割り当て機能。 戻り値の例としてドライバーのエアコン温度を上げる機能が割り当たっている場合は以下を返す。<br>"HVAC.TempUpDriverSide" |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

スイッチに割り当たっている機能を取得する機能

## getSwitchLEDBrightness

スイッチのLED輝度を取得する

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| brightness | UInt32 | **正常状態** [0%～100%], **故障状態** [101以上] |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

スイッチのLED輝度を取得する。

## setSwitchLEDBrightness

スイッチのLED輝度を設定する

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |
| brightness | UInt32 | 0〜100を指定する。 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

スイッチのLED輝度を設定する。

## setSwitchMode

スイッチの現在モードを変更する

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |
| customizMode | CustomizeModeType | スイッチのカスタマイズ状態 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

スイッチモードを変更する

機能割り当てはsetSwitchCustomizeにて行う。

## setSwitchCustomize

スイッチのカスタマイズ

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType | 操作対象のスイッチのID |

| | | |
|---|---|---|
| function | String | カスタマイズ後の割り当て機能。 例としてドライバーのエアコン温度を上げる機能が割り当てる場合は"HVAC.TempUpDriverSide"を設定する。 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |

**【機能】**

スイッチモードがCustomizの場合に任意の機能を割り当てる。

スイッチモードはgetSwitchModeで取得する。

## notify

スイッチイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| instanceId | IdType? | 操作対象のスイッチのID |
| eventFilter | SwitchEventKindType? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_ID | instanceIdが有効範囲外 |
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

イベント通知対象のスイッチで発生したeventFilterで指定したイベントの通知を要求する。イベントを通知するためのイベントキューを生成し，その通知ハンドルを notifyHandle に返す。

instanceIdを省略した場合，すべてのインスタンスで発生したイベントの通知を要求する。eventFilterを省略した場合，すべての種類のイベントの通知を要求する。

## unnotify

スイッチイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。指定した通知用のイベントキューは削除される。

### getEvent

スイッチイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントキューを識別するための通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | SwitchEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

# 4.44. Vehicle.Motion（略称：Motion）

## 4.44.1. 位置付けと機能

Vehicle.Motion（略称：Motion）は，車両の運動を操作するためのシングルトンオブジェクトである。Motionの状態を参照することにより，車両の現在の速度や加速度を取得することができる。また，Motionを制御することにより，車両のパワートレイン，ブレーキ，ステアリングなどを制御することができる。

#### 4.44.1.1. Motionの機能

Motionは，車両を縦方向の運動と横方向の運動を独立して制御する機能を持つ。縦方向の運動は，目標速度を指定して制御する方法（速度制御）と，停止するまでの距離を指定して制御する方法（停止制御）がある。横方向の運動は，目標軌跡を指定して制御する（軌跡追従制御）。

縦方向の速度制御では，指定された目標速度に向けて加減速または速度維持を行う。加減速する場合には，指定された応答プロファイルによって定まる加減速度制限をかける（Figure 13）。ただし，応答プロファイルが最大性能での加減速（Maximum）の場合には，加減速度制限はかけない。

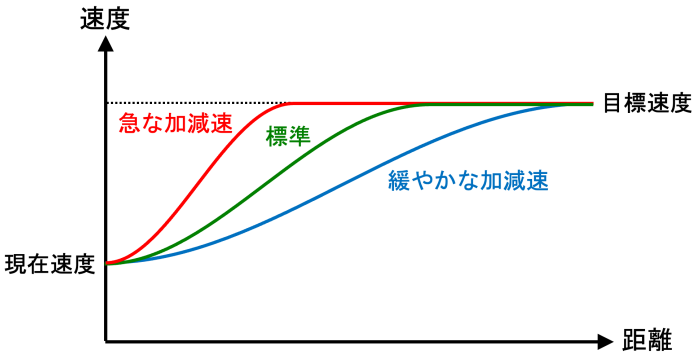



*Figure 13.* 応答プロファイルの速度制御への適用イメージ

縦方向の停止制御では，指定された目標停止位置に向けて速度維持または減速を行い，目標位置で車両を停止させる。減速する場合には，指定された応答プロファイルによって定まる減速度制限をかける（Figure 14）。ただし，応答プロファイルが最大性能での加減速（Maximum）の場合には，減速度制限をかけず，車両の最大性能で停止する。車両が目標停止位置で停止した後は，停止状態を維持する。停止状態を維持する間も，停止制御が継続しているものと扱う。

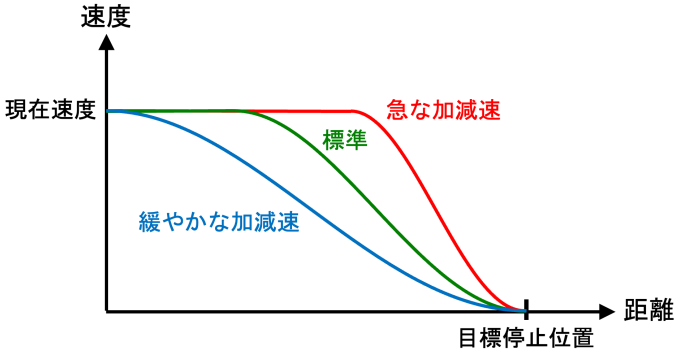



*Figure 14.* 応答プロファイルの停止制御への適用イメージ

なお，ビークルOSは，必要に応じて，縦方向の運動制御に対してジャーク制限をかける。

横方向の軌跡追従制御では，指定された目標軌跡に沿って走行するように制御を行う。軌跡追従制御において，指定された許容横方向誤差を満たせない場合，車両を減速させる場合がある。

アプリケーションは，目標軌跡を設定するサービスコールを継続的に呼び出して，十分に先までの目標軌跡を設定することを想定している。軌跡追従制御中に目標軌跡の残っている点が少なくなり，安定した横方向制御が難しくなった場合，ビークルOSはイベント（LateralPathInsufficient）を用いてアプリケーションに通知する。また，車両が目標軌跡の最後の点に到達することはないのが原則であるが，アプリケーションが想定したようにサービルコールを呼び出さず，目標軌跡の最後の点に到達した場合，その後は，その時点の曲率を維持するように制御を継続する。

#### 4.44.1.2. 運転者の操作による制御と安全機能

運転者は，アクセルペダル，ブレーキペダル，ステアリングホイールなどを操作することで，車両の縦方向および横方向の運動を制御することができる（自動運転時には，運転者の操作による制御を無効化することもある）。運転者の操作による制御は，アプリケーションで実現された標準制御と扱う。すなわち，

「運転者の操作による制御」というアプリケーションが，アクセルペダル，ブレーキペダル，ステアリングホイールなどの状態を元に，ある操作優先度で，車両の運動を制御するものと扱う。ただし，このアプリケーションは，MotionのAPIを利用するわけではない（Figure 15）。

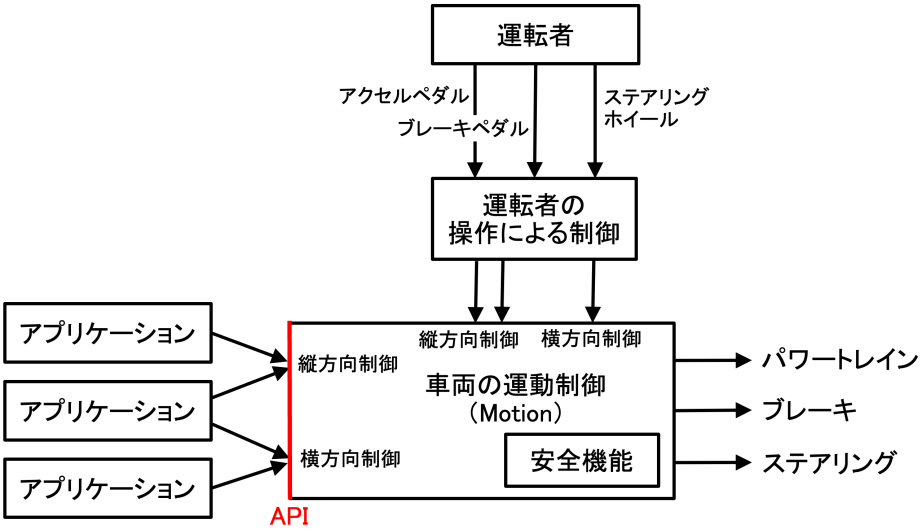



*Figure 15. Motionの制御の構造*

ビークルOSが車両運動に関してどのような安全機能を持つかは，車両依存である。ビークルOSが何らかの危険を検知した場合，ビークルOSの安全機能は，縦方向または横方向またはその両方の制御をオーバライドする。

目標軌跡に対する追従制御中に，許容横方向誤差を満たすために車両を減速させている間は，縦方向制御は安全機能によりオーバライドされたものと扱う。減速しても許容横方向誤差を満たせない場合には，横方向および縦方向の制御は安全機能によりオーバライドされ，車両の振る舞いは安全機能に委ねられる。

#### 4.44.1.3. Motionに対する制御の調停

Motionに対する制御の調停には，縦方向制御と横方向制御のそれぞれに対して，標準的な調停機能（後勝ちの原則）が適用される。また，縦方向制御と横方向制御のそれぞれに対して，LockableObjectと同等の標準的なロック機能を持つ。

車両の故障により制御できなくなった場合や，安全機能が制御をオーバライドした場合，ロックは解除され，それらの状態から回復するまで，ロックを取得することはできない。

### 4.44.2. 機能クラス

Motionは，以下の機能クラスを持つ。

| | |
|---|---|
| ControlLongitudinal | APIにより縦方向制御ができる |
| ControlLateral | APIにより横方向制御ができる |

### 4.44.3. リスククラス

Motionは，以下のリスククラスを持つ。

| | |
|---|---|
| RiskVehicleInstability | 車両の運動安定性を失うリスク（スピン，横滑り，横転など） |
| RiskCollision | 自車両の責任で何らかの物体（他の車両，歩行者，障害物，落下物など）に衝突するリスク |

| | |
|---|---|
| RiskViolation | 交通規則に違反するリスク |
| RiskRoadHazard | 道路の損傷，欠損，路肩崩落などにより安全に走行可能な路面が確保できないリスク |

以上のリスククラスの定義は，現時点でのイメージを示したものであり，より詳細な検討が必要である。

## 4.44.4. 構成情報

Motionは，以下の構成情報を持つ。

| | | |
|---|---|---|
| capabilities | enumSet (MotionCapabilityType) | 備えている機能クラスの集合 |
| riskClasses | enumSet(MotionRiskType) | 対策できていないリスククラスの集合 |

この他に，車両の運動性能に関する情報を追加する（TBD）。

## 4.44.5. 状態

Motionは，以下の状態を持つ。

| | | |
|---|---|---|
| speed | SpeedType | 現在速度 |
| acceleration | AccelerationType | 現在加速度 |
| angularVelocity | AngularVelocityType | 現在回転速度 |
| longitudinalLockStatus | LockStatusType | 縦方向のロック状態 |
| lateralLockStatus | LockStatusType | 横方向のロック状態 |
| longitudinalControlStatus | LongitudinalControlStatusType | 縦方向制御の状態 |
| targetSpeed | SpeedType? | 目標速度 |
| distanceToStop | UInt32? | 目標停止位置までの残り道のり距離（単位：mm） |
| responseProfile | LongitudinalResponseProfileType? | 応答プロファイル |
| lateralControlStatus | LateralControlStatusType | 横方向制御の状態 |
| targetPath | RelativePathType? | 残り目標軌跡 |
| lateralTolerance | UInt32? | 許容横方向誤差（単位：mm） |

targetSpeedは，縦方向が速度制御されている間のみ，distanceToStopは，縦方向が停止制御されている間のみ，responseProfileは，縦方向が速度制御または停止制御されている間のみ有効である。また，targetPathとlateralToleranceは，横方向が軌跡追従制御されている間のみ有効である。

縦方向制御の状態（longitudinalControlStatus）の状態遷移を，Figure 16に示す。故障回復時やオーバ

ライド解除時には，速度制御中と停止制御中のいずれに遷移する場合もある。

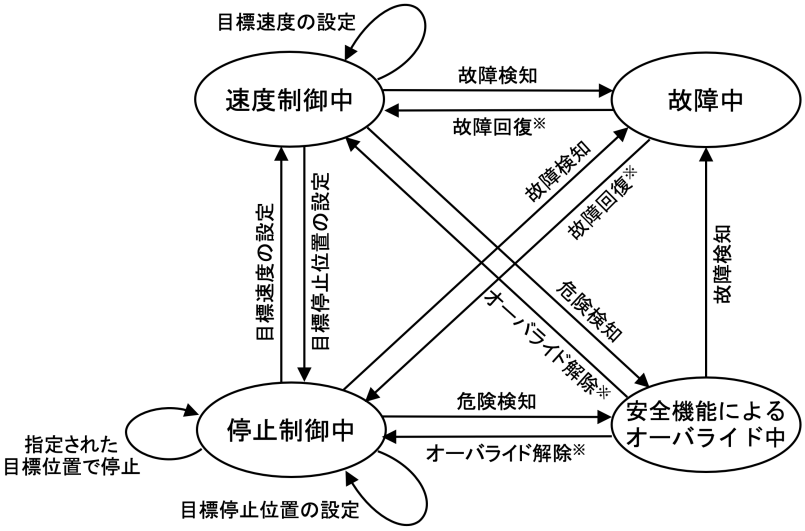



*Figure 16. Motionの縦方向制御の状態遷移*

また，横方向制御の状態（lateralControlStatus）の状態遷移を，Figure 17に示す。故障回復時やオーバライド解除時には，軌跡追従制御中と曲率維持中のいずれに遷移する場合もある。

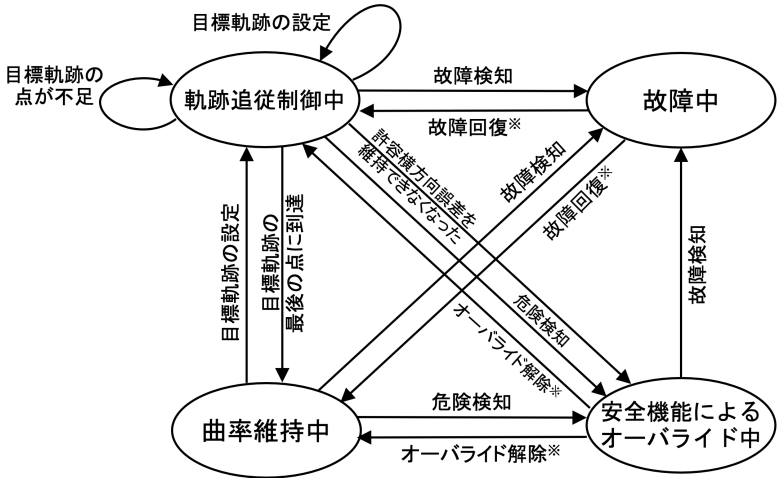



*Figure 17. Motionの横方向制御の状態遷移*

distanceToStopは，目標停止位置までの残り道のり距離であり，サービスコールにより目標停止位置までの道のり距離を設定した直後はその値となるが，その後は，走行した距離分減少した値となる。

targetPathは，目標軌跡の残り部分であり，サービスコールにより目標軌跡を設定した直後はその値となるが，その後は，走行に伴って到達した点が除かれたものとなる。

speed，acceleration，angularVelocityの定義は，COVESA VSS 6.0に合わせた。

## 4.44.6. サービスコール一覧

Motionに対するサービスコールは以下の通り。

| | |
|---|---|
| getConfig | Motionの構成情報の参照 |
| getStatus | Motionの状態の参照 |
| setSpeed | 目標速度の設定（縦方向制御） |

| | |
|---|---|
| stopDistance | 目標停止位置の設定（縦方向制御） |
| setPath | 目標軌跡の設定（横方向制御） |
| lockLongitudinal | Motionの縦方向制御のロックの取得 |
| unlockLongitudinal | Motionの縦方向制御のロックの解除 |
| lockLateral | Motionの横方向制御のロックの取得 |
| unlockLateral | Motionの横方向制御のロックの解除 |
| notify | Motionイベントの通知要求 |
| unnotify | Motionイベントの通知停止 |
| getEvent | Motionイベントの取得 |

## 4.44.7. イベント

Motionに対するイベント種別は以下の通り。

| | |
|---|---|
| LongitudinalControlledBySelf | 自アプリケーションが縦方向制御 |
| LongitudinalControlledByOther | 他のアプリケーションが縦方向制御 |
| LongitudinalStopCompleted | 指定された目標位置で停止 |
| LongitudinalFaultDetected | 縦方向制御の故障検知 |
| LongitudinalFaultRecovered | 縦方向制御が故障から回復 |
| LongitudinalOverrideActivated | 安全機能が縦方向制御をオーバライド |
| LongitudinalOverrideReleased | 縦方向制御のオーバライドが解除 |
| LateralControlledBySelf | 自アプリケーションが横方向制御 |
| LateralControlledByOther | 他のアプリケーションが横方向制御 |
| LateralPathEndReached | 指定された目標軌跡の最後の点に到達 |
| LateralPathInsufficient | 指定された目標軌跡の点が不足 |
| LateralFaultDetected | 横方向制御の故障検知 |
| LateralFaultRecovered | 横方向制御が故障から回復 |
| LateralToleranceExceeded | 許容横方向誤差を満たせなくなった |
| LateralOverrideActivated | 安全機能が横方向制御をオーバライド |
| LateralOverrideReleased | 横方向制御のオーバライドが解除 |
| OwnLongitudinalLockRevoked | 自アプリケーションの縦方向制御のロックの強制解除 |

| | |
|---|---|
| OtherLongitudinalLockReleased | 他のアプリケーション等による縦方向制御のロック解除 |
| OwnLateralLockRevoked | 自アプリケーションの横方向制御のロックの強制解除 |
| OtherLateralLockReleased | 他のアプリケーション等による横方向制御のロック解除 |
| EventOverflow | イベントキューのオーバフロー |
| ConfigChanged | 構成情報の変更 |

横方向制御において許容横方向誤差を満たせなくなった場合，横方向制御は安全機能によりオーバライドされるが，この時は，LateralToleranceExceededのみが発生し，LateralOverrideActivatedは発生しない。

この内，イベント種別がLongitudinalControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がLongitudinalStopCompletedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がLateralControlledByOtherの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がLateralPathEndReachedの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がLateralPathInsufficientの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| sourceApplication | ApplicationIdType | 操作したアプリケーション |

また，イベント種別がEventOverflowの場合には，以下の関連情報を持つ。

| | | |
|---|---|---|
| noLostEvents | UInt32 | 失われたイベントの数 |

## 4.44.8. データ型の定義

### (1) Motionの機能クラス

```
type MotionCapabilityType = enum {
    ControlLongitudinal,
    ControlLateral
```

```
};
```

**(2) Motionに対するリスククラス**

```
type MotionRiskType = enum {
    RiskVehicleInstability,
    RiskCollision,
    RiskViolation,
    RiskRoadHazard
};
```

**(3) Motionの構成情報（getConfigが返すデータ型）**

```
type MotionConfigType = struct {
    capabilities : enumSet(MotionCapabilityType),
    riskClasses : enumSet(MotionRiskType)
};
```

**(4) Motionの状態**

```
type SpeedType = Float32;   // 速度（単位：km/h）。正の値は前進，負の値は後退を示す
```

```
type AccelerationType = struct {
    longitudinal : Float32, // 縦方向加速度（単位：m/s^2）
    lateral : Float32,  // 横方向加速度（単位：m/s^2）
    vertical : Float32  // 鉛直方向加速度（単位：m/s^2）
};
```

```
type AngularVelocityType = struct {
    roll : Float32, // ロール（前後軸まわりの回転速度，単位：度/s）
    pitch : Float32,    // ピッチ（左右軸まわりの回転速度，単位：度/s）
    yaw : Float32   // ヨー（鉛直軸まわりの回転速度，単位：度/s）
};
```

```
type LongitudinalControlStatusType = enum {
    SpeedControl,   // 速度制御中
    StopControl,    // 停止制御中
    Fault,  // 故障中
    SafetyOverride  // 安全機能によるオーバライド中
};
```

```
type LateralControlStatusType = enum {
```

```
    PathControl,    // 軌跡追従制御中
    CurvatureHold,  // 曲率維持中
    Fault,  // 故障中
    SafetyOverride  // 安全機能によるオーバライド中
};
```

```
type MotionStatusType = struct {
    speed : SpeedType,
    acceleration : AccelerationType,
    angularVelocity : AngularVelocityType,
    longitudinalLockStatus : LockStatusType,
    lateralLockStatus : LockStatusType,
    longitudinalControlStatus : LongitudinalControlStatusType,
    targetSpeed : SpeedType?,
    distanceToStop : UInt32?,
    responseProfile : LongitudinalResponseProfileType?,
    lateralControlStatus : LateralControlStatusType,
    targetPath : RelativePathType?,
    lateralTolerance : UInt32?
};
```

#### (5) Motionの応答プロファイル

```
type LongitudinalResponseProfileType = enum {
    Standard,   // 標準的な加減速
    Maximum,    // 車両の最大性能での加減速
    Rapid,  // 急な加減速
    Gentle  // 緩やかな加減速
};
```

#### (6) 走行軌跡

```
type Relative2DPositionType = struct {
    x : Int32,  // 車両の前後方向の相対位置（単位：mm，前方が正）
    y : Int32   // 車両の左右方向の相対位置（単位：mm，左方向が正）
};
```

Relative2DPositionTypeは，車両に固定された座標系（車両座標系）における相対位置を表すデータ型である。相対位置の原点は車両の後軸中心とする。

```
type RelativePathType = struct {
    points : Relative2DPositionType[],  // 軌跡を構成する点の列
    referenceTime : TimestampMsType // pointsが参照する車両座標系の基準時刻
};
```

アプリケーションが自車位置や周辺環境を認識した時点から，このデータ型を用いるサービスコールを呼

び出すまでの間に，車両は移動している。アプリケーションが，軌跡計算の元となった車両状態の取得時刻をreferenceTimeに指定し，pointsの各点はreferenceTime時点の車両座標系を基準とした相対位置を表すとすることで，車両の移動による位置ズレを補正した正確な軌跡の指定が可能となる。

Relative2DPositionTypeは，将来的には，共通データ型に移動すべきと思われる。

## 4.44.9. サービスコールの詳細規定

### getConfig

Motionの構成情報の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| config | MotionConfigType | Motionの構成情報 |

**【エラーコード】**

なし

**【機能】**

Motionの構成情報を返す。

### getStatus

Motionの状態の参照

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| status | MotionStatusType | Motionの状態 |

**【エラーコード】**

なし

**【機能】**

Motionに関するすべての状態を返す。

## setSpeed

目標速度の設定（縦方向制御）

**【パラメータ】**

| | | |
|---|---|---|
| targetSpeed | SpeedType | 目標速度 |
| responseProfile | LongitudinalResponseProfileType? | 応答プロファイル |
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | targetSpeed，responseProfile，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | Motionの縦方向制御が故障中 |
| E_OBJECT_STATUS | Motionの縦方向制御が安全機能にオーバライドされている |
| E_OBJECT_LOCKED | Motionの縦方向制御が，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_NOT_SUPPORTED | 縦方向制御する機能を持たない |

**【機能】**

目標速度をtargetSpeed，応答プロファイルをresponseProfileに設定して，車両の速度制御を開始する。

停止制御中に目標速度を設定した場合，目標停止位置の設定は解除され，停止制御は中断される。

responseProfileを省略した場合，標準的な加減速（Standard）を指定したものと扱う。

リスククラスRiskViolationに対応できていない場合でも，車両が私有地を走行していると判断できた場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## stopDistance

目標停止位置の設定（縦方向制御）

**【パラメータ】**

| | | |
|---|---|---|
| distanceToStop | UInt32 | 目標停止位置までの道のり距離（単位：mm） |
| responseProfile | LongitudinalResponseProfileType? | 応答プロファイル |

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | distanceToStop，responseProfile，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | Motionの縦方向制御が故障中 |
| E_OBJECT_STATUS | Motionの縦方向制御が安全機能にオーバライドされている |
| E_OBJECT_LOCKED | Motionの縦方向制御が，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_INFEASIBLE | 車両が停止中。指定された距離で停止できない |
| E_NOT_SUPPORTED | 縦方向制御する機能を持たない |

**【機能】**

目標停止位置までの道のり距離（目標軌跡に沿った距離）をdistanceToStop，応答プロファイルをresponseProfileに設定して，車両の停止制御を開始する。

車両が停止中にこのサービスコールを呼び出した場合は，distanceToStopが0でない限り，E_INFEASIBLEエラーとなる。

distanceToStopで指定された距離で停止できない場合は，次のように振る舞う。応答プロファイルが最大性能での加減速（Maximum）の場合には，車両に可能な最も短い距離で停止するように制御する。それ以外の場合は，E_INFEASIBLEエラーとなる。

速度制御中に目標停止位置を設定した場合，目標速度の設定は解除され，速度制御は中断される。

responseProfileを省略した場合，標準的な加減速（Standard）を指定したものと扱う。

リスククラスRiskViolationに対応できていない場合でも，車両が私有地を走行していると判断できた場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## setPath

目標軌跡の設定（横方向制御）

**【パラメータ】**

| | | |
|---|---|---|
| path | RelativePathType | 目標軌跡 |
| lateralTolerance | UInt32? | 許容横方向誤差（単位：mm） |

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | path，lateralTolerance，priorityが有効範囲外 |
| E_RISK_APPLICATION | アプリケーションが対応していないリスクがある |
| E_RISK_USER | ユーザがリスクを承認していない |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_FAULT | Motionの横方向制御が故障中 |
| E_OBJECT_STATUS | Motionの横方向制御が安全機能にオーバライドされている |
| E_OBJECT_LOCKED | Motionの横方向制御が，他のアプリケーションにより，priorityと同じかそれより高い優先度でロックされている |
| E_INFEASIBLE | 指定された許容横方向誤差を満たせない |
| E_NOT_SUPPORTED | 横方向制御する機能を持たない |

**【機能】**

目標軌跡をpath，許容横方向誤差をlateralToleranceに設定し，軌跡追従制御を開始する。

車両を減速させてもlateralToleranceで指定された許容横方向誤差を満たせない場合には，E_INFEASIBLEエラーとなる。

lateralToleranceを省略した場合，横方向誤差に関する制約は設けないものと扱う。

リスククラスRiskViolationに対応できていない場合でも，車両が私有地を走行していると判断できた場合には，操作を受け付ける。そうでない場合には，E_RISK_APPLICATIONエラーまたはE_RISK_USERエラーとなる。

## lockLongitudinal

Motionの縦方向制御のロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | priorityが有効範囲外 |

| | |
|---|---|
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |
| E_OBJECT_LOCKED | Motionの縦方向制御が，priorityと同じかそれより高い優先度でロックされている（自アプリケーションが，priorityと同じかそれより高い優先度でロックしている場合を含む） |
| E_OBJECT_FAULT | Motionの縦方向制御が故障中 |
| E_OBJECT_STATUS | Motionの縦方向制御が安全機能にオーバライドされている |

**【機能】**

Motionの縦方向制御を，指定した操作優先度でロックする。

## unlockLongitudinal

Motionの縦方向制御のロックの解除

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_OBJECT_STATUS | 自アプリケーションが，Motionの縦方向制御をロックしていない |

**【機能】**

Motionの縦方向制御を，ロック解除する。

## lockLateral

Motionの横方向制御のロックの取得

**【パラメータ】**

| | | |
|---|---|---|
| priority | PriorityType? | 操作優先度 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | priorityが有効範囲外 |
| E_DENIED_PRIORITY | priorityが，アプリケーションが使用できない値 |

| | |
|---|---|
| E_OBJECT_LOCKED | Motionの横方向制御が，priorityと同じかそれより高い優先度でロックされている（自アプリケーションが，priorityと同じかそれより高い優先度でロックしている場合を含む） |
| E_OBJECT_FAULT | Motionの横方向制御が故障中 |
| E_OBJECT_STATUS | Motionの横方向制御が安全機能にオーバライドされている |

**【機能】**

Motionの横方向制御を，指定した操作優先度でロックする。

## unlockLateral

Motionの横方向制御のロックの解除

**【パラメータ】**

なし

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_OBJECT_STATUS | 自アプリケーションが，Motionの横方向制御をロックしていない |

**【機能】**

Motionの横方向制御を，ロック解除する。

## notify

Motionイベントの通知要求

**【パラメータ】**

| | | |
|---|---|---|
| eventFilter | enumSet (MotionEventKindType)? | 通知するイベントの種類の集合 |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_PARAMETER | eventFilterが有効範囲外 |

**【機能】**

Motionで発生したeventFilterで指定した種別のイベントの通知を要求する。イベントを格納するためのイベントキューを生成し，その通知ハンドルをnotifyHandleに返す。

## unnotify

Motionイベントの通知停止

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | 操作対象の通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |

**【機能】**

notifyHandleで指定した通知を停止する。イベントを格納するためのイベントキューを削除する。

## getEvent

Motionイベントの取得

**【パラメータ】**

| | | |
|---|---|---|
| notifyHandle | NotifyHandleType | イベントを取得するイベントキューの通知ハンドル |

**【リターンパラメータ】**

| | | |
|---|---|---|
| returnValue | ReturnType | 正常終了またはエラーコード |
| event | MotionEventType | 取得したイベント |

**【エラーコード】**

| | |
|---|---|
| E_INVALID_HANDLE | notifyHandleが不正 |
| E_NO_DATA | イベントキューが空 |

**【機能】**

notifyHandleで指定した通知用のイベントキューから，先頭のイベントを取り出す。取り出したイベントは，イベントキューから削除される。


イベントがパラメータ等で渡されてくるプログラミング環境向けの物理APIでは，用意する必要がない。

## 4.45. Vehicle.CurrentLocation（略称：CurrentLocation）

別の資料を参照。

## 4.46. Vehicle.Driver（略称：Driver）

別の資料を参照。

## 4.47. Vehicle.Occupant（略称：Occupant）

＜TBD＞

## 4.48. Vehicle.SurroundModel（略称：SurroundModel）

別の資料を参照。

## 4.49. Vehicle.Atmosphere（略称：Atmosphere）

＜TBD＞

# Chapter 5. 拡張ビークルAPI

＜TBD＞

# Appendix A: 車両状態の定義

＜TBD＞

# Appendix B: C言語向け物理API（C言語バインディング）

## B.1. データ型

ここでは，Section 2.1.2で規定した論理APIにおけるデータ型を，どのようにC言語のデータ型で実現するかを規定する。なお，物理APIにおけるデータ型は，状況毎に最適なものとし，論理APIから機械的に決まるとは限らないものとする。

#### (1) 基本型

素直に対応するデータ型で実現する。

特殊値の定数名は，先頭に名前の衝突を避けるための文字列を付加することを原則とし，#defineを用いて対応する数値に定義する。

#### (2) 文字列型

`char *`で実現する。

#### (3) 固定長配列

配列で実現する。

#### (4) 可変長配列

配列を置いた領域へのポインタと要素数をフィールドに持つ構造体で実現することを原則とする。

#### (5) 列挙型

C言語の列挙型で実現する。

列挙子を表す定数名は，列挙子の名前の前に，列挙型を区別するための文字列を付加した名称とすることを原則とする。列挙子には，原則として，1から連続する正の数値を明示的に割り付ける。

ただし，ReturnTypeはint32_t型で実現し，E_OKに0，その他の列挙子には負の数値を割り付ける。ReturnTypeの列挙子を表す定数名は，列挙子の名前とし，#defineを用いて割り付けられた数値に定義する。

#### (6) 列挙型の集合型

各列挙子が1つのビットに対応する個別に定めるサイズの符号なし整数型（ビットマップ）で実現する。

空集合を表すマクロ（emptyEnumSet）と，列挙子からそれに対応するビットを求めるためのマクロ（enumSetBit）を用意する。これらのマクロの定義は次の通り。

```
#define emptyEnumSet            0U
#define enumSetBit(Enumerator)  (1U << ((Enumerator) - 1))
```

#### (7) 構造体

構造体で実現する。

#### (8) バリアント（タグ付き共用体）

パラメータの列挙型と，各ペイロードのデータ型（原則として構造体）の共用体をフィールドに持つ構造体で実現することを原則とする。

#### (9) オプショナル型

基礎となるデータ型に無効を示す値がある場合には，その値でnullを表すことを原則とする。

そうでない場合には，基礎となるデータ型へのポインタで実現し，NULLポインタでnullを表すことを原則とする。

## B.2. 共通データ型

#### (1) 戻り値のデータ型

```
typedef int32_t     ReturnType;

#define E_OK                    0       /* 正常終了 */
#define E_INVALID_ID            -1      /* IDが不正 */
#define E_INVALID_HANDLE        -2      /* ハンドルが不正 */
#define E_INVALID_PARAMETER     -3      /* その他のパラメータが不正 */
#define E_RISK_APPLICATION      -4      /* アプリケーションが対応していないリ
                                           スクがある */
#define E_RISK_USER             -5      /* ユーザがリスクを受け入れていない */
#define E_DENIED_PRIORITY       -6      /* 使用できない操作優先度 */
#define E_DENIED_ACCESS         -7      /* アクセスが認められていない */
#define E_OBJECT_FAULT          -8      /* 対象オブジェクトが故障中 */
#define E_OBJECT_LOCKED         -9      /* 対象オブジェクトがロックされている */
#define E_OBJECT_STATUS         -10     /* 対象オブジェクトが指定された操作を
                                           できない状態にある */
#define E_INFEASIBLE            -11     /* 指定された操作が現在の状態では実現
                                           できない */
#define E_NO_DATA               -12     /* データがない */
#define E_NO_MEMORY             -13     /* メモリ不足 */
#define E_NOT_SUPPORTED         -14     /* サポートされていない機能 */
#define E_COMMUNICATION         -15     /* 通信エラー */
#define E_NOT_OK                -16     /* その他のエラー */
```

#### (2) インスタンスIDのデータ型

```
typedef uint16_t    IdType;

#define ID_NULL     0           /* オプショナル型でnullを示す */
```

### (3) 操作優先度のデータ型

```
typedef uint8_t     PriorityType;

#define PRIORITY_NULL   0       /* オプショナル型でnullを示す */
```

### (4) アプリケーションIDのデータ型

アプリケーションIDのデータ型は，OS/プラットフォーム依存に規定する。

### (5) オブジェクトを表すデータ型

```
typedef enum {
    ObjectKindVehicle = 1,
    ObjectKindBody = 2,
    ObjectKindCabin = 3,
    ObjectKindDoor = 4,
    ObjectKindWindow = 5
} ObjectKindType;

typedef struct {
    ObjectKindType  objectKind;
    IdType          instanceId;
} RelatedObjectType;
```

シングルトンの場合は，instanceIdをID_NULLとする。

### (6) オブジェクトの場所を表すデータ型

```
typedef enum {
    ZoneLeft = 1,           /* 左 */
    ZoneRight = 2,          /* 右 */
    ZoneCenter = 3,         /* 中央 */
    ZoneTop = 4,            /* 上 */
    ZoneBottom = 5          /* 下 */
} ZoneType;

typedef struct {
    uint8_t     row;        /* 座席の何列目か */
    ZoneType    zone;       /* 左右上下方向の位置 */
} PlacementType;
```

### (7) ロック状態のデータ型

```
typedef struct {
    bool                locked;
    ApplicationIdType   lockApplication;
```

```
    PriorityType        lockPriority;
} LockStatusType;
```

lockApplicationとlockPriorityは，lockedがtrueの場合のみ有効。

#### (8) 位置を表すデータ型

```
typedef int8_t PositionType;

#define POSITION_UNKNOWN    -1
#define POSITION_NONZERO    -2
```

# B.3. コアビークルAPIのデータ型

## B.3.1. WindowのAPIのデータ型

#### (1) Windowの機能クラスのデータ型

```
typedef enum {
    WindowMovable = 1,              /* APIにより移動できる */
    WindowClosedDetectable = 2,     /* 全閉状態を検知できる */
    WindowHasMove2 = 3,             /* 第二次元方向に移動できる */
    WindowMoveProfileSupported = 4  /* 移動プロファイルをサポートしている */
} WindowCapabilityType;

typedef uint32_t WindowCapabilitySetType;
```

#### (2) Windowに対するリスククラスのデータ型

```
typedef enum {
    WindowRiskPinch = 1,        /* 挟み込みのリスク */
    WindowRiskOpen = 2          /* 開動作に伴う諸々のリスク */
} WindowRiskType;

typedef uint32_t WindowRiskSetType;
```

#### (3) Windowの構成情報のデータ型

```
typedef struct {
    char                    *instanceName;
    IdType                  instanceId;
    PlacementType           placement;
    WindowCapabilitySetType capabilities;
    WindowRiskSetType       riskClasses;
    RelatedObjectType       mountedOn;
```

```
    char                    *displayName;
} WindowConfigType;
```

### (4) Window状態のデータ型

```
typedef enum {
    WindowStopped = 1,          /* 停止中 */
    WindowOpening = 2,          /* 開動作中 */
    WindowClosing = 3,          /* 閉動作中 */
    WindowFault = 4,            /* 故障中 */
    WindowPinchAvoiding = 5     /* 挟み込み回避中 */
} WindowMainStatusType;

typedef struct {
    WindowMainStatusType    mainStatus;
    WindowMainStatusType    mainStatus2;
    LockStatusType          lockStatus;
    PositionType            position;
    PositionType            targetPosition;
    PositionType            position2;
    PositionType            targetPosition2;
} WindowStatusType;
```

mainStatus2，position2，targetPosition2は，Windowが第二次元方向に移動　する機能を持つ場合のみ有効。

### (5) Windowの移動プロファイルのデータ型

```
typedef enum {
    WindowMoveProfileStandard = 0,  /* 標準的な移動 */
    WindowMoveProfileFast = 1,      /* 高速で移動 */
    WindowMoveProfileSlow = 2       /* 低速で移動 */
} WindowMoveProfileType;
```

### (6) Windowイベントのデータ型

```
typedef enum {
    WindowControlledBySelf = 1,     /* 自アプリケーションによる制御 */
    WindowControlledByOther = 2,    /* 他のアプリケーションによる制御 */
    WindowTargetReached = 3,        /* 目標位置到達，移動停止完了 */
    WindowFaultDetected = 4,        /* 故障検知 */
    WindowFaultRecovered = 5,       /* 故障回復 */
    WindowPinchDetected = 6,        /* 挟み込み検知 */
    WindowPinchRecovered = 7,       /* 挟み込み回復 */
    WindowOwnLockRevoked = 29,      /* 自アプリケーションのロックの強制解除 */
    WindowOtherLockReleased = 30,   /* 他のアプリケーション等によるロック解除 */
    WindowEventOverflow = 31,       /* イベントキューのオーバフロー */
```

```
    WindowConfigChanged = 32        /* 構成情報の変更 */
} WindowEventKindType;

typedef uint32_t WindowEventKindSetType;

typedef struct {
    IdType              instanceId;
    WindowEventKindType eventKind;
    union {
        struct {
            ApplicationIdType   sourceApplication;
        } controlledByOther;
        struct {
            ApplicationIdType   sourceApplication;
        } targetReached;
        struct {
            uint32_t    noLostEvents;
        } eventOverflow;
    } eventInfo;
} WindowEventType;
```

sourceApplicationは，eventKindがControlledByOtherかTargetReachedの場合， noLostEventsは，eventKindがEventOverflowの場合のみ有効。