#

@

初始化时，ATP读取来自CC data plug的[DataPlugContent](#DataPlugContent).CCTrainType信息，生成TrainType。

$

On initialization, ATP generates [TrainType](#TrainType) according to [DataPlugContent](#DataPlugContent).CCTrainType from the CC data plug.

%

def [TrainType](#TrainType)(k):

return [DataPlugContent](#DataPlugContent).CCTrainType

#

@

初始化时，ATP读取来自CC data plug的[DataPlugContent](#DataPlugContent).CCCoreId信息，生成CoreId。

$

On initialization, ATP generates [CoreId](#CoreId) according to [DataPlugContent](#DataPlugContent).[CCCoreId](#CCCoreId) read from the CC data plug.

%

def [CoreId](#CoreId)(k):

return [DataPlugContent](#DataPlugContent).CCCoreId

#

@

OtherCoreId，远端车头号

$

Core id for CC on the distant cab.

%

def [OtherCoreId](#OtherCoreId)(k):

if ([CoreId](#CoreId)(k) is **END\_1**):

return **END\_2**

elif ([CoreId](#CoreId)(k) is **END\_2**):

return **END\_1**

else:

return **None**

#

@

初始化时，ATP读取来自CC data plug的[DataPlugContent](#DataPlugContent).CC\_SSID信息，生成SubSystemId。

$

On initialization, ATP generates [SubSystemId](#SubSystemId) according to [DataPlugContent](#DataPlugContent).CC\_SSID from the CC data plug.

%

def [SubSystemId](#SubSystemId)(k):

return [DataPlugContent](#DataPlugContent).CC\_SSID

#

@

上述信息均获取正确并且相一致后，生成TrainKnown信息。如果TrainKnown为**False**，则VIOM将输出全限制状态。

$

After all above-mentioned information has corrected, and correspondingly, ATP will generate [TrainKnown](#TrainKnown) information. If [TrainKnown](#TrainKnown) considered as **False**, ATP shall set all output ports as restricted.

%

def [TrainKnown](#TrainKnown)(k):

return (([TrainType](#TrainType)(k) == [ATPsetting](#ATPsetting).TrainTypeId)

and ([CoreId](#CoreId)(k) is **END\_1**

or [CoreId](#CoreId)(k) is **END\_2**)

and [IdenticalVersionOfDualCPU](#IdenticalVersionOfDualCPU)(k))

#

@

NonVitalRequestReady，通过与CCNV的通信接口，判断是否收到CCNV的消息[NonVitalRequest](#NonVitalRequest)

$

Through the communication with CCNV, ATP judges [NonVitalRequest](#NonVitalRequest)received from CCNV and generates [NonVitalRequestReady](#NonVitalRequestReady) If received a new message.

%

def NonVitalRequestReady(k):

return Message.[Exists](#Exists)([NonVitalRequest](#NonVitalRequest))

#

@

ATOcontrolTimeValid，CCNV消息有效标志，如果超过**CCNV\_VALIDITY\_CYCLES**个周期仍未收到新的CCNV消息，则设置为**False**。

$

[ATOcontrolTimeValid](#ATOcontrolTimeValid) stands for the effectiveness of CCNV message. If there is no updating CCNV message past the **CCNV\_VALIDITY\_CYCLES**, [ATOcontrolTimeValid](#ATOcontrolTimeValid) is set as **False**.

%

def [ATOcontrolTimeValid](#ATOcontrolTimeValid)(k):

if ([NonVitalRequestReady](#NonVitalRequestReady)(k)):

[ATOcontrolTimeValid](#ATOcontrolTimeValid) = **True**

[ATOcontrolTimer](#ATOcontrolTimer) = 0

elif ([ATOcontrolTimer](#ATOcontrolTimer)(k-1) < CCNV\_VALIDITY\_CYCLES):

[ATOcontrolTimer](#ATOcontrolTimer) = [ATOcontrolTimer](#ATOcontrolTimer)(k-1) + 1

else:

[ATOcontrolTimeValid](#ATOcontrolTimeValid) = **False**

return [ATOcontrolTimeValid](#ATOcontrolTimeValid)

#

@

VIOM1VitalInputsReceived，ATP判断是否收到了来自VIOM1的安全输入消息。其中viomId取值为0或1，表示位于**END\_1**车头2个VIOM中的1个。

$

ATP determines whether received a safety input message from VIOM1. During the calculation, the value viomId is either zero or one, which represents one of the two VIOM in the train **END\_1**.

%

def [VIOM1VitalInputsReceived](#VIOM1VitalInputsReceived)(viomId, k):

return [Message.Received](#Received)([VIOM1VitalInput](#VIOM1VitalInput)(viomId), k)

#

@

VIOM1VitalInputsAvailable，通过通信接口，获取来自VIOM1的安全输入消息，并判断消息传输的时间有效性以及顺序的正确性。其中ViomId取值为0或1，表示位于**END\_1**车头2个VIOM中的1个。

$

Through the communication, ATP gets the vital input message from VIOM1 and decides the time effectiveness and the correctness of the sequence of the message, which defined as [VIOM1VitalInputsAvailable](#VIOM1VitalInputsAvailable). During the calculation, the value ViomId is either zero or one, which represents one of the two VIOM in the train **END\_1**.

%

def [VIOM1VitalInputsAvailable](#VIOM1VitalInputsAvailable)(ViomId, k):

return Message.[Available](#Available)([VIOM1VitalInputsReceived](#VIOM1VitalInputsReceived)(ViomId, k),

[VIOM1VitalInput](#VIOM1VitalInput)(ViomId).AtpLoopHour,

**VIOM\_VALIDITY\_TIME**,

[VIOM1VitalInputsLastAge](#VIOM1VitalInputsLastAge)(ViomId, k-1),

k)

#

@

VIOM1VitalInputsLastAge，记录收到最新的**END\_1**端VIOM的存活时间为多少。

$

Records the survival time of received vital inputs from VIOM1.

%

def [VIOM1VitalInputsLastAge](#VIOM1VitalInputsLastAge)(ViomId, k):

return Message.[LastAge](#LastAge)([VIOM1VitalInputsAvailable](#VIOM1VitalInputsAvailable)(ViomId, k),

[VIOM1VitalInput](#VIOM1VitalInput)(ViomId).AtpLoopHour,

[VIOM1VitalInputsLastAge](#VIOM1VitalInputsLastAge)(ViomId, k-1),

k)

#

@

LatestVIOM1LoopHourVIOM，记录当前收到最新的**END\_1**端VIOM的周期时间信息。初始化时LatestVIOM1LoopHourVIOM为VIOM周期号的最小值0；

如果收到可用的VIOM1信息，或之前的VIOM1消息已无效但又新收到一条VIOM1消息，则将相应的LatestVIOM1LoopHourVIOM设置为新收到消息中的viomLoopHour值；否则，LatestVIOM1LoopHourVIOM保持不变。

$

ATP records the latest cycle time information of VIOM in **END\_1** by the term [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM).In initialization, set [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM) as the zero; If receiving an available VIOM1 message, or a new message and the previous one has invalid, ATP will set the related value of [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM) as the viomLoopHour of the message. Otherwise, [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM) keeps unchanged.

%

def [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM)(ViomId, k):

if ([VIOM1VitalInputsAvailable](#VIOM1VitalInputsAvailable)(k)

or (not [VIOM1VitalInputsValid](#VIOM1VitalInputsValid)(k-1)

and [VIOM1VitalInputsReceived](#VIOM1VitalInputsReceived)(ViomId, k))):

return [VIOM1VitalInput](#VIOM1VitalInput)[ViomId].ViomLoopHour

else:

return [LatestVIOM1LoopHourVIOM](#LatestVIOM1LoopHourVIOM)[ViomId](k-1)

#

@

VIOM2VitalInputsReceived，收到并校验正确来自VIOM2的安全输出消息。

$

ATP determines whether received a safety input message from VIOM2.

%

def [VIOM2VitalInputsReceived](#VIOM2VitalInputsReceived)(ViomId, k):

return [Message.Received](#Received)([VIOM2VitalInput](#VIOM2VitalInput)(ViomId), k)

#

@

VIOM2VitalInputsAvailable，通过通信接口，获取来自VIOM2的安全输入消息，并判断消息传输的时间有效性以及顺序的正确性。其中ViomId取值为0或1，表示位于**END\_2**车头2个VIOM中的1个。

$

Through the communication, ATP gets the vital input message from VIOM2 and decides the time effectiveness and the correctness of the sequence of the message, which defined as [VIOM2VitalInputsAvailable](#VIOM2VitalInputsAvailable). During the calculation, the value i is either zero or one, which represents one of the two VIOM in the train **END\_2**.

%

def [VIOM2VitalInputsAvailable](#VIOM2VitalInputsAvailable)(ViomId, k):

return Message.[Available](#Available)([VIOM2VitalInputsReceived](#VIOM2VitalInputsReceived)(ViomId, k),

[VIOM2VitalInput](#VIOM2VitalInput)(ViomId).AtpLoopHour,

**VIOM\_VALIDITY\_TIME**,

[VIOM2VitalInputsLastAge](#VIOM2VitalInputsLastAge)(ViomId, k-1),

k)

#

@

VIOM2VitalInputsLastAge，记录当前收到最新的**END\_2**端VIOM的周期时间信息。

$

Records the survival time of received vital inputs from VIOM2.

%

def [VIOM2VitalInputsLastAge](#VIOM2VitalInputsLastAge)(viomId, k):

return Message.[LastAge](#LastAge)([VIOM2VitalInputsAvailable](#VIOM2VitalInputsAvailable)(viomId, k),

[VIOM2VitalInput](#VIOM2VitalInput)(viomId).AtpLoopHour,

[VIOM2VitalInputsLastAge](#VIOM2VitalInputsLastAge)(viomId, k-1),

k)

#

@

LatestVIOM2LoopHourVIOM，记录当前收到最新的**END\_2**端VIOM的周期时间信息。初始化时LatestVIOM2LoopHourVIOM为VIOM周期号的最小值0；

如果收到可用的VIOM2信息，或之前的VIOM2消息已无效但又新收到一条VIOM2消息，则将相应的LatestVIOM2LoopHourVIOM设置为新收到消息中的ViomLoopHour值；否则，LatestVIOM2LoopHourVIOM保持不变。

$

ATP records the latest cycle time information of VIOM in **END\_2**by the term [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM).In initialization, set [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM) as the zero; If receiving an available VIOM1 message, or a new message and the previous one has invalid, ATP will set the related value of [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM) as the viomLoopHour of the message.

Otherwise, [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM) keeps unchanged.

%

def [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM)(ViomId, k):

if ([VIOM2VitalInputsAvailable](#VIOM2VitalInputsAvailable)(k)

or (not [VIOM2VitalInputsValid](#VIOM2VitalInputsValid)(k-1)

and [VIOM2VitalInputsReceived](#VIOM2VitalInputsReceived)(ViomId, k))):

return [VIOM2VitalInput](#VIOM2VitalInput)[ViomId].ViomLoopHour

else:

return [LatestVIOM2LoopHourVIOM](#LatestVIOM2LoopHourVIOM)[ViomId](k-1)

#

@

VIOM1VitalInputsValid，判断来自**END\_1**的VIOM安全输入信息是否在有效时间内。

$

ATP determines whether the vital inputs message from VIOM1 valid.

%

def [VIOM1VitalInputsValid](#VIOM1VitalInputsValid)(k):

return (Message.[Valid](#Valid)([VIOM1VitalInput](#VIOM1VitalInput)(1).AtpLoopHour, **VIOM\_VALIDITY\_TIME**)

or Message.[Valid](#Valid)([VIOM1VitalInput](#VIOM1VitalInput)(2).AtpLoopHour, **VIOM\_VALIDITY\_TIME**))

#

@

VIOM2VitalInputsValid，判断来自**END\_2**的VIOM安全输入信息是否在有效时间内。

$

ATP determines whether the vital inputs message from VIOM2 valid.

%

def [VIOM2VitalInputsValid](#VIOM2VitalInputsValid)(k):

return (Message.[Valid](#Valid)([VIOM2VitalInput](#VIOM2VitalInput)(1).AtpLoopHour, **VIOM\_VALIDITY\_TIME**)

or Message.[Valid](#Valid)([VIOM2VitalInput](#VIOM2VitalInput)(2).AtpLoopHour, **VIOM\_VALIDITY\_TIME**))

#

@

CoupledByEnd1或CoupledByEnd2，列车两端连挂其他车辆。如果该项目未配置连挂输入的采集，则认为列车未与其他车连挂。其状态来自于项目可配置的列车输入采集。

$

[CoupledByEnd1](#CoupledByEnd1) or [CoupledByEnd2](#CoupledByEnd2) shows that both ends of train connect with other trains. If the project is not configured with the capture of coupling input, it is certain that the train does not connect with other trains.

%

def [CoupledByEnd1](#CoupledByEnd1)(k):

return [Offline](#Offline).[GetCoupledByEnd1](#GetCoupledByEnd1)(k)

def [CoupledByEnd2](#CoupledByEnd2)(k):

return [Offline](#Offline).[GetCoupledByEnd2](#GetCoupledByEnd2)(k)

#

@

TrainNotCoupled，列车未与其他车辆连挂。

%

def [TrainNotCoupled](#TrainNotCoupled)(k):

return [Offline](#Offline).[GetTrainNotCoupled](#GetTrainNotCoupled)(k)

#

@

TrainCoupledType，根据项目配置，获取当前列车的连挂类型。支持以下四种连挂类型：**TRAIN\_COUPLED\_UNKNOWN**，当前连挂状态无效；

**TRAIN\_NO\_COUPLED**，列车未连挂；**TRAIN\_COUPLED\_END1**，列车END\_1端连挂；**TRAIN\_COUPLED\_END2**，列车END\_2端连挂。规则如下：

%

def [TrainCoupledType](#TrainCoupledType)(k):

if ([TrainNotCoupled](#TrainNotCoupled)(k)

and not [TrainCoupledByEnd1](#TrainCoupledByEnd1)(k)

and not [TrainCoupledByEnd2](#TrainCoupledByEnd2)(k)):

return **TRAIN\_NO\_COUPLED**

elif (not [TrainNotCoupled](#TrainNotCoupled)(k)

and [TrainCoupledByEnd1](#TrainCoupledByEnd1)(k)

and not [TrainCoupledByEnd2](#TrainCoupledByEnd2)(k)):

return **TRAIN\_COUPLED\_END1**

elif (not [TrainNotCoupled](#TrainNotCoupled)(k)

and not [TrainCoupledByEnd1](#TrainCoupledByEnd1)(k)

and [TrainCoupledByEnd2](#TrainCoupledByEnd2)(k)):

return **TRAIN\_COUPLED\_END2**

else:

return **TRAIN\_COUPLED\_UNKNOWN**

#

@

ATPtime，维护本端ATP的loop hour时间。根据本端[CoreId](#CoreId)，初始化为**END\_1**或 **END\_2**的初始值；如果超过了相应的最大值，则重新等于初始化的值。否则每周期加1

$

[ATPtime](#ATPtime) stands for the ATP loop hour of this train END. Based on [CoreId](#CoreId), ATP initialize [ATPtime](#ATPtime) as the initiative value of **END\_1** or **END\_2**; If the value exceeds the maximum loop hour, ATP shall set it as the initiative value; Otherwise, add one for each cycle.

%

def [ATPtime](#ATPtime)(k):

if ([CoreId](#CoreId)(k) is **END\_1**):

if (Initialization):

return **CC1\_INIT\_TIME**

elif ([ATPtime](#ATPtime)(k-1) >= **CC1\_MAX\_TIME**):

return **CC1\_INIT\_TIME**

else:

return [ATPtime](#ATPtime)(k-1) + 1

else:

if (Initialization):

return **CC2\_INIT\_TIME**

elif ([ATPtime](#ATPtime)(k-1) >= **CC2\_MAX\_TIME**):

return **CC2\_INIT\_TIME**

else:

return [ATPtime](#ATPtime)(k-1) + 1

#

@

OtherATPmessageReceived，本周期收到冗余ATP消息并校验正确。

$

The message transmitted from the distant ATP in the other END shall be protected by check words. And before using the information, ATP shall verify the check words.

%

def [OtherATPmessageReceived](#OtherATPmessageReceived)(k):

return [Message.Received](#Received)([OtherCCsynchroReport](#OtherCCsynchroReport),k)

#

@

OtherATPmessageAvailable，判断来自冗余ATP消息的有效性：

$

[OtherATPmessageAvailable](#OtherATPmessageAvailable), ATP shall judge the effectiveness of message from the redundant ATP, shown as following pseudo-codes:

%

def [OtherATPmessageAvailable](#OtherATPmessageAvailable)(k):

return Message.[Available](#Available)([OtherATPmessageReceived](#OtherATPmessageReceived)(k),

[OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore,

**OTHER\_ATP\_VALIDITY\_TIME**,

[LastOtherATPmessageAge](#LastOtherATPmessageAge)(k-1),

k)

#

@

LastOtherATPmessageAge，获取到的远端ATP消息的存活时间。

%

def [LastOtherATPmessageAge](#LastOtherATPmessageAge)(k):

return Message.[LastAge](#LastAge)([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k),

[OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore,

[LastOtherATPmessageAge](#LastOtherATPmessageAge)(k-1),

k)

#

@

OtherATPmessageValid，接收到的冗余ATP消息是否在有效期内。如果该消息已失效，则设置OtherATPmessageValid为**False**；否则为**True**。

$

[OtherATPmessageValid](#OtherATPmessageValid) represents the effectiveness of the messages from redundant ATP. If this message is invalid, ATP will set [OtherATPmessageValid](#OtherATPmessageValid) as **False**; otherwise, it is set as **True**.

%

def [OtherATPmessageValid](#OtherATPmessageValid)(k):

return Message.[Valid](#Valid)([OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore,

**OTHER\_ATP\_VALIDITY\_TIME**,

k)

#

@

OtherATPminTime，本端ATP维护的冗余ATP的最小时间。设置规则如下：初始化时根据所在车头设置OtherATPminTime为默认值；否则，如果本周期收新的冗余ATP消息可用，则更新OtherATPminTime为消息中的currentTime；否则，如果冗余ATP消息仍然在有效期内，则对OtherATPminTime每周期加1，若越界则重新等于初始化值；否则，如果当前收到的新的冗余ATP消息（但不可用），则将OtherATPminTime更新为消息中的时间；其他情况，OtherATPminTime累加1，若越界则重新等于初始化值。

$

The [OtherATPminTime](#OtherATPminTime) stands for the local ATP maintained minimum time of

the redundant ATP. The setting rule is as following: In initialization, ATP set

the [OtherATPminTime](#OtherATPminTime) as default value based on the [CoreId](#CoreId) of the redundant

ATP. Or else:, if the updating message from the new redundant ATP in this

cycle is available, ATP will update [OtherATPminTime](#OtherATPminTime) as the current time in

the message. Or else:, if the redundant ATP message is still effective, ATP

will add 1 in the [OtherATPminTime](#OtherATPminTime) until it is out of bound, and set is as

initialization value. Or else:, If the received a new redundant ATP message,

but it was not available, ATP shall update [OtherATPminTime](#OtherATPminTime) as in the

message. Otherwise, accumulate [OtherATPminTime](#OtherATPminTime).

%

if ([CoreId](#CoreId) == **END\_1**)

if (Initialization)

[OtherATPminTime](#OtherATPminTime) = **CC2\_INIT\_TIME**

elif (([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

or (([OtherATPmessageValid](#OtherATPmessageValid)(k) == **False**)

and [Exists](#Exists)([OtherCCsynchroReport](#OtherCCsynchroReport))))

[OtherATPminTime](#OtherATPminTime) = [OtherCCsynchroReport](#OtherCCsynchroReport).CurrentTime

else:

if ([OtherATPminTime](#OtherATPminTime)(k-1) >= **CC2\_MAX\_TIME**)

[OtherATPminTime](#OtherATPminTime) = **CC2\_INIT\_TIME**

else:

[OtherATPminTime](#OtherATPminTime) = [OtherATPminTime](#OtherATPminTime)(k-1) + 1

else:

if (Initialization)

[OtherATPminTime](#OtherATPminTime) = **CC1\_INIT\_TIME**

elif (([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

or (([OtherATPmessageValid](#OtherATPmessageValid)(k) == **False**)

and [Exists](#Exists)([OtherCCsynchroReport](#OtherCCsynchroReport))))

[OtherATPminTime](#OtherATPminTime) = [OtherCCsynchroReport](#OtherCCsynchroReport).CurrentTime

else:

if ([OtherATPminTime](#OtherATPminTime)(k-1) >= **CC1\_MAX\_TIME**)

[OtherATPminTime](#OtherATPminTime) = **CC1\_INIT\_TIME**

else:

[OtherATPminTime](#OtherATPminTime) = [OtherATPminTime](#OtherATPminTime)(k-1) + 1

#

@

OtherATPmaxTime，维护冗余ATP的最大时间。

$

The [OtherATPmaxTime](#OtherATPmaxTime) stands for the local ATP maintained maximum time of the redundant ATP. The rules to update [OtherATPmaxTime](#OtherATPmaxTime) are similar with [OtherATPminTime](#OtherATPminTime) except that when received a new message from the redundant ATP, the [OtherATPmaxTime](#OtherATPmaxTime) shall add the maximum transmission delay in network.

%

if ([CoreId](#CoreId) == **END\_1**)

if (Initialization)

[OtherATPmaxTime](#OtherATPmaxTime) = **CC2\_INIT\_TIME**

elif (([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

or (([OtherATPmessageValid](#OtherATPmessageValid)(k) == **False**)

and [Exists](#Exists)([OtherCCsynchroReport](#OtherCCsynchroReport))))

[OtherATPmaxTime](#OtherATPmaxTime) = [OtherCCsynchroReport](#OtherCCsynchroReport).CurrentTime

+ LoopHourModularSub(ATPtime(k),

[OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore)

else:

if ([OtherATPmaxTime](#OtherATPmaxTime)(k-1) >= **CC2\_MAX\_TIME**)

[OtherATPmaxTime](#OtherATPmaxTime) = **CC2\_INIT\_TIME**

else:

[OtherATPmaxTime](#OtherATPmaxTime) = [OtherATPmaxTime](#OtherATPmaxTime)(k-1) + 1

else:

if (Initialization)

[OtherATPmaxTime](#OtherATPmaxTime) = **CC1\_INIT\_TIME**

elif (([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

or (([OtherATPmessageValid](#OtherATPmessageValid)(k) == **False**)

and [Exists](#Exists)([OtherCCsynchroReport](#OtherCCsynchroReport))))

[OtherATPmaxTime](#OtherATPmaxTime) = [OtherCCsynchroReport](#OtherCCsynchroReport).CurrentTime

+ LoopHourModularSub(ATPtime(k),

[OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore)

else:

if ([OtherATPmaxTime](#OtherATPmaxTime)(k-1) >= **CC1\_MAX\_TIME**)

[OtherATPmaxTime](#OtherATPmaxTime) = **CC1\_INIT\_TIME**

else:

[OtherATPmaxTime](#OtherATPmaxTime) = [OtherATPmaxTime](#OtherATPmaxTime)(k-1) + 1

#

@

OtherATP，解析并存储远端ATP的消息。初始化或者远端消息过期时，设置相应的值为默认状态；当本周期收到新的远端消息时，将其设置为新收到消息的值；否则，保持不变。

$

OtherATP, parse and store messages from the distant ATP.In initialization or the message has expired, set all variables as default value; when new message available, set the corresponding value from the new message; otherwise, remain unchanged.

%

def [OtherATP](#OtherATP)(k):

if (Initialization

or (not [OtherATPmessageValid](#OtherATPmessageValid)(k))):

[OtherATP](#OtherATP).LatestTimeOtherCore = **INVALID\_LOOP\_HOUR**

[OtherATP](#OtherATP).CoreId = **None**

[OtherATP](#OtherATP).BeaconId = **None**

[OtherATP](#OtherATP).EnableDoorOpening\_A = **False**

[OtherATP](#OtherATP).EnableDoorOpening\_B = **False**

[OtherATP](#OtherATP).PsdManagerOpeningOrder = **False**

[OtherATP](#OtherATP).PsdIdSide\_A = **None**

[OtherATP](#OtherATP).PsdValiditySide\_A = **None**

[OtherATP](#OtherATP).PsdClosedSide\_A = **False**

[OtherATP](#OtherATP).PsdIdSide\_B = **None**

[OtherATP](#OtherATP).PsdValiditySide\_B = **None**

[OtherATP](#OtherATP).PsdClosedSide\_B = **False**

[OtherATP](#OtherATP).ZcVersion = **None**

[OtherATP](#OtherATP).LocatedOnKnownPath = **False**

[OtherATP](#OtherATP).LocatedWithMemLocation = **False**

[OtherATP](#OtherATP).Location.Ext2 = **None**

[OtherATP](#OtherATP).Location.Uncertainty = **None**

[OtherATP](#OtherATP).Location.Ext1 = **None**

[OtherATP](#OtherATP).SleepZoneId = **None**

[OtherATP](#OtherATP).SleepZoneVersion = **None**

[OtherATP](#OtherATP).MotionSinceLastReloc = **None**

[OtherATP](#OtherATP).MotionSinceMemLoc = **None**

[OtherATP](#OtherATP).TrainFilteredStopped = **False**

[OtherATP](#OtherATP).SafetyParameterVersion = **None**

[OtherATP](#OtherATP).SafetyApplicationVersion = **None**

[OtherATP](#OtherATP).CC\_SSID = **None**

[OtherATP](#OtherATP).OverlapExpired = **False**

elif ([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k)):

[OtherATP](#OtherATP).LatestTimeOtherCore = [OtherCCsynchroReport](#OtherCCsynchroReport).LatestTimeOtherCore(k)

[OtherATP](#OtherATP).CoreId = [OtherCCsynchroReport](#OtherCCsynchroReport).CoreId

[OtherATP](#OtherATP).BeaconId = [OtherCCsynchroReport](#OtherCCsynchroReport).BeaconId

[OtherATP](#OtherATP).EnableDoorOpening\_A = [OtherCCsynchroReport](#OtherCCsynchroReport).EnableDoorOpening\_A

[OtherATP](#OtherATP).EnableDoorOpening\_B = [OtherCCsynchroReport](#OtherCCsynchroReport).EnableDoorOpening\_B

[OtherATP](#OtherATP).PsdManagerOpeningOrder = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdManagerOpeningOrder

[OtherATP](#OtherATP).PsdIdSide\_A = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdIdSide\_A

[OtherATP](#OtherATP).PsdValiditySide\_A = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdValiditySide\_A

[OtherATP](#OtherATP).PsdClosedSide\_A = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdClosedSide\_A

[OtherATP](#OtherATP).PsdIdSide\_B = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdIdSide\_B

[OtherATP](#OtherATP).PsdValiditySide\_B = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdValiditySide\_B

[OtherATP](#OtherATP).PsdClosedSide\_B = [OtherCCsynchroReport](#OtherCCsynchroReport).PsdClosedSide\_B

[OtherATP](#OtherATP).ZcVersion = [OtherCCsynchroReport](#OtherCCsynchroReport).ZcVersion

[OtherATP](#OtherATP).LocatedOnKnownPath = [OtherCCsynchroReport](#OtherCCsynchroReport).LocatedOnKnownPath

[OtherATP](#OtherATP).LocatedWithMemLocation = [OtherCCsynchroReport](#OtherCCsynchroReport).LocatedWithMemLocation

[OtherATP](#OtherATP).Location.Ext2 = [OtherCCsynchroReport](#OtherCCsynchroReport).Location.Ext2

[OtherATP](#OtherATP).Location.Uncertainty = [OtherCCsynchroReport](#OtherCCsynchroReport).Location.Uncertainty

[OtherATP](#OtherATP).Location.Ext1 = [OtherCCsynchroReport](#OtherCCsynchroReport).Location.Ext1

[OtherATP](#OtherATP).SleepZoneId = [OtherCCsynchroReport](#OtherCCsynchroReport).SleepZoneId

[OtherATP](#OtherATP).SleepZoneVersion = [OtherCCsynchroReport](#OtherCCsynchroReport).SleepZoneVersion

[OtherATP](#OtherATP).MotionSinceLastReloc = [OtherCCsynchroReport](#OtherCCsynchroReport).MotionSinceLastReloc

[OtherATP](#OtherATP).MotionSinceMemLoc = [OtherCCsynchroReport](#OtherCCsynchroReport).MotionSinceMemLoc

[OtherATP](#OtherATP).TrainFilteredStopped = [OtherCCsynchroReport](#OtherCCsynchroReport).TrainFilteredStopped

[OtherATP](#OtherATP).SafetyParameterVersion = [OtherCCsynchroReport](#OtherCCsynchroReport).SafetyParameterVersion

[OtherATP](#OtherATP).SafetyApplicationVersion = [OtherCCsynchroReport](#OtherCCsynchroReport).SafetyApplicationVersion

[OtherATP](#OtherATP).CC\_SSID = [OtherCCsynchroReport](#OtherCCsynchroReport).CC\_SSID

[OtherATP](#OtherATP).OverlapExpired = [OtherCCsynchroReport](#OtherCCsynchroReport).OverlapExpired

else:

pass

return [OtherATP](#OtherATP)

#

@

BlockModeUsed，当前是否现在选择BM模式。其状态来自于项目可配置的列车输入采集。

$

[BlockModeUsed](#BlockModeUsed) represents that either of train end chooses BM mode.

%

def [BlockModeUsed](#BlockModeUsed)(k):

return [Offline.GetBlockModeUsed](#GetBlockModeUsed)(k)

#

@

BMvariantValidWhileTemporallyValid，当前是否使用BM变量。其状态来自于项目可配置的列车输入采集。

$

The status of [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid) shows whether it is in the BM mode.

%

def [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k):

return [Offline.GetBMvariantValidWhileTemporallyValid](#GetBMvariantValidWhileTemporallyValid)(k)

#

@

BeaconVariantsUpdating，判断是否要更新BM变量。若本周期满足以下所有条件时，则认为需要更新BM变量，设置BeaconVariantsUpdating为**True**。当前使用BM变量（[BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)）；本周期未停车且收到信标消息且判断该信标带有BM变量；上周期列车未定位，或该BM信标方向与列车运营方向一致。否则，设置BeaconVariantsUpdating为**False**。

$

[BeaconVariantsUpdating](#BeaconVariantsUpdating) used to determine ATP whether to update the BM variants in this cycle. If all the following conditions are fulfilled, ATP shall set [BeaconVariantsUpdating](#BeaconVariantsUpdating) as **True**:The current operational mode is BLOCK MODE;And train moved and ATP received a BM beacon in this cycle; And the train is either not localized, or the direction of the BM variants is as same as the orientation of the train front end. Otherwise, ATP shall set [BeaconVariantsUpdating](#BeaconVariantsUpdating) as False.

%

def [BeaconVariantsUpdating](#BeaconVariantsUpdating)(k):

return ([BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)

and [**BeaconMessageReceive**](#BeaconMessageReceive)(k)

and [TrackMap](#TrackMap).[IsBmBeacon](#IsBmBeacon)([**BeaconMessage**](#BeaconMessage).ID)

and not [TrainFilteredStopped](#TrainFilteredStopped)(k)

and (not [TrainLocalized](#TrainLocalized)(k-1)

or ([**TrackMap**](#TrackMap).[BmBeaconDirection](#BmBeaconDirection)([**BeaconMessage**](#BeaconMessage).ID)== [TrainFrontOrientation](#TrainFrontOrientation)(k-1))))

#

@

BMbeaconReadAge，记录读取BM信标到当前的时间，默认值为**REPORT\_AGE\_MAX**。

如果BM信标变量无效，该值应被设置为默认值，BM信标变量无效的条件如下：初始化；或当前不在BM模式(not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid))；或[BMbeaconReadAge](#BMbeaconReadAge)已大于[ATPsetting](#ATPsetting).VariantsBMfullValidityTime；或本周期收到的BM信标（[BeaconVariantsUpdating](#BeaconVariantsUpdating)为**True**）中[DefaultMessage](#DefaultMessage)为**True**或[BlockModeVariantAvailable](#BlockModeVariantAvailable)为**False**；或本周期列车由定位转为失位状态；或当前使用的BM信标方向与已定位的列车运营方向[TrainFrontOrientation](#TrainFrontOrientation)不同。否则，如果本周期更新BM信标，则将该变量的初始值设置为1（因为ATP使用的是上个周期读到的信标信息）。其他情况，累加该变量。

%

def [BMbeaconReadAge](#BMbeaconReadAge)(k):

if (Initialization

or not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)

or [BMbeaconReadAge](#BMbeaconReadAge)(k-1) > ([ATPsetting](#ATPsetting).VariantsBMfullValidityTime - 1)

or ([BeaconVariantsUpdating](#BeaconVariantsUpdating)(k)

and ([DefaultMessage](#DefaultMessage)(k) or not [BlockModeVariantAvailable](#BlockModeVariantAvailable)(k)))

or ([TrainLocalized](#TrainLocalized)(k-1)

and (not [TrainLocalized](#TrainLocalized)(k)

or TrackMap.BmBeaconDirection([UsedBMbeaconId](#UsedBMbeaconId)(k-1))

is not [TrainFrontOrientation](#TrainFrontOrientation)(k-1)))):

return **REPORT\_AGE\_MAX**

elif ([BeaconVariantsUpdating](#BeaconVariantsUpdating)(k)):

return 1

else:

return [BMbeaconReadAge](#BMbeaconReadAge)(k-1) + 1

#

@

BMbeaconVariantValue，获取来自BM信标中该变量的值，输入索引和周期，若过期为假值.

%

def [BMbeaconVariantValue](#BMbeaconVariantValue)(lineSection, VarIndex, k):

if ([BMbeaconReadAge](#BMbeaconReadAge)(k) > [ATPsetting](#ATPsetting).VariantsBMfullValidityTime):

return **False**

else:

for Var in [range](#range)(0, MAX\_BM\_VARIANT\_NB):

if ([BMbeaconVariants](#BMbeaconVariants)[Var].LineSection == LineSection

and [BMbeaconVariants](#BMbeaconVariants)[Var].Index == VarIndex):

return [BMbeaconVariants](#BMbeaconVariants)[Var].Value

else:

continue

else:

return **False**

#

@

UsedBMbeaconId用于记录当前所使用的BM变量来自哪个BM信标，判断条件如下：当初始化，非使用BM变量（not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)），该信标方向与当前车头方向不符，或列车失位时，清除UsedBMbeaconId；否则，如果收到有效的BM信标，记录该信标id到UsedBMbeaconId；否则，保持UsedBMbeaconId不变。

$

[UsedBMbeaconId](#UsedBMbeaconId) records the used BM variants came from which BM beacon: When one of the following conditions fulfilled, ATP clear the [UsedBMbeaconId](#UsedBMbeaconId): initialization,the BLOCK MODE variant is not temporally valid, the direction of the used BM beacon is not as same as train front orientation, the train is not localized. Or else:, when received a valid BM beacon, ATP update [UsedBMbeaconId](#UsedBMbeaconId); Otherwise, keep this value unchanged.

%

def [UsedBMbeaconId](#UsedBMbeaconId)(k):

if ([BeaconVariantsUpdating](#BeaconVariantsUpdating)(k)):

return [BeaconMessage](#BeaconMessage).Id

elif (Initialization

or not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)

or ([**TrainLocalized**](#TrainLocalized)(k-1)

and ([**TrackMap**](#TrackMap).[BmBeaconDirection](#BmBeaconDirection)([UsedBMbeaconId](#UsedBMbeaconId)(k-1))

is not [TrainFrontOrientation](#TrainFrontOrientation)(k-1)))

or ([**TrainLocalized**](#TrainLocalized)(k-1) and not [TrainLocalized](#TrainLocalized)(k))):

return **None**

else:

return [UsedBMbeaconId](#UsedBMbeaconId)(k-1)

#

@

BMvariantValidLastRisingAge, 记录从选择BM模式到当前经过的时间

%

def [BMvariantValidLastRisingAge](#BMvariantValidLastRisingAge)(k):

if (not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)):

[BMvariantValidLastRisingAge](#BMvariantValidLastRisingAge) = 0

else:

[BMvariantValidLastRisingAge](#BMvariantValidLastRisingAge) = [BMvariantValidLastRisingAge](#BMvariantValidLastRisingAge)(k-1) + 1

#

@

CBIvariantReportReceived，ATP软件收到CCNV转发的“CBI variant report”消息，并安全校核字校验正确。

%

def [CBIvariantReportReceived](#CBIvariantReportReceived)(cbi, k):

return [Message.Received](#Received)([CBIvariantReport](#CBIvariantReport)(cbi), k)

#

@

CBIvariantReportAvailable，联锁消息可用

%

def [CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbi, k):

return Message.[Available](#Available)([CBIvariantReportReceived](#CBIvariantReportReceived)(cbi, k),

[CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour,

[ATPsetting](#ATPsetting).VariantsBMlowValidityTime,

[min](#min)([CBIvariantReportLastAge](#CBIvariantReportLastAge)(cbi, k-1),

[CBIminProductionAge](#CBIminProductionAge)(cbi, k-1),

[BMvariantValidLastRisingAge](#BMvariantValidLastRisingAge)(k)),

k)

#

@

CBIvariantReportLastAge，记录最新收到的联锁消息已存活的时间。

%

def [CBIvariantReportLastAge](#CBIvariantReportLastAge)(cbi, k):

return Message.[LastAge](#LastAge)([CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbi, k),

[CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour,

[CBIvariantReportLastAge](#CBIvariantReportLastAge)(cbi, k-1),

k)

#

@

BMcbiVariants, 当来自CBI的变量可用时，存储CBI变量；其他时候保持不变。

%

def [BMcbiVariants](#BMcbiVariants)(cbi, k):

if ([CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbi, k)):

for idx in [range](#range)(0, [CBIvariantReport](#CBIvariantReport).NumberOrVariants):

[BMcbiVariants](#BMcbiVariants)[cbi].Variants[idx] = [CBIvariantReport](#CBIvariantReport).Variant[idx]

else:

pass

return [BMcbiVariants](#BMcbiVariants)

#

@

CBIvariantAge，CBI变量的有效存活时间，最大值为**REPORT\_AGE\_MAX**。

该值与CBIvariantReportLastAge的区别是在判断回复远端ATP消息时，使用

[OtherATPmaxTime](#OtherATPmaxTime)进行计算，在判断有效期时导向安全侧。

%

def [CBIvariantAge](#CBIvariantAge)(cbi, k):

if (Initialization

or [CBIvariantAge](#CBIvariantAge)(k-1) >= **REPORT\_AGE\_MAX**):

return **REPORT\_AGE\_MAX**

elif ([CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbi, k)

and Message.[ReplyLocalCC](#ReplyLocalCC)([CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour)):

return (1 + Message.[ModularSub](#ModularSub)([ATPtime](#ATPtime)(k), [CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour))

elif ([CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbi, k)

and Message.[ReplyDistantCC](#ReplyDistantCC)([CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour)):

return (1 + Message.[ModularSub](#ModularSub)([OtherATPmaxTime](#OtherATPmaxTime)(k), [CBIvariantReport](#CBIvariantReport)(cbi).CcLoopHour))

else:

return (1 + [CBIvariantAge](#CBIvariantAge)(cbi, k-1))

#

@

CBIvariantLowValidity，判断是否在CBI无线的短有效期内，用于PZ的监控。 在CBTC或者使用来自BM信标变量的情况下，该值为真。

%

def [CBIvariantLowValidity](#CBIvariantLowValidity)(cbi, k):

if (not [BlockModeUsed](#BlockModeUsed)(k)

or not [CBIvariantMoreAvailableThanBeacon](#CBIvariantMoreAvailableThanBeacon)(cbi, k)

or [CBIvariantAge](#CBIvariantAge)(cbi, k) <= [ATPsetting](#ATPsetting).VariantsBMlowValidityTime):

return **True**

else:

return **False**

#

@

BMcbiVariantValue，根据联锁变量索引，获得CBI的变量。

%

def [BMcbiVariantValue](#BMcbiVariantValue)(CbiId, VarIndex, k):

if ([CBIvariantAge](#CBIvariantAge)(CbiId, k) > [ATPsetting](#ATPsetting).VariantsBMfullValidityTime):

return **False**

else:

return [BMcbiVariants](#BMcbiVariants)[CbiId].Variants[VarIndex]

#

@

AppliedCBIvariantLoopHour，记录当前使用的CBI的变量的CC时间，供CCNV使用。

%

def [AppliedCBIvariantLoopHour](#AppliedCBIvariantLoopHour)(cbiId, k):

if ([CBIvariantReportAvailable](#CBIvariantReportAvailable)(cbiId, k)):

return [CBIvariantReport](#CBIvariantReport)(cbiId).CcLoopHour

else:

return [AppliedCBIvariantLoopHour](#AppliedCBIvariantLoopHour)(cbiId, k-1)

#

@

CBIvariantMoreAvailableThanBeacon，通过比较最后一次收到的BM信标的有效期，和对应变量所在该联锁区的无线变量，判断对于该变量，是使用来自CI无线的变量而非来自信标的变量。

$

ATP shall use the more recent message from beacons and CBI radio.

%

def [CBIvariantMoreAvailableThanBeacon](#CBIvariantMoreAvailableThanBeacon)(CbiId, k):

if (Initialization

or not [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k-1)

or not [ATPsetting](#ATPsetting).BlockModeThroughRadio(k)):

return **False**

else:

return ([UsedBMbeaconId](#UsedBMbeaconId)(k) is **None**

or ([CBIvariantAge](#CBIvariantAge)(CbiId, k) <= [ATPsetting](#ATPsetting).VariantsBMfullValidityTime

and ([CBIvariantReportLastAge](#CBIvariantReportLastAge)(CbiId, k)

<= [BMbeaconReadAge](#BMbeaconReadAge)(k) + [ATPsetting](#ATPsetting).VariantsBMradioPriorityDelay)

and ([CBIvariantReportLastAge](#CBIvariantReportLastAge)(CbiId, k) <= [CBIminProductionAge](#CBIminProductionAge)(CbiId, k))))

#

@

BMvariantValue，统一来自BM信标和CBI无线的BM变量

%

def [BMvariantValue](#BMvariantValue)(Variant, k):

if ([CBIvariantMoreAvailableThanBeacon](#CBIvariantMoreAvailableThanBeacon)(Variant.Cbi.Id, k)):

return [BMcbivariantValue](#BMcbivariantValue)(Variant.Cbi.Id, Variant.Cbi.Index, k)

else:

return [BMbeaconVariantValue](#BMbeaconVariantValue)(Variant.LineSec.Id, Variant.LineSec.Index, k)

#

@

BMvariantRemainingTime，BM变量的剩余有效期

%

def [BMvariantRemainingTime](#BMvariantRemainingTime)(cbi, k):

if (not [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)):

return 0

elif ([CBIvariantMoreAvailableThanBeacon](#CBIvariantMoreAvailableThanBeacon)(cbi, k)):

return [max](#max)(0, [ATPsetting](#ATPsetting).VariantsBMfullValidityTime - [CBIvariantAge](#CBIvariantAge)(cbi, k))

else:

return [max](#max)(0, [ATPsetting](#ATPsetting).VariantsBMfullValidityTime - [BMbeaconReadAge](#BMbeaconReadAge)(k))

#

@

VersionAuthorizationReceived，收到版本授权

%

def [VersionAuthorizationReceived](#VersionAuthorizationReceived)(lcId, k):

return [Message.Received](#Received)([VersionAuthorization](#VersionAuthorization)(lcId), k)

#

@

VersionAuthorizationAvailable，LC版本授权消息可用

%

def VersionAuthorizationAvailable(lcId, k):

return Message.[Available](#Available)([VersionAuthorizationReceived](#VersionAuthorizationReceived)(lcId, k),

[VersionAuthorization](#VersionAuthorization)(lcId).CcLoopHour,

[ATPsetting](#ATPsetting).VersionsValidityTime,

[LastVersionReportAge](#LastVersionReportAge)(lcId, k-1),

k)

#

@

LastVersionReportAge，记录从上次收到LC的版本信息到现在的时间。

%

def [LastVersionReportAge](#LastVersionReportAge)(lcId, k):

return Message.[LastAge](#LastAge)([VersionAuthorizationAvailable](#VersionAuthorizationAvailable)(lcId, k),

[VersionAuthorization](#VersionAuthorization)(lcId).CcLoopHour,

[LastVersionReportAge](#LastVersionReportAge)(lcId, k-1),

k)

#

@

ReceivedVersionMessages，用于存储从LC收到的**MAX\_ZC\_NB**个ZC区的授权信息。由于每个ZC分属不同的LC管理，因此收到特定的LC消息时应仅更新其所对应ZC的版本授权状态。

%

def [ReceivedVersionMessages](#ReceivedVersionMessages)(LcId, k):

if (Initialization):

ReceivedVersionMessages = **None**

elif ([VersionAuthorizationAvailable](#VersionAuthorizationAvailable)(LcId, k)):

if (Message.[ReplyLocalCC](#ReplyLocalCC)([VersionAuthorization](#VersionAuthorization)(LcId).CcLoopHour)):

NewValidity = ([VersionAuthorization](#VersionAuthorization)(LcId).CcLoopHour

+ [ATPsetting](#ATPsetting).VersionsValidityTime)

else:

NewValidity

= ([ATPtime](#ATPtime)(k) + [ATPsetting](#ATPsetting).VersionsValidityTime

- ([OtherATPmaxTime](#OtherATPmaxTime)(k) - [VersionAuthorization](#VersionAuthorization)(LcId).CcLoopHour))

for ZcId in [range](#range)(0, **MAX\_ZC\_NB**):

if ([TrackMap](#TrackMap).Zc[ZcId].LcId == LcId):

[ReceivedVersionMessages](#ReceivedVersionMessages)[ZcId].VitalAuthorization

= [VersionAuthorization](#VersionAuthorization)(LcId).VitalAuthorization[ZcId]

[ReceivedVersionMessages](#ReceivedVersionMessages)[ZcId].ValidityTime = NewValidity

else:

pass

else:

[ReceivedVersionMessages](#ReceivedVersionMessages) = [ReceivedVersionMessages](#ReceivedVersionMessages)(k-1)

return [ReceivedVersionMessages](#ReceivedVersionMessages)

#

@

VersionAuthorizedByLC，获取ZC的版本授权状态

%

def [VersionAuthorizedByLC](#VersionAuthorizedByLC)(ZcId, k):

if (Message.IsMoreRecent

([ReceivedVersionMessages](#ReceivedVersionMessages)([TrackMap](#TrackMap).Zc[ZcId].LcId ,k)[zcId].ValidityTime,

[ATPtime](#ATPtime)(k))):

return [ReceivedVersionMessages](#ReceivedVersionMessages)([TrackMap](#TrackMap).Zc[ZcId].LcId, k)[zcId].VitalAuthorization

else:

return **False**

#

@

SameVersionWithDistantCore，比较来自远端ATP的安全软件，项目配置数据，以及线路地图版本号与本端是否一致

$

The local ATP shall compare the information from the redundant ATP to ensure the consistency, which includes versions of vital software, project configuration data and the track map.

%

def [SameVersionWithDistantCore](#SameVersionWithDistantCore)(k):

if ([OtherATPmessageValid](#OtherATPmessageValid)(k)

and ([OtherATP](#OtherATP).SafetyParameterVersion == [ATPsetting](#ATPsetting).SafetyParameterVersion)

and ([OtherATP](#OtherATP).SafetyApplicationVersion == [SafeApplicationVersion](#SafeApplicationVersion))):

for ZcId in [range](#range)(0, **MAX\_ZC\_NB**):

if ([OtherATP](#OtherATP).ZcVersion[ZcId] != [TrackMap](#TrackMap).[ZC](#ZC)[ZcId].Version):

return **False**

else:

continue

else:

return **True**

else:

return **False**

#

@

EOAReportReceived，收到EOA消息

%

def [EOAReportReceived](#EOAReportReceived)(k):

return [Message.Received](#Received)([EOAReport](#EOAReport), k)

#

@

ZCmessageReady，表示本周期收到了有效的来自ZC的EOA和变量消息。当前时间大于消息中的ccLoopHour；消息中的ccLoopHour+EOA有效期，应大于当前时间。

$

[ZCmessageReady](#ZCmessageReady) represents that an available EOA and variants message from [ZC](#ZC) received in this cycle.

%

def [ZCmessageReady](#ZCmessageReady)(k):

return (Message.[Available](#Available)([EOAReportReceived](#EOAReportReceived)(k),

[EOA\_Report](#EOA_Report).CcLoopHour,

[ATPsetting](#ATPsetting).EOAvalidityTime,

[LastEOAReportAge](#LastEOAReportAge)(k-1),

k)

and ([VersionAuthorizedByLC](#VersionAuthorizedByLC)([SSIDofZC](#SSIDofZC), k))

and (Message.[ReplyLocalCC](#ReplyLocalCC)([EOA\_Report](#EOA_Report).CcLoopHour)

or [SameVersionWithDistantCore](#SameVersionWithDistantCore)(k)))

#

@

LastEOAReportAge，数值型，上次发出loc-report的周期数减去EOA在ZC端消耗的时间（CC周期数）。

$

[LastEOAReportAge](#LastEOAReportAge) represents the value calculated by current ATP time minus the previous loc-report number and the EOA consuming time in [ZC](#ZC).

%

def [LastEOAReportAge](#LastEOAReportAge)(k):

return Message.[LastAge](#LastAge)([ZCmessageReady](#ZCmessageReady)(k),

[EOA\_Report](#EOA_Report).CcLoopHour,

[LastEOAReportAge](#LastEOAReportAge)(k-1),

k)

#

@

EOAgroundAge，数值型，在收到EOA消息时，其时间已经消耗了几个CC的周期。需同时维护WithoutSpaceEoa和普通EOA。

$

[EOAgroundAge](#EOAgroundAge) stands for the number of CC cycle when receiving the EOA information.

%

def [EOAgroundAge](#EOAgroundAge)(k):

if (Initialization):

[EOAgroundAge](#EOAgroundAge).WithoutSpacing = **REPORT\_AGE\_MAX**

[EOAgroundAge](#EOAgroundAge).Classic = **REPORT\_AGE\_MAX**

elif ([ZCmessageReady](#ZCmessageReady)(k)):

[EOAgroundAge](#EOAgroundAge).WithoutSpacing = ([round.ceil](#roundceil)

(([EOA\_Report](#EOA_Report).MessageContainerCreationTime

- [EOA\_Report](#EOA_Report).WithoutSpacingEoaCreationTime)

\* SYNCHRODATE\_TIME\_UNIT\_MS / ATP\_CYCLE\_TIME\_MS))

[EOAgroundAge](#EOAgroundAge).Classic = [round.ceil](#roundceil)(([EOA\_Report](#EOA_Report).MessageContainerCreationTime

- [EOA\_Report](#EOA_Report).EoaCreationTime)

\* SYNCHRODATE\_TIME\_UNIT\_MS / ATP\_CYCLE\_TIME\_MS)

else:

[EOAgroundAge](#EOAgroundAge) = [EOAgroundAge](#EOAgroundAge)(k-1)

return [EOAgroundAge](#EOAgroundAge)

#

@

ReceivedEOAreport，判断当新收到EOA消息的有效期大于之前存储EOA消息有效期时，更新EOA。需同时维护WithoutSpaceEoa和普通EOA。当存储的EOA消息过期后，清除该消息。

%

def [ReceivedEOAreport](#ReceivedEOAreport)(k):

if (Initialization):

[ReceivedEOAreport](#ReceivedEOAreport) = **None**

elif ([ZCmessageReady](#ZCmessageReady)(k)):

[ReceivedEOAreport](#ReceivedEOAreport).[TrainFrontEnd](#TrainFrontEnd) = [EOA\_Report](#EOA_Report).[TrainFrontEnd](#TrainFrontEnd)

[ReceivedEOAreport](#ReceivedEOAreport).Classic = [UpdateReceivedEoa](#UpdateReceivedEoa)([EOA\_Report](#EOA_Report).CcLoopHour,

[EOAgroundAge](#EOAgroundAge)(k).Classic,

[EOA\_Report](#EOA_Report).Classic,

[ReceivedEOAreport](#ReceivedEOAreport)(k-1).Classic)

[ReceivedEOAreport](#ReceivedEOAreport).WithoutSpacing = ([UpdateReceivedEoa](#UpdateReceivedEoa)

([EOA\_Report](#EOA_Report).CcLoopHour,

[EOAgroundAge](#EOAgroundAge)(k).WithoutSpacing,

[EOA\_Report](#EOA_Report).WithoutSpacing,

[ReceivedEOAreport](#ReceivedEOAreport)(k-1).WithoutSpacing))

[ReceivedEOAreport](#ReceivedEOAreport).CcLoopHour = [EOA\_Report](#EOA_Report).CcLoopHour

else:

[ReceivedEOAreport](#ReceivedEOAreport) = [ReceivedEOAreport](#ReceivedEOAreport)(k-1)

if ([Message.IsMoreRecent](#IsMoreRecent)([ATPtime](#ATPtime)(k), [ReceivedEOAreport](#ReceivedEOAreport).Classic.ValidityTime)):

clean\_reseived\_eoa\_classic

if ([Message.IsMoreRecent](#IsMoreRecent)([ATPtime](#ATPtime)(k), [ReceivedEOAreport](#ReceivedEOAreport).WithoutSpacing.ValidityTime)):

clean\_reseived\_eoa\_without\_space

return [ReceivedEOAreport](#ReceivedEOAreport)

def [UpdateReceivedEoa](#UpdateReceivedEoa)(NewEoaLoopHour, EoaGroundAge, NewReceivedEoa, PreviousReceivedEoa):

if ([Message.ReplyLocalCc](#ReplyLocalCc)(NewEoaLoopHour)):

NewValidity = (NewEoaLoopHour - EoaGroundAge + [ATPsetting](#ATPsetting).EOAvalidityTime)

else:

NewValidity = ([ATPtime](#ATPtime)(k) - EoaGroundAge + [ATPsetting](#ATPsetting).EOAvalidityTime

- ([OtherATPmaxTime](#OtherATPmaxTime)(k) - NewEoaLoopHour))

if ([Message.IsMoreRecent](#IsMoreRecent)(NewValidity, [ATPtime](#ATPtime)(k))

and ([Message.IsMoreRecent](#IsMoreRecent)(NewValidity, PreviousReceivedEoa.ValidityTime))):

return NewReceivedEoa

else:

return PreviousReceivedEoa

#

@

VariantGroundAge，将ZC端的变量生存时间转换为CC周期数

$

[VariantGroundAge](#VariantGroundAge) shows the survival time of the variants in [ZC](#ZC).

%

def [VariantGroundAge](#VariantGroundAge)(lineSec, k):

[VariantGroundAge](#VariantGroundAge) = [round.ceil](#roundceil)(([EOA\_Report](#EOA_Report).MessageContainerCreationTime

- [VariantReport](#VariantReport)(lineSec).CreationTime)

\* SYNCHRODATE\_TIME\_UNIT\_MS / ATP\_CYCLE\_TIME\_MS)

return [VariantGroundAge](#VariantGroundAge)

#

@

VariantReportReceived，收到ZC变量消息

%

def [VariantReportReceived](#VariantReportReceived)(LineSec, k):

return [Message.Received](#Received)([VariantReport](#VariantReport)(LineSec), k)

#

@

ReceivedVariantReport，存储来自ZC的变量消息，如ST\_VARIANT\_RCV所示，按照LineSection进行存储：

%

def [ReceivedVariantReport](#ReceivedVariantReport)(LineSec, k):

if ([ZCmessageReady](#ZCmessageReady)(k)

and [VariantReportReceived](#VariantReportReceived)(LineSec, k)):

if (Message.ReplyLocalCc([ReceivedEOAreport](#ReceivedEOAreport)(k).CcLoopHour)):

NewValidity = ([ReceivedEOAreport](#ReceivedEOAreport)(k).CcLoopHour

- [VariantGroundAge](#VariantGroundAge)(LineSec, k)

+ [ATPsetting](#ATPsetting).VariantsCBTCvalidityTime)

else:

NewValidity = ([ATPtime](#ATPtime)(k) - [VariantGroundAge](#VariantGroundAge)(LineSec, k)

+ [ATPsetting](#ATPsetting).VariantsCBTCvalidityTime

- ([OtherATPmaxTime](#OtherATPmaxTime)(k) - [ReceivedEOAreport](#ReceivedEOAreport)(k).CcLoopHour))

if (Message.IsMoreRecent(NewValidity, [ATPtime](#ATPtime)(k))

and ([Message.IsMoreRecent](#IsMoreRecent)

(NewValidity, [ReceivedVariantReport](#ReceivedVariantReport)[LineSec](k-1).ValidityTime))):

[ReceivedVariantReport](#ReceivedVariantReport)[LineSec].ValidityTime = NewValidity

[ReceivedVariantReport](#ReceivedVariantReport)[LineSec].Variants = [VariantReport](#VariantReport)(LineSec, Variants)

else:

[ReceivedVariantReport](#ReceivedVariantReport)[LineSec] = [ReceivedVariantReport](#ReceivedVariantReport)[LineSec](k-1)

else:

[ReceivedVariantReport](#ReceivedVariantReport)[LineSec] = [ReceivedVariantReport](#ReceivedVariantReport)[LineSec](k-1)

return [ReceivedVariantReport](#ReceivedVariantReport)

#

@

CBTCvariantValue，维护CBTC下变量的值。如果变量有效期大于当前时间，则使用该变量；否则为限制状态。

$

ATP shall maintain the validation of CBTC variants message from ZC. if the validation timeout, ATP should set all CBTC variants to restrictive state.

%

def [CBTCvariantValue](#CBTCvariantValue)(Variant, k):

if ([ReceivedVariantReport](#ReceivedVariantReport)(Variant.LineSec.Id, k).ValidityTime > [ATPtime](#ATPtime)(k)):

return [ReceivedVariantReport](#ReceivedVariantReport)(Variant.LineSec.Id, k).Status(Variant.LineSec.Index)

else:

return **False**

#

@

VariantValue，统一CBTC和BM下的变量

%

def [VariantValue](#VariantValue)(Variant, k):

if ([BlockModeUsed](#BlockModeUsed)(k)):

return [BMvariantValue](#BMvariantValue)(Variant, k)

else:

return [CBTCvariantValue](#CBTCvariantValue)(Variant, k)

#

@

BeaconCount，ATP记录从上电开始，到当前周期共收到多少次Top-loc信号。

$

[BeaconCount](#BeaconCount) represents the accumulated number of received Top-loc signal from power on to current cycle.

%

if (Initialization)

[BeaconCount](#BeaconCount) = 0

else:

[BeaconCount](#BeaconCount) = [LockedTopLocCounter](#LockedTopLocCounter)(k) + [BeaconCount](#BeaconCount)(k-1)

#

@

如果本周期[BeaconMessageReceive](#BeaconMessageReceive)为**True**，则从[LockedBeaconMsgByte](#LockedBeaconMsgByte)中获取信标ID，设置BeaconMessage.ID；其他情况保持不变。

$

If the status of [BeaconMessageReceive](#BeaconMessageReceive) is **True**, the [BeaconMessage](#BeaconMessage).ID is obtained by [LockedBeaconMsgByte](#LockedBeaconMsgByte); Otherwise, keep it unchanged.

%

if (Initialization)

[BeaconMessage](#BeaconMessage).ID = 0

elif ([BeaconMessageReceive](#BeaconMessageReceive)(k))

[BeaconMessage](#BeaconMessage).ID = [LockedBeaconMsgByte](#LockedBeaconMsgByte)[**BEACON\_ID\_BITS**]

else:

[BeaconMessage](#BeaconMessage).ID = [BeaconMessage](#BeaconMessage).ID(k-1)

#

@

如果本周期[BeaconMessageReceive](#BeaconMessageReceive)为**True**，则从[LockedBeaconMsgByte](#LockedBeaconMsgByte)中获取变量数据，设置数组BeaconMessage.Variants[**MAX\_BM\_VARIANT\_NB**]；若本周期未读到新的信标则保持不变。其中**BM\_VARIANTS\_BIT\_0**...**BM\_VARIANTS\_BIT\_15**表示[REF4]中定义的信标消息中表示BM信标变量的位数。

$

If the [BeaconMessageReceive](#BeaconMessageReceive) is **True**, the variants is come from [LockedBeaconMsgByte](#LockedBeaconMsgByte)and ATP set as [BeaconMessage](#BeaconMessage).Variants[**MAX\_BM\_VARIANT\_NB**]; if there is no beacon read at the end of cycle, there is no changes. **BM\_VARIANTS\_BIT\_0**...**BM\_VARIANTS\_BIT\_15** represents the index of BM beacon variants defined in [REF4].

%

if (Initialization)

[BeaconMessage](#BeaconMessage).Variants = {0,..,0}

elif ([BeaconMessageReceive](#BeaconMessageReceive)(k))

[BeaconMessage](#BeaconMessage).Variants(k)

= {[LockedBeaconMsgByte](#LockedBeaconMsgByte)[**BM\_VARIANTS\_BIT\_0**],

...,

[LockedBeaconMsgByte](#LockedBeaconMsgByte)[**BM\_VARIANTS\_BIT\_15**]}

else:

[BeaconMessage](#BeaconMessage).Variants = [BeaconMessage](#BeaconMessage).Variants(k-1)

#

@

如果本周期[BeaconMessageReceive](#BeaconMessageReceive)为**True**，则从[LockedBeaconMsgByte](#LockedBeaconMsgByte)中判断是否默认消息，设置[BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage)；若本周期未读到新的信标则保持不变。其中**DEFAULT\_MESSAGE\_BIT**表示[REF4]中定义的信标消息中表示信标是否为默认消息的位数。

$

If the [BeaconMessageReceive](#BeaconMessageReceive) is **True**, the default message is judged by [LockedBeaconMsgByte](#LockedBeaconMsgByte)and ATP set the [BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage); if there is no new beacon read, it keeps unchanged. **DEFAULT\_MESSAGE\_BIT** represents the index of beacon that judges default message, which defined in the [REF4].

%

if (Initialization)

[BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage) = **False**

elif ([BeaconMessageReceive](#BeaconMessageReceive)(k))

[BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage)(k)

= [LockedBeaconMsgByte](#LockedBeaconMsgByte)[**DEFAULT\_MESSAGE\_BI**T]

else:

[BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage) = [BeaconMessage](#BeaconMessage).[DefaultMessage](#DefaultMessage)(k-1)

#

@

如果本周期[BeaconMessageReceive](#BeaconMessageReceive)为**True**，则从[LockedBeaconMsgByte](#LockedBeaconMsgByte)中判断变量是否可用信息，设置[BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable)；若本周期未读到新的信标则保持不变。其中**BLOCK\_MODE\_VARIANT\_AVAILABLE\_BIT**表示[REF4]中定义的信标消息中表示信标所带变量是否可用的位数。

$

If the [BeaconMessageReceive](#BeaconMessageReceive) is **True**, it is feasible to judge whether the variants are available through [LockedBeaconMsgByte](#LockedBeaconMsgByte) and ATP set as [BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable); If there is no new beacon read, it keeps invariable. **BLOCK\_MODE\_VARIANT\_AVAILABLE\_BIT** stands for the index of the beacon variants in the beacon message defined in [REF4].

%

if (Initialization)

[BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable) = **False**

elif ([BeaconMessageReceive](#BeaconMessageReceive)(k))

[BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable)(k)

= [LockedBeaconMsgByte](#LockedBeaconMsgByte)[**BLOCK\_MODE\_VARIANT\_AVAILABLE\_BIT**]

else:

[BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable)(k)

= [BeaconMessage](#BeaconMessage).[BlockModeVariantAvailable](#BlockModeVariantAvailable)(k-1)

#

@

TeethCounter，ATP根据[IdenticalLockedOdometer](#IdenticalLockedOdometer)中锁存的最后一个中断的[CogCounter](#CogCounter)变化值，更新TeethCounter，作为主任务使用的里程计齿数值。TeethCounter的计算应考虑里程计安装方向和CogCounter的寄存器取值范围。[TeethCounter](#TeethCounter)是有符号值。如果[TeethCounter](#TeethCounter)大于0，则表示里程计相对于初始位置向列车**END\_1**方向转动；反之如果小于0，则表示里程计相对于初始位置向列车**END\_2**方向转动。

$

[TeethCounter](#TeethCounter) used as the odometer cog value in one deferred task, which is the difference of the [CogCounter](#CogCounter) in the last interrupt of adjacent cycle. The calculation of the [TeethCounter](#TeethCounter) shall consider the installation direction of the odometer and the register range of the [CogCounter](#CogCounter). [TeethCounter](#TeethCounter) is a signed value. If [TeethCounter](#TeethCounter) greater than 0, then means the odometer rotating toward to the train **END\_1** direction; other hand, if it less than 0, then means the odometer rotating toward to the **END\_2**.

%

[TeethCounter](#TeethCounter)(k)

= [TeethCounter](#TeethCounter)(k-1)

+ ([IdenticalLockedOdometer](#IdenticalLockedOdometer)[ATP\_INTERRUPT\_NB - 1].[CogCounter](#CogCounter)(k)

- [IdenticalLockedOdometer](#IdenticalLockedOdometer)[ATP\_INTERRUPT\_NB - 1].[CogCounter](#CogCounter)(k-1))

\* [ATPsetting](#ATPsetting).CCcoreOdoCogIncreasing[[CoreId](#CoreId)]

#

@

CogPositionBeforeTopLoc，CogPositionAfterTopLoc，如果本周期读到信标，则通过[IdenticalLockedOdometer](#IdenticalLockedOdometer)计算读到信标瞬间的里程计齿数信息：

使用Top-loc发生的前一个中断的[CogCounter](#CogCounter)来更新。CogPositionBeforeTopLoc；使用Top-loc发生时中断的[CogCounter](#CogCounter)来更新CogPositionAfterTopLoc；其他情况，CogPositionBeforeTopLoc和CogPositionAfterTopLoc保持不变。其中i表示锁存收到Top-loc信号的那个中断。如果上下CPU收到Top-loc相差1个中断，则使用较早的的中断作为计算CogPositionBeforeTopLoc的依据，而较迟的那个中断作为计算CogPositionAfterTopLoc的依据。

$

If a beacon with top-loc received in this cycle, ATP shall record the cog position of the interrupt when and just before the top-loc happen:

[CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc), the [CogCounter](#CogCounter) in the interrupt just before the top-loc happen; [CogPositionAfterTopLoc](#CogPositionAfterTopLoc), the [CogCounter](#CogCounter) in the interrupt when the top-loc happen. Which, i means the interrupt received top-loc signal.

%

[CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k)

= [TeethCounter](#TeethCounter)(k-1)

+ (([IdenticalLockedOdometer](#IdenticalLockedOdometer)[i-1].[CogCounter](#CogCounter)(k)

- [IdenticalLockedOdometer](#IdenticalLockedOdometer)[**ATP\_INTERRUPT\_NB**-1].[CogCounter](#CogCounter)(k-1))

\* [ATPsetting](#ATPsetting).CCcoreOdoCogIncreasing[[CoreId](#CoreId)]))

[CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k)

= [TeethCounter](#TeethCounter)(k-1)

+ (([IdenticalLockedOdometer](#IdenticalLockedOdometer)[i].[CogCounter](#CogCounter)(k)

- [IdenticalLockedOdometer](#IdenticalLockedOdometer)[**ATP\_INTERRUPT\_NB**-1].[CogCounter](#CogCounter)(k-1))

\* [ATPsetting](#ATPsetting).CCcoreOdoCogIncreasing[[CoreId](#CoreId)]))

#

@

WheelStopped，如果当前在进行传感器测试，且任一中断中未发生三路全通或全堵错误，且一个周期所有中断内三路传感器的导通状态都与上周期的结果相同时，输出WheelStopped为**True**。否则为**False**。

$

Wheel shall consider safely stopped [WheelStopped](#WheelStopped) at cycle k if the following conditions are fulfilled: sensors test has been performed, and at least one sensor out of three sensors C1, C2, C3 has detected expected sequence,

and at least one sensor out of three sensors C1, C2, C3 has not detected expected sequence, and sensors test result combination on three sensors C1, C2, C3 has not changed between cycle k-1 and k.

%

if ([SensorTestPerformed](#SensorTestPerformed)(k) == **True**)

[WheelStopped](#WheelStopped)(k)

= (([UnconsistentSensorTest](#UnconsistentSensorTest)(k) == **False**)

and ([SensorSequenceDetected\_1](#SensorSequenceDetected_1) = [SensorSequenceDetected\_1](#SensorSequenceDetected_1)(k-1))

and ([SensorSequenceDetected\_2](#SensorSequenceDetected_2) = [SensorSequenceDetected\_2](#SensorSequenceDetected_2)(k-1))

and ([SensorSequenceDetected\_3](#SensorSequenceDetected_3) = [SensorSequenceDetected\_3](#SensorSequenceDetected_3)(k-1)))

else:

[WheelStopped](#WheelStopped) = **False**

#

@

WheelFilteredStopped，判断本周期车轮是否处于滤过停止状态，规则如下：

如果WheelFilteredStopped上周期为**False**，而本周期[WheelStopped](#WheelStopped)由**False**变为**True**，则认为本周期为**True**。在此条件下，记录停车时的齿数LastStopCogPosition为当前齿数。WheelFilteredStopped由**True**变为**False**的条件：齿数移动超过1个齿。

$

At cycle k, [WheelFilteredStopped](#WheelFilteredStopped) shall change from **False** to **True** on raising edge of [WheelStopped](#WheelStopped) information, That is, if: [WheelStopped](#WheelStopped) information was **False** at cycle k-1,and [WheelStopped](#WheelStopped) information was **True** at cycle k. and then: LastStopCogPosition is assigned to [TeethCounter](#TeethCounter)， At cycle k, [WheelFilteredStopped](#WheelFilteredStopped) shall change from **True** to **False**, according following expression: the cog moved more than one cog;

%

def [WheelFilteredStopped](#WheelFilteredStopped)(k):

if (not [WheelFilteredStopped](#WheelFilteredStopped)(k-1)

and not [WheelStopped](#WheelStopped)(k-1)

and [WheelStopped](#WheelStopped)(k)):

LastStopCogPosition = [TeethCounter](#TeethCounter)(k)

return **True**

elif ([WheelFilteredStopped](#WheelFilteredStopped)(k-1)

and not [UnconsistentSensorTest](#UnconsistentSensorTest)(k)

and [abs](#abs)([TeethCounter](#TeethCounter)(k) - LastStopCogPosition) <= 1):

return **True**

else:

return **False**

#

@

MaxCountCogsRunInCycleExceeded，里程计转过齿数不能超过周期最大值，也不能超过的相邻中断的最大值。

$

ATP shall detect whether the cog number counted in adjacent interrupt is greater than the default maximum cog number on cycle or on interrupt.

%

def [MaxCountCogsRunInCycleExceeded](#MaxCountCogsRunInCycleExceeded)(k):

if (abs([IdenticalLockedOdometer](#IdenticalLockedOdometer)[**ATP\_INTERRUPT\_NB**-1].CogCounter(k)

- [IdenticalLockedOdometer](#IdenticalLockedOdometer)[**ATP\_INTERRUPT\_NB**-1].CogCounter(k-1))

> ATPsetting.OdoMaxCogOnCycle):

return **True**

else:

for i in [range](#range)(**ATP\_INTERRUPT\_NB-1**):

if ([abs](#abs)([IdenticalLockedOdometer](#IdenticalLockedOdometer)[i].[CogCounter](#CogCounter)

- [IdenticalLockedOdometer](#IdenticalLockedOdometer)[i+1].[CogCounter](#CogCounter))> [ATPsetting](#ATPsetting).OdoMaxCogOnIntrrupt):

return **True**

else:

continue

return **False**

#

@

WheelKinematicsInvalidForCogCount，如果ATP检测到某个中断的齿数转过最大值时，设置齿数计算错误。

$

If the calculated movement exceeds the default one, ATP shall set the wheel kinematics invalid.

%

[WheelKinematicsInvalidForCogCount](#WheelKinematicsInvalidForCogCount) = [MaxCountCogsRunInCycleExceeded](#MaxCountCogsRunInCycleExceeded)(k))

#

$

If [OdometerState](#OdometerState) is **NOT\_INITIALIZED** at cycle k, and if wheel detected stopped at cycle k, then [WheelMinimumMovement](#WheelMinimumMovement) and [WheelMaximumMovement](#WheelMaximumMovement) shall be set to zero.

%

if ([OdometerState](#OdometerState)(k) == **NOT\_INITIALIZED**)

if ([WheelFilteredStopped](#WheelFilteredStopped)(k) == **True**))

[WheelMinimumMovement](#WheelMinimumMovement) = 0

[WheelMaximumMovement](#WheelMaximumMovement) = 0

else:

[WheelMinimumMovement](#WheelMinimumMovement) = —[ATPsetting](#ATPsetting).MaxMotionPerCycle

[WheelMaximumMovement](#WheelMaximumMovement) = [ATPsetting](#ATPsetting).MaxMotionPerCycle

#

@

里程计状态由**NOT\_INITIALIZED**变为**WAITING\_COG\_POSITION\_CODE\_READY**的条件是:上周期在**NOT\_INITIALIZED**；上周期在[WheelFilteredStopped](#WheelFilteredStopped)；本周期未[WheelFilteredStopped](#WheelFilteredStopped)而且未检测到传感器测试失败。

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **NOT\_INITIALIZED** to **WAITING\_COG\_POSITION\_CODE\_READY** if: a falling edge is detected on [[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped) information, and sensors test is consistent at cycle k and was consistent at cycle k-1。

%

if (([OdometerState](#OdometerState)(k-1) = **NOT\_INITIALIZED**)

and (not [[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)(k) and not [UnconsistentSensorTest](#UnconsistentSensorTest)(k))

and ([[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)(k-1))

[OdometerState](#OdometerState) = **WAITING\_COG\_POSITION\_CODE\_READY**

#

@

若检测到传感器三路全通或全堵，则进入**INVALID**传感器无效。

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **NOT\_INITIALIZED** to **INVALID，**if sensors test is not consistent at cycle k.

%

if (([OdometerState](#OdometerState)(k-1) = **NOT\_INITIALIZED**)

**and** [UnconsistentSensorTest](#UnconsistentSensorTest)(k))

[OdometerState](#OdometerState) = **INVALID**

#

@

InitializationTimer，在**WAITING\_COG\_POSITION\_CODE\_READY**状态下累加初始化时间.

$

ATP shall accumulate the time for waiting cog position ready state.

%

if ([OdometerState](#OdometerState)(k-1) == **WAITING\_COG\_POSITION\_CODE\_READY**

and [OdometerState](#OdometerState)(k) == **WAITING\_COG\_POSITION\_CODE\_READY**)

[InitializationTimer](#InitializationTimer) = [InitializationTimer](#InitializationTimer)(k-1) + 1

elif ([OdometerState](#OdometerState)(k-1) != **WAITING\_COG\_POSITION\_CODE\_READY**

and [OdometerState](#OdometerState)(k) == **WAITING\_COG\_POSITION\_CODE\_READY**)

[InitializationTimer](#InitializationTimer) = 1

else:

[InitializationTimer](#InitializationTimer) = 0

#

@

由**WAITING\_COG\_POSITION\_CODE\_READY**转回**NOT\_INITIALIZED**状态的条件：

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **WAITING\_COG\_POSITION\_CODE\_READY** to **NOT\_INITIALIZED** if:

wheel is detected stopped ([[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)),and cog position remains

unknown (not [OdometerCogPositionReady](#OdometerCogPositionReady)), and there is no sensors test

inconsistency, and time elapsed since last time [OdometerState](#OdometerState) was **NOT\_INITIALIZED** ([InitializationTimer](#InitializationTimer)) is strictly less than [ATPsetting](#ATPsetting).OdoInitTimeout

%

if ([OdometerState](#OdometerState)(k-1) = **WAITING\_COG\_POSITION\_CODE\_READY**)

and [[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)(k)

and not [OdometerCogPositionReady](#OdometerCogPositionReady)(k)

and not [UnconsistentSensorTest](#UnconsistentSensorTest)(k)

and ([InitializationTimer](#InitializationTimer)(k) < [ATPsetting](#ATPsetting).OdoInitTimeout)

[OdometerState](#OdometerState) = **NOT\_INITIALIZED**

#

@

由**WAITING\_COG\_POSITION\_CODE\_READY**转入**INITIALIZED**状态的条件：

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **WAITING\_COG\_POSITION\_CODE\_READY** to **INITIALIZED**

If: Cog position is safely known which means that wheel angular position is well-known;and there is no sensors test inconsistency; and time elapsed since last time [OdometerState](#OdometerState) was **NOT\_INITIALIZED** ([InitializationTimer](#InitializationTimer)) is strictly less than [ATPsetting](#ATPsetting).OdoInitTimeout.

%

if ([OdometerState](#OdometerState)(k-1) == **WAITING\_COG\_POSITION\_CODE\_READY**

and [OdometerCogPositionReady](#OdometerCogPositionReady)(k)

and not [UnconsistentSensorTest](#UnconsistentSensorTest)(k)

and ([InitializationTimer](#InitializationTimer)(k)< [ATPsetting](#ATPsetting).OdoInitTimeout))

[OdometerState](#OdometerState) = **INITIALIZED**

#

@

由**WAITING\_COG\_POSITION\_CODE\_READY**转入**INVALID**的条件：

￥

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **WAITING\_COG\_POSITION\_CODE\_READY** to **INVALID** if:sensors test inconsistency is detected, or time elapsed since last time [OdometerState](#OdometerState) was **NOT\_INITIALIZED** ([InitializationTimer](#InitializationTimer)) is more than or equal to the [ATPsetting](#ATPsetting).OdoInitTimeout。

%

if ([OdometerState](#OdometerState)(k-1) == **WAITING\_COG\_POSITION\_CODE\_READY**)

and (([InitializationTimer](#InitializationTimer)(k)>= [ATPsetting](#ATPsetting).OdoInitTimeout)

or [UnconsistentSensorTest](#UnconsistentSensorTest)(k))

[OdometerState](#OdometerState) = **INVALID**

#

@

在里程计初始化阶段，ATP需根据当前车头激活方向和上周期位移的结果，对本周期位移进行过估处理。

$

When odometer is initializing, wheel movement shall be over and under estimated considering maximum acceleration per cycle according with the train front:

%

if ([OdometerState](#OdometerState)(k) == **WAITING\_COG\_POSITION\_CODE\_READY**)

if (([TrainFrontEnd](#TrainFrontEnd)(k-1) == **END\_2**) or ([NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k-1) == **True**))

[WheelMinimumMovement](#WheelMinimumMovement)(k) = [WheelMinimumMovement](#WheelMinimumMovement)(k-1) + [ATPsetting](#ATPsetting).MaxMotionPerCycle

[WheelMaximumMovement](#WheelMaximumMovement)(k) = [WheelMaximumMovement](#WheelMaximumMovement)(k-1) - [ATPsetting](#ATPsetting).MaxMotionPerCycle

else:

[WheelMinimumMovement](#WheelMinimumMovement)(k) = [WheelMinimumMovement](#WheelMinimumMovement)(k-1) — [ATPsetting](#ATPsetting).MaxMotionPerCycle

[WheelMaximumMovement](#WheelMaximumMovement)(k) = [WheelMaximumMovement](#WheelMaximumMovement)(k-1) + [ATPsetting](#ATPsetting).MaxMotionPerCycle

#

@

当上周期里程计已在**INITIALIZED**状态，并满足以下条件之一时，里程计状态由**INITIALIZED**变为**INVALID**：传感器测试检测出三路全通全堵；或者，非停车状态，而且齿数齿号也不一致。

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **INITIALIZED** to **INVALID** if:[OdometerState](#OdometerState) was evaluated Initialized at cycle k-1,And:Sensors test result is inconsistent;Or neither wheel filtered stopped nor cog position ready.

%

if ([OdometerState](#OdometerState)(k-1) is **INITIALIZED**

and (([UnconsistentSensorTest](#UnconsistentSensorTest)(k) == **True**)

or (not [WheelFilteredStopped](#WheelFilteredStopped)(k)

and not [OdometerCogPositionReady](#OdometerCogPositionReady)(k)))):

[OdometerState](#OdometerState) = **INVALID**

#

@

在**INITIALIZED**状态，如果齿数齿号匹配，则计算车轮最大最小位移依据伪代码中的公式：对于车载ATP软件的位移，在齿数齿号匹配的状态下，无论ATP位于**END\_1**还是**END\_2**，也无论激活哪段车头，始终以**END\_1**方向为位移的正方向。即当位移大于0时，表示列车向**END\_1**端方向运行，反之则向**END\_2**端方向运行。

$

If motion and speed are available at cycle k, then wheel curvilinear movement calculates as follows: When odometer cog-counter-code matched, regardless of ATP in **END\_1** or **END\_2**, and no matter the activation of train front, the direction towards **END\_1** is always be set as the positive direction. That is, when the movement is greater than 0, indicating the direction of the train is running to **END\_1**, and vice versa to **END\_2**.

%

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

[WheelMinimumMovement](#WheelMinimumMovement)(k) = [MinCogCalibration](#MinCogCalibration)(k-1) \* ([TeethCounter](#TeethCounter)(k) — [TeethCounter](#TeethCounter)(k-1))

[WheelMaximumMovement](#WheelMaximumMovement)(k) = [MaxCogCalibration](#MaxCogCalibration)(k-1) \* ([TeethCounter](#TeethCounter)(k) — [TeethCounter](#TeethCounter)(k-1))

#

@

在无效状态停车，并未检测到传感器错误，则能回到非初始化状态。

$

At cycle k, ATP shall consider that [OdometerState](#OdometerState) changes from **INVALID** to **NOT\_INITIALIZED** if:[OdometerState](#OdometerState) was evaluated Invalid at cycle k-1,and wheel is detected stopped ([[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)),and there is no sensors test inconsistency.

%

if ([OdometerState](#OdometerState)(k-1) == **INVALID**

and [[WheelFilteredStopped](#WheelFilteredStopped)](#WheelStopped)(k)

and not [UnconsistentSensorTest](#UnconsistentSensorTest)(k))

[OdometerState](#OdometerState) = **NOT\_INITIALIZED**

#

@

在里程计无效状态下，ATP直接使用测得值计算车轮位移（因为此时列车运动学失效，后续功能并不使用测得的列车车轮位移）。

$

In invalid status, ATP shall calculate wheel movement by using measured value of the odometer.

%

if ([OdometerState](#OdometerState)(k) == **INVALID**)

[WheelMinimumMovement](#WheelMinimumMovement)(k) = [MinCogCalibration](#MinCogCalibration)(k-1) \* ([TeethCounter](#TeethCounter)(k) — [TeethCounter](#TeethCounter)(k-1))

[WheelMaximumMovement](#WheelMaximumMovement)(k) = [MaxCogCalibration](#MaxCogCalibration)(k-1) \* ([TeethCounter](#TeethCounter)(k) — [TeethCounter](#TeethCounter)(k-1))

#

@

SensorTestContradiction，当里程计读数为0，但中断中却未进行传感器测试时，设置该变量为**True**，否则为**False**。

NoCommunicationWithOdometer，当SensorTestContradiction保持为**True**超过限定时间后，设置该值为真，表明中断中的传感器测试判断失败。

$

ATP shall invalidate wheel kinematic if minimum odometer motion is null and sensors test is not performed for more than [ATPsetting](#ATPsetting).OdoTestContradictionDuration.

%

def [SensorTestContradiction](#SensorTestContradiction)(k):

return (not [WheelFilteredStopped](#WheelFilteredStopped)(k)

and [TeethCounter](#TeethCounter)(k-1) == [TeethCounter](#TeethCounter)(k-2)

and not [SensorTestPerformed](#SensorTestPerformed)(k))

def [NoCommunicationWithOdometer](#NoCommunicationWithOdometer)(k):

if (Initialization

or not [SensorTestContradiction](#SensorTestContradiction)(k)):

SensorTestContradictionDuration = 0

return **False**

else:

SensorTestContradictionDuration = SensorTestContradictionDuration(k-1) + 1

if (SensorTestContradictionDuration > [ATPsetting](#ATPsetting).OdoTestContradictionDuration):

return **True**

else:

return False

#

@

ValidWheelKinematic，车轮运动学特性有效.

$

Wheel kinematic is valid if odometer is valid, the calculated motion is not greater than the default value, and there is communication with odometer.

%

[ValidWheelKinematic](#ValidWheelKinematic)(k)

= (([OdometerState](#OdometerState)(k) != **INVALID**)

and (not [WheelKinematicsInvalidForCogCount](#WheelKinematicsInvalidForCogCount)(k))

and (not [NoCommunicationWithOdometer](#NoCommunicationWithOdometer)(k)))

#

@

WheelMinSpeed，里程计测得车轮最小速度，非负值。

%

def [WheelMinSpeed](#WheelMinSpeed)(k):

return [round.floor](#roundfloor)([abs](#abs)([WheelMinimumMovement](#WheelMinimumMovement)(k)) / ATP\_CYCLE\_TIME)

#

@

WheelMaxSpeed，ATP根据里程计测得位移计算车轮最大速度，该值为非负数，并且向上取整。

$

ATP calculates the maximum wheel speed according to the maximum wheel movement; this value is non-negative and rounded up.

%

def [WheelMaxSpeed](#WheelMaxSpeed)(k):

return [round.ceil](#roundceil)([abs](#abs)([WheelMaximumMovement](#WheelMaximumMovement)(k)) / **ATP\_CYCLE\_TIME**)

#

@

InstantaneousWheelAcceleration，在进行最大位移过估算法之前，需计算瞬时车轮加速度（为减少采样周期过短使得采样误差导致的加速度大幅变化，ATP使用相邻2个周期的算术平均加速度作为瞬时加速度）。

$

When wheel motion and acceleration are measurable, then instantaneous acceleration computed according following expression:

%

def [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k):

return (([abs](#abs)([TeethCounter](#TeethCounter)(k) - [TeethCounter](#TeethCounter)(k-2))

- [abs](#abs)([TeethCounter](#TeethCounter)(k-2) - [TeethCounter](#TeethCounter)(k-4)))

\* [MaxCogCalibration](#MaxCogCalibration)(k-1) / [pow](#pow)(2\*ATP\_CYCLE\_TIME))

#

@

FilteredWheelAcceleration，在进行最大位移过估算法之前，ATP需计算**FILTERED\_ACCELERATION\_NB**个周期的滤波平均加速度

$

[FilteredWheelAcceleration](#FilteredWheelAcceleration) measurement is the average of [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration) over **FILTERED\_ACCELERATION\_NB** cycles for filtering the fluctuation causing by the sampling period.

%

[FilteredWheelAcceleration](#FilteredWheelAcceleration)(k)

= ([InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k)

+ [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k-1)

+ ...

+ [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k-**FILTERED\_ACCELERATION\_NB**+1))

/ **FILTERED\_ACCELERATION\_NB**

#

@

AverageWheelAcceleration，在进行最大位移过估算法之前，ATP需计算**AVERAGE\_ACCELERATION\_NB**个周期的平均车轮加速度

$

When wheel motion and acceleration are measurable, sliding average acceleration at cycle k defined by following expression:

%

[AverageWheelAcceleration](#AverageWheelAcceleration)(k)

= ([InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k)

+ [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k-1)

+ ...

+ [InstantaneousWheelAcceleration](#InstantaneousWheelAcceleration)(k-**AVERAGE\_ACCELERATION\_NB**+1))

/ **AVERAGE\_ACCELERATION\_NB**

#

@

StartBrakingMovementMax，记录由**COASTING**→**BRAKING**，**COASTING**→**SLIDING**，或**BRAKING**→**SLIDING**状态时的最大位移。

$

ATP records the maximum movement when the state transferring from **COASTING** to **BRAKING** or **SLIDING**, or from **BRAKING** to **SLIDING**.

%

def [StartBrakingMovementMax](#StartBrakingMovementMax)(k):

if (Initialization

or ([MotionOverEstimationState](#MotionOverEstimationState)(k) is **COASTING**)):

return 0

elif (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **COASTING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **BRAKING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **COASTING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **SLIDING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **BRAKING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **SLIDING**))):

return [MaximumTrainMotion](#MaximumTrainMotion)(k-1)

else:

return [StartBrakingMovementMax](#StartBrakingMovementMax)(k-1)

#

@

MaxMotionDuringBrakingOrSliding，在制动或者打滑状态下反向运行的最大位移.

$

ATP shall record the reversed motions during the braking or sliding state.

%

if ([StartBrakingMovementMax](#StartBrakingMovementMax)(k-1) > 0)

[MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding)(k)

= [min](#min)(([MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding)(k-1) + [WheelMaximumMovement](#WheelMaximumMovement)(k)), 0)

elif ([StartBrakingMovementMax](#StartBrakingMovementMax)(k-1) < 0)

[MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding)(k)

= [max](#max)(([MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding)(k-1) + [WheelMaximumMovement](#WheelMaximumMovement)(k)), 0)

else:

[MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding) = 0

#

@

MaxMotionOdometerSignChanged，用于监控是否发生了测得车轮位移反向.

$

If the reversed motion during braking or sliding state is greater than a project defined distance, ATP shall consider the motion sign changed.

%

def [MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k):

return ([sign](#sign)([StartBrakingMovementMax](#StartBrakingMovementMax)(k-1)) != [sign](#sign)([WheelMaximumMovement](#WheelMaximumMovement)(k))

and ([abs](#abs)([MaxMotionDuringBrakingOrSliding](#MaxMotionDuringBrakingOrSliding)(k))

> [ATPsetting](#ATPsetting).OdoMinDistAfterSenseChange))

#

@

StartSlidingSpeed，记录由**COASTING**或**BRAKING**进入**SLIDING**状态时的速度。

$

ATP shall record the speed when the train begins to slide.

%

if (Initialization

or ([MotionOverEstimationState](#MotionOverEstimationState)(k) == **COASTING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k) == **BRAKING**))

[StartSlidingSpeed](#StartSlidingSpeed) = 0

elif ((([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **COASTING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **BRAKING**))

and ([MotionOverEstimationState](#MotionOverEstimationState)(k) == **SLIDING**))

[StartSlidingSpeed](#StartSlidingSpeed) = [WheelMaxSpeed](#WheelMaxSpeed)(k-1)

else:

[StartSlidingSpeed](#StartSlidingSpeed) = [StartSlidingSpeed](#StartSlidingSpeed)(k-1)

#

@

TimeInSliding，记录在**SLINDING**状态下持续了多少个周期.

$

ATP shall record how many cycles staying in **SLIDING** state.

%

if (Initialization

or (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **SLIDING**)

and ([MotionOverEstimationState](#MotionOverEstimationState)(k) != **SLIDING**)))

[TimeInSliding](#TimeInSliding) = 0

elif ([MotionOverEstimationState](#MotionOverEstimationState)(k) == **SLIDING**)

[TimeInSliding](#TimeInSliding) = [TimeInSliding](#TimeInSliding)(k-1) + 1

else:

[TimeInSliding](#TimeInSliding) = [TimeInSliding](#TimeInSliding)(k-1)

#

@

当满足以下条件时，[MotionOverEstimationState](#MotionOverEstimationState)由**COASTING**转入**BRAKING**，并执行：

$

The state transfers from “**COASTING**” to “**BRAKING**” when:

%

if (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **COASTING**)

and ([WheelFilteredStopped](#WheelFilteredStopped)(k) != **True**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k)< [ATPsetting](#ATPsetting).BrakingStartAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k)>= [ATPsetting](#ATPsetting).SlidingStartAcc)

and [OdometerState](#OdometerState)(k) is **INITIALIZED**)

[MotionOverEstimationState](#MotionOverEstimationState) = **BRAKING**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from “**COASTING**” to “**SLIDING**” when:

%

if (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **COASTING**)

and ([WheelFilteredStopped](#WheelFilteredStopped)(k) != **True**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < [ATPsetting](#ATPsetting).SlidingStartAcc)

and [OdometerState](#OdometerState)(k) is **INITIALIZED**)

[MotionOverEstimationState](#MotionOverEstimationState) = **SLIDING**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from “**BRAKING**” to “**SLIDING**” when:

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **BRAKING**

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < [ATPsetting](#ATPsetting).SlidingStartAcc)

and ([AverageWheelAcceleration](#AverageWheelAcceleration)(k) < [ATPsetting](#ATPsetting).BrakingStartAcc)

and ([OdometerState](#OdometerState)(k) is **INITIALIZED**)

and (not [MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k)))

[MotionOverEstimationState](#MotionOverEstimationState) = **SLIDING**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from “**BRAKING**” to “**COASTING**” when:

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **BRAKING**

and (([AverageWheelAcceleration](#AverageWheelAcceleration)(k)>= [ATPsetting](#ATPsetting).BrakingStartAcc)

or ([OdometerState](#OdometerState)(k) is **INVALID**)

or ([MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k))))

[MotionOverEstimationState](#MotionOverEstimationState) = **COASTING**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from “**SLIDING**” to “**COASTING**” when:

%

if (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **SLIDING**)

and (([OdometerState](#OdometerState)(k) is **INVALID**)

or ([MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k) == **True**))

[MotionOverEstimationState](#MotionOverEstimationState) = **COASTING**

#

@

SlidingEnded，判断是否结束打滑状态的条件之一。

$

At cycle k, if motion overestimation status is **SLIDING**, ATP shall consider that sliding effect is ended ([SlidingEnded](#SlidingEnded)) if [FilteredWheelAcceleration](#FilteredWheelAcceleration) is strictly less than [ATPsetting](#ATPsetting).SlippingStopAcc and strictly greater than [ATPsetting](#ATPsetting).SlidingStopAcc for more than [ATPsetting](#ATPsetting).SlidingGripRecoveryTime.

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **SLIDING**

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > [ATPsetting](#ATPsetting).SlidingStopAcc)

and ([MotionOverEstimationState](#MotionOverEstimationState)(k-2) == **SLIDING**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-1) < [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-1) > [ATPsetting](#ATPsetting).SlidingStopAcc)

...

and ([MotionOverEstimationState](#MotionOverEstimationState)(k-[ATPsetting](#ATPsetting).SlidingGripRecoveryTime) == **SLIDING**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-[ATPsetting](#ATPsetting).SlidingGripRecoveryTime+1)

< [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-[ATPsetting](#ATPsetting).SlidingGripRecoveryTime+1)

> [ATPsetting](#ATPsetting).SlidingStopAcc))

[SlidingEnded](#SlidingEnded) = **True**

else:

[SlidingEnded](#SlidingEnded) = **False**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from “**SLIDING**” to “**BRAKING**” when:

%

if (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **SLIDING**)

and ([OdometerState](#OdometerState)(k) is **INITIALIZED**)

and (not [MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k))

and ([TimeInSliding](#TimeInSliding)(k-1) <= [ATPsetting](#ATPsetting).SlidingTimeout)

and (0 < (|[StartSlidingSpeed](#StartSlidingSpeed)(k-1)| + [TimeInSliding](#TimeInSliding)(k-1)\* [ATPsetting](#ATPsetting).SlidingStopAcc))

and ((|[StartSlidingSpeed](#StartSlidingSpeed)(k-1)| + [TimeInSliding](#TimeInSliding)(k-1)\* [ATPsetting](#ATPsetting).SlidingStopAcc)

< |[WheelMaxSpeed](#WheelMaxSpeed)(k)|)

and ([SlidingEnded](#SlidingEnded)(k) == **True**))

[MotionOverEstimationState](#MotionOverEstimationState) = **BRAKING**

#

@

SlidingExcess，测得的加速度在项目配置范围内满足一定时间，是ATP判断过度打滑的必要条件之一。

$

At cycle k, if motion overestimation status is **SLIDING**, ATP shall consider that sliding is excess ([SlidingExcess](#SlidingExcess)) if [FilteredWheelAcceleration](#FilteredWheelAcceleration) is strictly less than [ATPsetting](#ATPsetting).SlippingStopAcc and strictly greater than [ATPsetting](#ATPsetting).SlidingStopAcc for more than [ATPsetting](#ATPsetting).SlidingExcessTime.

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) == **SLIDING**

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > [ATPsetting](#ATPsetting).SlidingStopAcc)

and ([MotionOverEstimationState](#MotionOverEstimationState)(k-2) == **SLIDING**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-1) < [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-1) > [ATPsetting](#ATPsetting).SlidingStopAcc)

...

and ([MotionOverEstimationState](#MotionOverEstimationState)(k-[ATPsetting](#ATPsetting).SlidingExcessTime) == **SLIDING**)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-[ATPsetting](#ATPsetting).SlidingExcessTime+1)

< [ATPsetting](#ATPsetting).SlippingStopAcc)

and ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k-[ATPsetting](#ATPsetting).SlidingExcessTime+1)

> [ATPsetting](#ATPsetting).SlidingStopAcc))

[SlidingExcess](#SlidingExcess) = **True**

else:

[SlidingExcess](#SlidingExcess) = **False**

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from **SLIDING** to **SKIDDING** when:

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **SLIDING**

and [OdometerState](#OdometerState)(k) is **INITIALIZED**

and not [MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k)

and ([TimeInSliding](#TimeInSliding)(k-1) > [ATPsetting](#ATPsetting).SlidingTimeout

or ([StartSlidingSpeed](#StartSlidingSpeed)(k-1)+ [TimeInSliding](#TimeInSliding)(k-1) \* [ATPsetting](#ATPsetting).SlidingStopAcc) <= **0**

or ((([StartSlidingSpeed](#StartSlidingSpeed)(k-1) + [TimeInSliding](#TimeInSliding)(k-1)\* [ATPsetting](#ATPsetting).SlidingStopAcc)

>= [WheelMaxSpeed](#WheelMaxSpeed)(k))

and [SlidingExcess](#SlidingExcess)(k))))

[MotionOverEstimationState](#MotionOverEstimationState) = **SKIDDING**

#

@

由于当前项目应用的车辆都装有ABS防抱死系统，使得在制动时列车的加速度不会连续若干周期小于[ATPsetting](#ATPsetting).SlidingStopAcc。因此，如果某周期瞬间加速度小于[ATPsetting](#ATPsetting).SlidingStopAcc；并且在之后的测得加速度满足[SlidingExcess](#SlidingExcess)条件，那么，下列两个条件可以同时成立：

$

Because the application of anti-lock braking system for the train of current project, makes the brake acceleration cannot continuous less than [ATPsetting](#ATPsetting).SlidingStopAcc for serious cycles. Therefore, if there was an unexpected instantaneous acceleration less than [ATPsetting](#ATPsetting).SlidingStopAcc, and the after cycles' acceleration met the criteria of [SlidingExcess](#SlidingExcess), then the following two conditions can hold simultaneously.

%

|[StartSlidingSpeed](#StartSlidingSpeed)(k-1)| + [TimeInSliding](#TimeInSliding)(k) \* [ATPsetting](#ATPsetting).SlidingStopAcc >= |[WheelMaxSpeed](#WheelMaxSpeed)(k)|)

AND ([SlidingExcess](#SlidingExcess)(k) == **True**)

#

$

The [MotionOverEstimationState](#MotionOverEstimationState) transfers from **SKIDDING** to **COASTING** when:

%

if ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **SKIDDING**

and ([WheelFilteredStopped](#WheelFilteredStopped)(k)

or [OdometerState](#OdometerState)(k) is **INVALID**))

[MotionOverEstimationState](#MotionOverEstimationState) = **COASTING**

#

@

StartBrakingMovementMin，记录由**COASTING**进入**BRAKING**，**COASTING**进入**SLIDING**，或者**BRAKING**进入**SLIDING**状态时的最小位移。

$

ATP records the minimum movement when the state transferring from **COASTING** to **BRAKING** or **SLIDING**, or from **BRAKING** to **SLIDING**.

%

def [StartBrakingMovementMin](#StartBrakingMovementMin)(k):

if (Initialization

or [OdometerState](#OdometerState)(k-1) is not **INITIALIZED**

or ([MotionOverEstimationState](#MotionOverEstimationState)(k) is **COASTING**)):

return 0

elif (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **COASTING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **BRAKING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **COASTING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **SLIDING**)

or ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **BRAKING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) is **SLIDING**)):

return [MinimumTrainMotion](#MinimumTrainMotion)(k-1)

else:

return [StartBrakingMovementMin](#StartBrakingMovementMin)(k-1)

#

@

OverestimatedMotionMin，根据打滑状态机，对里程计测得的最小位移进行补偿。在**BRAKING**或**SLIDING**状态时，由于车辆ABS的作用，会在瞬间释放制动而使得转速突然增大，可能接近但不会大于进入制动状态时刻的速度。而由于获取里程计读值有1个齿的采样误差，在该误差的作用下，可能会使得测得位移大于进入制动状态时刻的位移，即出现测得车轮最小位移大于列车最大位移的情形。为防止这种情况，需要对列车最小位移进行调整，即始终使用进入制动状态时刻与测得车轮最小位移中绝对值较小的一个。其他情况，无需补偿，使用测得位移。

%

def [OverstimatedMotionMin](#OverstimatedMotionMin)(k):

if (sign([StartBrakingMovementMin](#StartBrakingMovementMin)(k)) == sign([WheelMinimumMovement](#WheelMinimumMovement)(k))

and ([MotionOverEstimationState](#MotionOverEstimationState) (k) is **BRAKING**

or [MotionOverEstimationState](#MotionOverEstimationState) (k) is **SLIDING**)):

if ([StartBrakingMovementMin](#StartBrakingMovementMin)(k) >= 0):

return min([StartBrakingMovementMin](#StartBrakingMovementMin)(k), [WheelMinimumMovement](#WheelMinimumMovement)(k))

else:

return (-1 \* min(abs([StartBrakingMovementMin](#StartBrakingMovementMin)(k)), abs([WheelMinimumMovement](#WheelMinimumMovement)(k))))

else:

return [WheelMinimumMovement](#WheelMinimumMovement)(k)

#

@

OverestimatedMotionMax，根据打滑状态机，对里程计测得的最大位移进行补偿。如果本周期在**BRAKING**状态，按如下规则更新列车最大位移：如果本周期测得车轮位移与[StartBrakingMovementMax](#StartBrakingMovementMax)方向相同，且前者的绝对值大于后者的绝对值，表明由于采样齿数波动，测得位移大于[StartBrakingMovementMax](#StartBrakingMovementMax)，此时使用测得位移作为最大列车位移；否则，根据配置对测得位移进行补偿，取[StartBrakingMovementMax](#StartBrakingMovementMax)与补偿后的测得位移中绝对值较小的一个，位移方向与[StartBrakingMovementMax](#StartBrakingMovementMax)相同。如果本周期在**SLIDING**状态时，按如下规则更新列车最大位移：如果本周期测得车轮位移与[StartBrakingMovementMax](#StartBrakingMovementMax)方向相同，且前者的绝对值大于后者的绝对值，表明由于采样齿数波动，使得测得位移大于[StartBrakingMovementMax](#StartBrakingMovementMax)。此时使用测得位移作为最大列车位移；否则，使用[StartBrakingMovementMax](#StartBrakingMovementMax)。其他状态下，无需对测得最大位移进行补偿。

$

In **BRAKING** state, the maximum train motion overestimated as [ATPsetting](#ATPsetting).SlidingCoefficient (15% normally) at most. If the overestimated motion has greater than the start braking movement, ATP shall use the start breaking movement as the current train motion. That said the train speed during braking could not faster than before. In state **SLIDING**: If both [WheelMaximumMovement](#WheelMaximumMovement) and [StartBrakingMovementMax](#StartBrakingMovementMax) are same direction, and the absolute value of the former is greater than the absolute value of the latter, indicating that due to the sampling error makes the measured movement greater than [StartBrakingMovementMax](#StartBrakingMovementMax). In this case, ATP shall uses [WheelMaximumMovement](#WheelMaximumMovement) as current train maximum motion.

Otherwise, uses [StartBrakingMovementMax](#StartBrakingMovementMax) as train maximum motion. In other state (**COASTING**, **SKIDDING**), uses measured wheel maximum movement as current overestimated maximum train motion.

%

def [OverestimatedMotionMax](#OverestimatedMotionMax)(k):

if [MotionOverEstimationState](#MotionOverEstimationState)(k) is **BRAKING**:

if (sign([StartBrakingMovementMax](#StartBrakingMovementMax)(k)) == sign([WheelMaximumMovement](#WheelMaximumMovement)(k))

and abs([WheelMaximumMovement](#WheelMaximumMovement)(k)) > abs([StartBrakingMovementMax](#StartBrakingMovementMax)(k))):

return [WheelMaximumMovement](#WheelMaximumMovement)(k)

elif ([StartBrakingMovementMax](#StartBrakingMovementMax)(k) >= **0**):

return min(abs([StartBrakingMovementMax](#StartBrakingMovementMax)(k)),

abs([WheelMaximumMovement](#WheelMaximumMovement)(k) \* ATPsetting.SlidingCoefficient))

else:

return **-1** \* min(abs([StartBrakingMovementMax](#StartBrakingMovementMax)(k)),

abs([WheelMaximumMovement](#WheelMaximumMovement)(k) \* ATPsetting.SlidingCoefficient))

elif [MotionOverEstimationState](#MotionOverEstimationState)(k) is **SLIDING**:

if (sign([StartBrakingMovementMax](#StartBrakingMovementMax)(k)) == sign([WheelMaximumMovement](#WheelMaximumMovement)(k))

and abs([WheelMaximumMovement](#WheelMaximumMovement)(k)) > abs([StartBrakingMovementMax](#StartBrakingMovementMax)(k))):

return [WheelMaximumMovement](#WheelMaximumMovement)(k)

else:

return [StartBrakingMovementMax](#StartBrakingMovementMax)(k)

else: # Coasting and Skidding

return [WheelMaximumMovement](#WheelMaximumMovement)(k)

#

@

StartSlippingSpeed，记录由**COASTING**或**MOTORING**进入**SLIPPING**状态时的速度。

$

ATP shall record the speed when the train begins to slip.

%

def [StartSlippingSpeed](#StartSlippingSpeed)(k):

if (Initialization

or [OdometerState](#OdometerState)(k-1) is not **INITIALIZED**

or [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **COASTING**

or [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **MOTORING**):

return **0**

elif (([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **COASTING**

or [MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **MOTORING**)

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **SLIPPING**):

return [WheelMinSpeed](#WheelMinSpeed)(k-1)

else:

return [StartSlippingSpeed](#StartSlippingSpeed)(k-1)

#

@

TimeInSlipping，记录在**SLIPPING**状态下持续了多少个周期.

$

ATP shall record how many cycles staying in **SLIPPING** state.

%

def [TimeInSlipping](#TimeInSlipping)(k):

if (Initialization

or ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **SLIPPING**

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is not **SLIPPING**)):

return 0

elif ([MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **SLIPPING**):

return [TimeInSlipping](#TimeInSlipping)(k-1) + 1

else:

return [TimeInSlipping](#TimeInSlipping)(k-1)

#

@

OdometerAxleMotorized，表示需考虑里程计所安在车轴牵引导致的空转。

$

If the project that odometer installed on the traction axle of the train, ATP shall consider the slipping effect to impact the underestimation of measured wheel movement.

%

def [OdometerAxleMotorized](#OdometerAxleMotorized)(k):

return not [ATPsetting](#ATPsetting).OdoNotOnMotorizedAxle

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from “**COASTING**” to “**MOTORING**” when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **COASTING**

and [OdometerAxleMotorized](#OdometerAxleMotorized)(k)

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) <= ATPsetting.SlippingStartAcc

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > ATPsetting.TractionStartAcc

and [OdometerState](#OdometerState)(k) is **INITIALIZED**)

[MotionUnderEstimationState](#MotionUnderEstimationState) = **MOTORING**

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from “**COASTING**” to “**SLIPPING**” when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **COASTING**

and [OdometerAxleMotorized](#OdometerAxleMotorized)(k)

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > [ATPsetting](#ATPsetting).SlippingStartAcc

and [OdometerState](#OdometerState)(k) is **INITIALIZED**)

[MotionUnderEstimationState](#MotionUnderEstimationState) = **SLIPPING**

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from “**MOTORING**” to “**SLIPPING**” when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **MOTORING**

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > [ATPsetting](#ATPsetting).SlippingStartAcc

and [AverageWheelAcceleration](#AverageWheelAcceleration)(k) > [ATPsetting](#ATPsetting).MotoringStartAc)

and [OdometerState](#OdometerState)(k) is **INITIALIZED**

and [OdometerAxleMotorized](#OdometerAxleMotorized)(k))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **SLIDING**

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from **MOTORING** to **COASTING**” when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **MOTORING**

and ([AverageWheelAcceleration](#AverageWheelAcceleration)(k) <= [ATPsetting](#ATPsetting).TractionStartAcc

or [OdometerState](#OdometerState)(k) is **INVALID**

or not [OdometerAxleMotorized](#OdometerAxleMotorized)(k)))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **COASTING**

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from **SLIPPING** to **COASTING** when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **SLIPPING**

and ([OdometerState](#OdometerState)(k) is **INVALID**

or not [OdometerAxleMotorized](#OdometerAxleMotorized)(k)))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **COASTING**

#

@

SlippingEnded，判断是否结束空转状态的条件之一。

%

def [SlippingEnded](#SlippingEnded)(k):

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is not **SLIPPING**):

slipping\_ended\_counter = **0**

return **False**

elif ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < ATPsetting.SlippingStopAcc

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > ATPsetting.SlidingStopAcc):

slipping\_ended\_counter = slipping\_ended\_counter **+ 1**

return (slipping\_ended\_counter >= ATPsetting.SlippingGripRecoveryTime)

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from **SLIPPING** to **MOTORING** when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **SLIPPING**

and [OdometerState](#OdometerState)(k) is **INITIALIZED**

and [OdometerAxleMotorized](#OdometerAxleMotorized)(k)

and [TimeInSlipping](#TimeInSlipping)(k-1) <= [ATPsetting](#ATPsetting).SlippingTimeout

and (([StartSlippingSpeed](#StartSlippingSpeed)(k-1) + [TimeInSlipping](#TimeInSlipping)(k-1)\* [ATPsetting](#ATPsetting).SlippingStopAcc)

> [WheelMinSpeed](#WheelMinSpeed)(k))

and [SlippingEnded](#SlippingEnded)(k))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **MOTORING**

#

@

SlippingExcess，测得的加速度在项目配置范围内满足一定时间，是ATP判断过度空转的必要条件之一。

%

def [SlippingExcess](#SlippingExcess)(k):

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is not **SLIPPING**):

slipping\_excess\_counter = **0**

return **False**

elif ([FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) < ATPsetting.SlippingStopAcc

and [FilteredWheelAcceleration](#FilteredWheelAcceleration)(k) > ATPsetting.SlidingStopAcc):

slipping\_excess\_counter = slipping\_excess\_counter + **1**

return (slipping\_excess\_counter >= ATPsetting.SlippingExcessTime)

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from **SLIPPING** to **SKIDDING** when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **SLIPPING**

and [OdometerState](#OdometerState)(k) is **INITIALIZED**

and [OdometerAxleMotorized](#OdometerAxleMotorized)(k)

and ([TimeInSlipping](#TimeInSlipping)(k-1) > [ATPsetting](#ATPsetting).SlippingTimeout

or ((([StartSlippingSpeed](#StartSlippingSpeed)(k-1) + [TimeInSlipping](#TimeInSlipping)(k-1)\* [ATPsetting](#ATPsetting).SlippingStopAcc)

<= [WheelMinSpeed](#WheelMinSpeed)(k))

and [SlippingExcess](#SlippingExcess)(k))))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **SKIDDING**

#

$

The [MotionUnderEstimationState](#MotionUnderEstimationState) transfers from **SKIDDING** to **COASTING** when:

%

if ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **SKIDDING**

and ([WheelFilteredStopped](#WheelFilteredStopped)(k)

or [OdometerState](#OdometerState)(k) is **INVALID**))

[MotionUnderEstimationState](#MotionUnderEstimationState) = **COASTING**

#

@

StartMotoringMovementMin，记录由**COASTING**进入**MOTORING**，**COASTING**进入**SLIPPING**，或者**MOTORING**进入**SLIPPING**状态时的最小位移。

$

ATP records the minimum movement when the state transferring from **COASTING** to **MOTORING** or **SLIPPING**, or from **MOTORING** to **SLIPPING**.

%

def [StartMotoringMovementMin](#StartMotoringMovementMin)(k):

if (Initialization

or [OdometerState](#OdometerState)(k-1) is not **INITIALIZED**

or ([MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **COASTING**)):

return 0

elif (([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **COASTING**

and [MotionUnderEstimationState](#MotionUnderEstimationState) (k) is **MOTORING**)

or ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **COASTING**

and [MotionUnderEstimationState](#MotionUnderEstimationState) (k) is **SLIPPING**)

or ([MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is **MOTORING**

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **SLIPPING**)):

return [MinimumTrainMotion](#MinimumTrainMotion)(k-1)

else:

return [StartMotoringMovementMin](#StartMotoringMovementMin)(k-1)

#

@

UnderestimatedMotionMin，根据空转状态机，对里程计测得的最小位移进行补偿。在**MOTORING**状态下，使用牵引入口位移和将测得位移低估15%补偿后二者较大的一个，作为补偿后的位移。在**SLIPPING**状态下，使用牵引入口位移作为补偿后的位移。

%

def [UnderestimatedMotionMin](#UnderestimatedMotionMin)(k):

if [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **MOTORING**:

if ([WheelMinimumMovement](#WheelMinimumMovement)(k) >= 0):

return max(abs([StartMotoringMovementMin](#StartMotoringMovementMin)(k)),

abs([WheelMinimumMovement](#WheelMinimumMovement)(k) \* ATPsetting.SlippingCoefficient))

else:

return -1 \* max(abs([StartMotoringMovementMin](#StartMotoringMovementMin)(k)),

abs([WheelMinimumMovement](#WheelMinimumMovement)(k) \* ATPsetting.SlippingCoefficient))

elif [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **SLIPPING**:

return [StartMotoringMovementMin](#StartMotoringMovementMin)(k)

else:

return [WheelMinimumMovement](#WheelMinimumMovement)(k)

#

@

UnderestimatedMotionMax，根据空转状态机，对里程计测得的最大位移进行补偿。

%

def [UnderestimatedMotionMax](#UnderestimatedMotionMax)(k):

return [WheelMaximumMovement](#WheelMaximumMovement)(k)

#

@

SlipSlideDetected，是否检测到打滑空转

$

For calibration validation purpose, ATP shall consider that slip/side detected if:

motion overestimation modeling status is not coasting nor braking,or motion underestimation modeling status is not coasting nor motoring.

%

[SlipSlideDetected](#SlipSlideDetected)(k)

= (([MotionOverEstimationState](#MotionOverEstimationState)(k) != **COASTING**

and [MotionOverEstimationState](#MotionOverEstimationState)(k) != **BRAKING**)

or ([MotionUnderEstimationState](#MotionUnderEstimationState)(k) != **COASTING**

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k) != **MOTORING**))

#

@

SlipSlideModellingFault，打滑补偿模型错误

$

When the overestimation or underesimation state is **SKIDDING**, or the motion signed changed in **BRAKING** or **SLIDING** state, ATP shall consider the overestimation model as fault.

%

def [SlipSlideModellingFault](#SlipSlideModellingFault)(k):

if ([MotionOverEstimationState](#MotionOverEstimationState)(k) is **SKIDDING**

or [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **SKIDDING**

or (([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **BRAKING**

or [MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **SLIDING**)

and [MaxMotionOdometerSignChanged](#MaxMotionOdometerSignChanged)(k)))):

return **True**

elif ([MotionOverEstimationState](#MotionOverEstimationState)(k-1) is **COASTING**

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k-1) is COASTING)

return **False**

else:

return [SlipSlideModellingFault](#SlipSlideModellingFault)(k-1)

#

@

ValidSlipSlideModelling，打滑补偿模型有效

$

If overestimation model was fault, then ATP considers the model invalid.

%

def [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k):

if ([ValidSlipSlideModelling](#ValidSlipSlideModelling)(k-1))

return not [SlipSlideModellingFault](#SlipSlideModellingFault)(k)

elif (([MotionOverEstimationState](#MotionOverEstimationState)(k) is **COASTING**)

and [MotionUnderEstimationState](#MotionUnderEstimationState)(k) is **COASTING**

and [WheelFilteredStopped](#WheelFilteredStopped)(k))

return **True**

else:

return [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k-1)

#

@

MaximumSScompensatedMotion，经过打滑空转补偿后的最大位移

%

def [MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k):

if [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k):

return (sign([OverestimatedMotionMax](#OverestimatedMotionMax)(k))

\* max(abs([OverestimatedMotionMax](#OverestimatedMotionMax)(k)), abs([UnderestimatedMotionMax](#UnderestimatedMotionMax)(k)))

else:

return [WheelMaximumMovement](#WheelMaximumMovement)(k)

#

@

MinimumSScompensatedMotion，经过打滑空转补偿后的最小位移

%

def [MinimumSScompensatedMotion](#MinimumSScompensatedMotion)(k):

if [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k):

return (sign([OverestimatedMotionMin](#OverestimatedMotionMin)(k))

\* min(abs([OverestimatedMotionMin](#OverestimatedMotionMin)(k)), abs([UnderestimatedMotionMin](#UnderestimatedMotionMin)(k)))

else:

return [WheelMinimumMovement](#WheelMinimumMovement)(k)

#

@

RadarRawSpeed，直接获取的雷达测速值，该值始终为正。

%

def [RadarRawSpeed](#RadarRawSpeed)(k):

if [RadarInfo](#RadarInfo)(k).DrsValid:

return [RadarInfo](#RadarInfo)(k).DrsSpeed

else:

return **MAX\_RADAR\_SPEED**

#

@

RadarDirection，雷达测得方向，向End1方向为+1，向End2为-1，其余为0

%

def [RadarDirection](#RadarDirection)(k):

if not [RadarInfo](#RadarInfo)(k).DrsValid:

return 0

elif (([CoreId](#CoreId)(k) is **END\_1**

and [RadarInfo](#RadarInfo)(k).DrsDirection > 0)

or ([CoreId](#CoreId)(k) is **END\_2**

and [RadarInfo](#RadarInfo)(k).DrsDirection <= 0)):

return 1

else:

return -1

#

@

RadarMotionMax，绝对值向上过估的雷达最大位移，向END1方向该值为正，向END2方向该值为负。

%

def [RadarMotionMax](#RadarMotionMax)(k):

if not [RadarInfo](#RadarInfo)(k).DrsValid:

return 0

elif [RadarRawSpeed](#RadarRawSpeed)(k) >= [ATPsetting](#ATPsetting).RadarSpeedThreshold:

return (**ATP\_CYCLE\_TIME** \* [RadarDirection](#RadarDirection)(k) \* ([RadarRawSpeed](#RadarRawSpeed)(k)

+ [RadarRawSpeed](#RadarRawSpeed)(k) \* [ATPsetting](#ATPsetting).RadarDeviationAboveThreshold / **1000**))

else:

return (**ATP\_CYCLE\_TIME** \* [RadarDirection](#RadarDirection)(k)

\* ([RadarRawSpeed](#RadarRawSpeed)(k) + [ATPsetting](#ATPsetting).RadarDeviationBelowThreshold))

#

@

RadarMotionMin，绝对值向下过估的雷达最小位移，向END1方向该值为正，向END2方向该值为负。

%

def [RadarMotionMin](#RadarMotionMin)(k):

if not [RadarInfo](#RadarInfo)(k).DrsValid:

return 0

elif [RadarRawSpeed](#RadarRawSpeed)(k) >= [ATPsetting](#ATPsetting).RadarSpeedThreshold:

return (**ATP\_CYCLE\_TIME** \* [RadarDirection](#RadarDirection)(k) \* ([RadarRawSpeed](#RadarRawSpeed)(k)

- [RadarRawSpeed](#RadarRawSpeed)(k) \* [ATPsetting](#ATPsetting).RadarDeviationAboveThreshold / **1000**))

else:

return (**ATP\_CYCLE\_TIME** \* [RadarDirection](#RadarDirection)(k)

\* max(**0**, [RadarRawSpeed](#RadarRawSpeed)(k) - [ATPsetting](#ATPsetting).RadarDeviationBelowThreshold))

#

@

RadarSpeedValid，判断雷达速度是否可用

%

def [RadarSpeedValid](#RadarSpeedValid)(k):

return ([ATPsetting](#ATPsetting).RadarApplied

and [RadarInfo](#RadarInfo)(k).DrsValid

and (not [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k)

or ([RadarDirection](#RadarDirection)(k) \* [MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k) >= 0

and abs([RadarMotionMin](#RadarMotionMin)(k)) <= abs([MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k))

and abs([RadarMotionMax](#RadarMotionMax)(k)) >= abs([MinimumSScompensatedMotion](#MinimumSScompensatedMotion)(k)))))

#

@

OdometerSpeedAvailable，当前里程计测速是否可用于参考速度判断

%

[OdometerSpeedAvailable](#OdometerSpeedAvailable)(k):

return ([ValidWheelKinematic](#ValidWheelKinematic)(k)

and [OdometerState](#OdometerState)(k) is **INITIALIZED**)

#

@

OdometerSpeedUnderThreshold，本端里程计测速低于阈值。

$

ATP shall detect whether the measured wheel speed is under threshold.

%

def [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k):

return ([WheelMinSpeed](#WheelMinSpeed)(k) < ATPsetting.OdoLockedAxleThresholdSpeed)

#

@

ReferenceSpeedUnderThreshold\_1，来自CCNV的参考速度1是否小于指定阈值。

$

[ReferenceSpeedUnderThreshold\_1](#ReferenceSpeedUnderThreshold_1) defines whether the referenced speed 1 from CCNV is lower than a configurable threshold.

%

def [ReferenceSpeedUnderThreshold\_1](#ReferenceSpeedUnderThreshold_1)(k):

if [RadarSpeedValid](#RadarSpeedValid)(k):

return ([RadarRawSpeed](#RadarRawSpeed)(k) < [ATPsetting](#ATPsetting).OdoLockedAxleThresholdSpeed)

else:

return ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest).OdometerRef1SpeedUnderThreshold(k))

#

@

ReferenceSpeedAvailable\_1，来自CCNV的参考速度1是否可用

$

[ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1) defines whether the referenced speed 1 from CCNV is valid or not.

%

def [ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k):

return ([RadarSpeedValid](#RadarSpeedValid)(k)

or ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest).OdometerRef1Available(k)))

#

@

ReferenceSpeedUnderThreshold\_2，来自CCNV的参考速度2是否小于指定阈值。

$

[ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2) defines whether the referenced speed 2 from CCNV is lower than a configurable threshold.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2)(k) = [NonVitalRequest](#NonVitalRequest).OdometerRef2SpeedUnderThreshold(k)

else:

[ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2) = **False**

#

@

ReferenceSpeedAvailable\_2，来自CCNV的参考速度2是否可用

$

[ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2) shows whether the referenced speed 2 from CCNV is effective or not.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k) = [NonVitalRequest](#NonVitalRequest).OdometerRef2Available(k)

else:

[ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2) = **False**

#

@

[OdometerRef\_1](#OdometerRef_1).[PossiblyDisabled](#PossiblyDisabled)，当本端里程计可用且不为0速，而参考速度1可用但为0速时，则认为参考速度1可能错误

$

The independent source of odometry reference 1 said to disable if following conditions reached: local source of odometry is available ([ValidWheelKinematic](#ValidWheelKinematic)),and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that train speed is greater than reference speed threshold,and source of odometry reference 1 is available,and odometer reference 1 indicates that train speed is less than reference speed threshold.

%

[OdometerRef\_1](#OdometerRef_1). [PossiblyDisabled](#PossiblyDisabled)(k)

= ([ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k)

and [ReferenceSpeedUnderThreshold\_1](#ReferenceSpeedUnderThreshold_1)(k)

and [OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)

and not [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

[OdometerRef\_2](#OdometerRef_2). [PossiblyDisabled](#PossiblyDisabled)，当本端里程计可用且不为0速，而参考速度2可用但为0速时，则认为参考速度2可能错误

$

The independent source of odometry reference 2 said to disable if following conditions reached:local source of odometry is available ([ValidWheelKinematic](#ValidWheelKinematic)),and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that train speed is greater than reference speed threshold,and source of odometry reference 2 is available,and odometer reference 2 indicates that train speed is less than reference speed threshold.

%

[OdometerRef\_2](#OdometerRef_2). [PossiblyDisabled](#PossiblyDisabled)(k)

= ([ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k)

and [ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2)(k)

and [OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)

and not [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

[OdometerRef\_1](#OdometerRef_1).[PossiblyEnabled](#PossiblyEnabled)，当本端里程计和参考速度1均可用且测得列车在动时，认为参考速度1可能已恢复有效。

$

The independent source of odometry reference 1 said to enable if following conditions reached:local source of odometry is available,and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that train speed is greater than reference speed threshold,and source of odometry reference 1 is available,

and odometer reference 1 indicates that train speed is greater than reference speed threshold.

%

[OdometerRef\_1](#OdometerRef_1). [PossiblyEnabled](#PossiblyEnabled)(k)

= ([ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k)

and not [ReferenceSpeedUnderThreshold\_1](#ReferenceSpeedUnderThreshold_1)(k)

and [OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)

and not [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

[OdometerRef\_2](#OdometerRef_2).[PossiblyEnabled](#PossiblyEnabled)，当本端里程计和参考速度2均可用且测得列车在动时，认为参考速度2可能已恢复有效。

$

The independent source of odometry reference 2 said to enable if following conditions reached:local source of odometry is available,and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that train speed is greater than reference speed threshold,and source of odometry reference 2 is available,

and odometer reference 2 indicates that train speed is greater than reference speed threshold.

%

[OdometerRef\_2](#OdometerRef_2). [PossiblyEnabled](#PossiblyEnabled)(k)

= ([ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k)

and not [ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2)(k)

and [OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)

and not [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

[OdometerRef\_1](#OdometerRef_1).[OutOfOrder](#OutOfOrder)，当判断参考速度1可能不可用时，延迟一段时间，若仍不可用，则判断参考速度1失效。当判断参考速度1可能可用时，延迟一段时间，若仍可用，则判断参考速度1有效

$

The independent source of odometry reference 1 is said to be out of order if it is possibly disabled during more than [ATPsetting](#ATPsetting).OdoLockedAxleDisablingLatency. When the independent source of odometry reference 1 had out of order, it considered not out of order one if the source of odometry reference 1 is possibly enabled during more than [ATPsetting](#ATPsetting).OdoLockedAxleEnablingLatency:

%

if ([OdometerRef\_1](#OdometerRef_1).[PossiblyDisabled](#PossiblyDisabled)(k) == **True**

and [OdometerRef\_1](#OdometerRef_1).[PossiblyDisabled](#PossiblyDisabled)(k-1) == **True**

...

and [OdometerRef\_1](#OdometerRef_1).[PossiblyDisabled](#PossiblyDisabled)(k+1-[ATPsetting](#ATPsetting).OdoLockedAxleDisablingLatency) == **True**)

[OdometerRef\_1](#OdometerRef_1).OutOfOrder = **True**

if ([OdometerRef\_1](#OdometerRef_1).[PossiblyEnabled](#PossiblyEnabled)(k) == **True**

and [OdometerRef\_1](#OdometerRef_1).[PossiblyEnabled](#PossiblyEnabled)(k-1) == **True**

...

and [OdometerRef\_1](#OdometerRef_1).[PossiblyEnabled](#PossiblyEnabled)(k+1-[ATPsetting](#ATPsetting).OdoLockedAxleEnablingLatency) == **True**)

[OdometerRef\_1](#OdometerRef_1).[OutOfOrder](#OutOfOrder) = **False**

#

@

[OdometerRef\_2](#OdometerRef_2).[OutOfOrder](#OutOfOrder)，当判断参考速度2可能不可用时，延迟一段时间，若仍不可用，则判断参考速度2失效。当判断参考速度2可能可用时，延迟一段时间，若仍可用，则判断参考速度2有效。

$

The independent source of odometry reference 2 is said to be out of order if it is possibly disabled during more than [ATPsetting](#ATPsetting).OdoLockedAxleDisablingLatency. When the independent source of odometry reference 2 had out of order, It considered not out of order one if the source of odometry reference 2 is possibly enabled during more than [ATPsetting](#ATPsetting).OdoLockedAxleEnablingLatency:

%

if ([OdometerRef\_2](#OdometerRef_2).[PossiblyDisabled](#PossiblyDisabled)(k) == **True**

and [OdometerRef\_2](#OdometerRef_2).[PossiblyDisabled](#PossiblyDisabled)(k-1) == **True**

...

and [OdometerRef\_2](#OdometerRef_2).[PossiblyDisabled](#PossiblyDisabled)(k+1-[ATPsetting](#ATPsetting).OdoLockedAxleDisablingLatency) == **True**)

[OdometerRef\_2](#OdometerRef_2).[OutOfOrder](#OutOfOrder) = **True**

if ([OdometerRef\_2](#OdometerRef_2).[PossiblyEnabled](#PossiblyEnabled)(k) == **True**

and [OdometerRef\_2](#OdometerRef_2).[PossiblyEnabled](#PossiblyEnabled)(k-1) == **True**

...

and [OdometerRef\_2](#OdometerRef_2).[PossiblyEnabled](#PossiblyEnabled)(k+1-[ATPsetting](#ATPsetting).OdoLockedAxleEnablingLatency) == **True**)

[OdometerRef\_2](#OdometerRef_2).OutOfOrder = **False**

#

@

[OdometerRef\_1](#OdometerRef_1).[Contradictory](#Contradictory)，若参考速度1有效且判断车动，而本端里程计判断车静止，则认为参考速度1判断出里程计可能故障。

$

The source of odometry reference 1 said to be contradictory with local source of odometry if:local source of odometry is available ([ValidWheelKinematic](#ValidWheelKinematic))

and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that wheel speed is less than reference speed threshold,and source of odometry reference 1 is available and not out of order,and odometer reference 1 indicates that train speed is greater than reference speed threshold.

%

[OdometerRef\_1](#OdometerRef_1). [Contradictory](#Contradictory)(k)

= (not [OdometerRef\_1](#OdometerRef_1).OutOfOrder(k)

and [ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k)

and not [ReferenceSpeedUnderThreshold\_1](#ReferenceSpeedUnderThreshold_1)(k)

and [ValidWheelKinematic](#ValidWheelKinematic)(k)

and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

[OdometerRef\_2](#OdometerRef_2). [Contradictory](#Contradictory)，若参考速度2有效且判断车动，而本端里程计判断车静止，则认为参考速度2判断出里程计可能故障。

$

The source of odometry reference 2 said to be contradictory with local source of odometry if:local source of odometry is available ([ValidWheelKinematic](#ValidWheelKinematic))

and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold) indicates that wheel speed is less than reference speed threshold,and source of odometry reference 2 is available and not out of order,and odometer reference 2 indicates that train speed is greater than reference speed threshold.

%

[OdometerRef\_2](#OdometerRef_2).[Contradictory](#Contradictory) (k)

= (not [OdometerRef\_2](#OdometerRef_2).OutOfOrder(k)

and [ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k)

and not [ReferenceSpeedUnderThreshold\_2](#ReferenceSpeedUnderThreshold_2)(k)

and [ValidWheelKinematic](#ValidWheelKinematic)(k)

and [OdometerSpeedUnderThreshold](#OdometerSpeedUnderThreshold)(k))

#

@

AxlePossiblyLocked，在两路参考速度都正常（没有失效out of order）的情况下，当两路参考速度均判断本端里程计可能故障的情况下，认为当前可能轴锁。或者，当有一路参考速度认为轴锁，而另一路参考速度失效或不可用，也认为当前可能轴锁。

$

Odometer axle shall consider possibly locked if:Both independent sources of odometry indicates a contradiction with local odometer,Or one source of odometry is contradictory and the other one is out of order (or not available).

%

[AxlePossiblyLocked](#AxlePossiblyLocked)(k)

= (([OdometerRef\_1](#OdometerRef_1).[Contradictory](#Contradictory)(k) and [OdometerRef\_2](#OdometerRef_2).[Contradictory](#Contradictory)(k))

or ([OdometerRef\_1](#OdometerRef_1).[Contradictory](#Contradictory)(k)

and ([OdometerRef\_2](#OdometerRef_2).[OutOfOrder](#OutOfOrder)(k) or not [ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k)))

or ([OdometerRef\_2](#OdometerRef_2).[Contradictory](#Contradictory)(k)

and ([OdometerRef\_1](#OdometerRef_1).[OutOfOrder](#OutOfOrder)(k) or not [ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k))))

#

@

UnrecoverableAxleLocked，当连续若干个周期判断可能轴锁，或者已经判断为轴锁，则永久轴锁.

$

If [AxlePossiblyLocked](#AxlePossiblyLocked) situation lasts more than [ATPsetting](#ATPsetting).OdoLockedAxleTimeout, the odometer axle shall be considered locked. Once [UnrecoverableAxleLocked](#UnrecoverableAxleLocked) set to **True**, it will stay at state **True** unless ATP re-initialized.

%

[UnrecoverableAxleLocked](#UnrecoverableAxleLocked)(k)

= [UnrecoverableAxleLocked](#UnrecoverableAxleLocked)(k-1)

or ([AxlePossiblyLocked](#AxlePossiblyLocked)(k)

and [AxlePossiblyLocked](#AxlePossiblyLocked)(k-1)

and ...

and [AxlePossiblyLocked](#AxlePossiblyLocked)(k+1-[ATPsetting](#ATPsetting).OdoLockedAxleTimeout)

#

@

AxleLockedDetectionAvailable，只要有一路参考速度可以工作，就认为轴锁侦测可用。

$

If only one or no source of odometry is available, then ATP shall invalidate kinematic while this situation lasting.

%

[AxleLockedDetectionAvailable](#AxleLockedDetectionAvailable)

= ((not [OdometerRef\_1](#OdometerRef_1).[OutOfOrder](#OutOfOrder) and [ReferenceSpeedAvailable\_1](#ReferenceSpeedAvailable_1)(k))

or (not [OdometerRef\_2](#OdometerRef_2).[OutOfOrder](#OutOfOrder) and [ReferenceSpeedAvailable\_2](#ReferenceSpeedAvailable_2)(k)))

#

@

WheelTrainKinematicCorrelation，车轮和列车的速度一致性

$

Wheel and train kinematic shall consider correctly correlated if and only if:

odometer axle is not detected locked,and odometer axle detection is available

%

[WheelTrainKinematicCorrelation](#WheelTrainKinematicCorrelation)(k)

= [AxleLockedDetectionAvailable](#AxleLockedDetectionAvailable)(k) and not [UnrecoverableAxleLocked](#UnrecoverableAxleLocked)(k)

#

@

ValidTrainKinematic，列车位移速度计算有效的条件

$

Train kinematic information shall declare invalid if at least one of following condition is **True**:odometer kinematic is not valid,or neither radar speed nor over-estimation modeling of train movement is valid,or train movement and wheel movement has been detected de-correlated.

%

def [ValidTrainKinematic](#ValidTrainKinematic)(k):

return (([ValidSlipSlideModelling](#ValidSlipSlideModelling)(k)

or [RadarSpeedValid](#RadarSpeedValid)(k))

and [WheelTrainKinematicCorrelation](#WheelTrainKinematicCorrelation)(k)

and [ValidWheelKinematic](#ValidWheelKinematic)(k))

#

@

MaximumTrainMotion，根据来自里程计或雷达的信息，计算列车的周期最大位移。该值为矢量，向**END\_1**方向为正，**END\_2**方向为负。

%

def [MaximumTrainMotion](#MaximumTrainMotion)(k):

if (not [RadarSpeedValid](#RadarSpeedValid)(k)):

return [MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k)

elif (not [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k)):

return [RadarMotionMax](#RadarMotionMax)(k)

else:

return (min(abs([MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k)), abs([RadarSpeedMax](#RadarSpeedMax)(k)))

\* sign([MaximumSScompensatedMotion](#MaximumSScompensatedMotion)(k)))

#

@

MinimumTrainMotion，根据来自里程计或雷达的信息，计算列车的周期最小位移。该值为矢量，向**END\_1**方向为正，**END\_2**方向为负。

%

def [MinimumTrainMotion](#MinimumTrainMotion)(k):

if (not [RadarSpeedValid](#RadarSpeedValid)(k)):

return [MinimumSScompensatedMotion](#MinimumSScompensatedMotion)(k)

elif (not [ValidSlipSlideModelling](#ValidSlipSlideModelling)(k)):

return [RadarMotionMin](#RadarMotionMin)(k)

else:

return (max(abs([[MinimumSScompensatedMotion](#MinimumSScompensatedMotion)](#OverestimatedMotionMin)(k)), abs([RadarSpeedMin](#RadarSpeedMin)(k)))

\* sign([[MinimumSScompensatedMotion](#MinimumSScompensatedMotion)](#OverestimatedMotionMin)(k)))

#

$

TrainStopped, train shall consider strictly stopped if and only if:wheel is detected strictly stopped,and train kinematic elaboration is valid,and wheel is not detected sliding.

%

[TrainStopped](#TrainStopped)(k)

= (([WheelStopped](#WheelStopped)(k) == **True**)

and (([MotionOverEstimationState](#MotionOverEstimationState) == **COASTING**)

or ([MotionOverEstimationState](#MotionOverEstimationState) == **BRAKING**))

and ([ValidTrainKinematic](#ValidTrainKinematic)(k) == **True**))

#

@

TrainFilteredStopped，列车准静止判断。

$

Train shall consider stopped with the tolerance of one cog detection if:wheel is detected at filtered stop,and train kinematic elaboration is valid,and wheel is not detected sliding.

%

[TrainFilteredStopped](#TrainFilteredStopped)(k)

= (([WheelFilteredStopped](#WheelFilteredStopped)(k) == **True**)

and (([MotionOverEstimationState](#MotionOverEstimationState) == **COASTING**)

or ([MotionOverEstimationState](#MotionOverEstimationState) == **BRAKING**))

and ([ValidTrainKinematic](#ValidTrainKinematic)(k) == **True**))

#

@

TrainHasMoved，表明自上电以后，列车是否移动过。

%

def [TrainHasMoved](#TrainHasMoved)(k):

if (Initialization):

return **False**

elif (not [TrainHasMoved](#TrainHasMoved)(k-1)

and not [TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([TeethCounter](#TeethCounter)(k) != [TeethCounter](#TeethCounter)(k-1))):

return **True**

else:

return [TrainHasMoved](#TrainHasMoved)(k-1)

#

@

TrainMinSpeed，计算列车最小速度。

%

def TrainMinSpeed(k):

if ([OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)):

return [max](#max)(0, ([round.floor](#roundfloor)([abs](#abs)([MinimumTrainMotion](#MinimumTrainMotion)(k) / ATP\_CYCLE\_TIME))

+ (([ATPsetting](#ATPsetting).BrakingMinAcc - [ATPsetting](#ATPsetting).MaxGradientAcc)

\* ATP\_CYCLE\_TIME / 2)))

else:

return 0

#

@

TrainMaxSpeed，考虑打滑过估补偿的列车最大速度，该速度为非负值。

$

According to the matching of odometer cog counter and code, maximum train speed shall computed using the followings expressions:

%

def [TrainMaxSpeed](#TrainMaxSpeed)(k):

if ([OdometerSpeedAvailable](#OdometerSpeedAvailable)(k)):

return ([round.ceil](#roundceil)([abs](#abs)([MaximumTrainMotion](#MaximumTrainMotion)(k) / **ATP\_CYCLE\_TIM**E))

+ (([ATPsetting](#ATPsetting).TractionMaxAcc[[TrainMinSpeed](#TrainMinSpeed)(k)] + [ATPsetting](#ATPsetting).MaxGradientAcc)

\* **ATP\_CYCLE\_TIME** / 2))

else:

return [round.ceil](#roundceil)([abs](#abs)([MaximumTrainMotion](#MaximumTrainMotion)(k)) / **ATP\_CYCLE\_TIME**)

#

@

NewBeaconObtained，表明收到了可用的RB

%

def [NewBeaconObtained](#NewBeaconObtained)(k):

if ([BeaconMessageReceive](#BeaconMessageReceive)(k)

and [ValidTrainKinematic](#ValidTrainKinematic)(k)

and not [TrainFilteredStopped](#TrainFilteredStopped)(k)

and [OdometerState](#OdometerState)(k) is **INITIALIZED**):

[NewBeaconObtained](#NewBeaconObtained) = **True**

else:

[NewBeaconObtained](#NewBeaconObtained) = **False**

#

@

BeaconBeforeLastObtained，记录读到的次新的信标

%

def [BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k):

if (Initialization):

[BeaconBeforeLastObtained](#BeaconBeforeLastObtained) = **None**

elif ([NewBeaconObtained](#NewBeaconObtained)(k)):

[BeaconBeforeLastObtained](#BeaconBeforeLastObtained) = [BeaconLastObtained](#BeaconLastObtained)(k-1)

else:

[BeaconBeforeLastObtained](#BeaconBeforeLastObtained) = [BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k-1)

return [BeaconBeforeLastObtained](#BeaconBeforeLastObtained)

#

@

BeaconLastObtained，记录读到的最新的信标

%

def [BeaconLastObtained](#BeaconLastObtained)(k):

if (Initialization):

[BeaconLastObtained](#BeaconLastObtained) = **None**

elif ([NewBeaconObtained](#NewBeaconObtained)(k)):

[BeaconLastObtained](#BeaconLastObtained) = [TrackMap](#TrackMap).[Beacons](#Beacons)[[LockedBeaconMsgByte](#LockedBeaconMsgByte).Id]

else:

[BeaconLastObtained](#BeaconLastObtained) = [BeaconLastObtained](#BeaconLastObtained)(k-1)

return [BeaconLastObtained](#BeaconLastObtained)

#

@

在本周期的里程计和测速信息有效的情况下，需要根据当前齿数和锁存的读到信标时的Top-loc信息，计算DistLastBeaconMax和DistLastBeaconMin，表示当前经过信标后已运行的最大最小距离。

$

If a valid beacon with top-loc received between cycle k-1 and k, then minimum and maximum distance ran since top-loc shall evaluate as the difference between current teeth counter and recorded cog position just before or after top-loc. If there is no beacon received, ATP updates distances from last beacon using the train movements.

%

if ([NewBeaconObtained](#NewBeaconObtained)(k)):

[DistLastBeaconMin](#DistLastBeaconMin)(k)= [MinCogCalibration](#MinCogCalibration)(k) \* ([TeethCounter](#TeethCounter)(k)- [CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k))

[DistLastBeaconMax](#DistLastBeaconMax)(k)= [MaxCogCalibration](#MaxCogCalibration)(k) \* ([TeethCounter](#TeethCounter)(k)- [CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k))

else:

[DistLastBeaconMin](#DistLastBeaconMin) = [DistLastBeaconMin](#DistLastBeaconMin)(k-1) + [MinimumTrainMotion](#MinimumTrainMotion)(k)

[DistLastBeaconMax](#DistLastBeaconMax) = [DistLastBeaconMax](#DistLastBeaconMax)(k-1) + [MaximumTrainMotion](#MaximumTrainMotion)(k)

#

@

End2RunningForward，根据车轮旋转方向，判断列车是否向**END\_2**方向运行

%

def [End2RunningForward](#End2RunningForward)(k):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

return ([MaximumTrainMotion](#MaximumTrainMotion)(k) < 0)

else:

return **True**

#

@

End1RunningForward，根据车轮旋转方向，判断列车是否向**END\_1**方向运行

%

def [End1RunningForward](#End1RunningForward)(k):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

return ([MaximumTrainMotion](#MaximumTrainMotion)(k) > 0)

else:

return **True**

#

@

在**CALI\_WATING**状态下使用默认齿距值

$

From power-up and while calibration process is not successfully performed, ATP shall use default calibration to compute train motion and shall consider itself in the state of waiting for the first beacon belonging to a couple of calibration.

%

if (Initialization)

[CalibrationState](#CalibrationState) = **CALI\_WAITING**

[MinCogCalibration](#MinCogCalibration) = [ATPsetting](#ATPsetting).OdoCaliDefaultCogLengthMin

[MaxCogCalibration](#MaxCogCalibration) = [ATPsetting](#ATPsetting).OdoCaliDefaultCogLengthMax

#

@

当读到线路地图中的MTIB1时，齿距校准状态从**CALI\_WAITING**转入**CALI\_MEASURING**。

$

If ATP is in the state of **CALI\_WAITING**, can transform to the measuring state if following conditions fulfilled:a valid beacon has been received and this beacon belongs to a couple of calibration,and train kinematic was valid,and no excessive slip/slide effect was detected,and [WheelMinimumMovement](#WheelMinimumMovement) is not null,Then, ATP shall memorize:position of the wheel before and after top location signal of received beacon,the ID of received beacon,the sign of the movement when crossing beacon,and shall consider itself as **CALI\_MEASURING**.

%

if (([CalibrationState](#CalibrationState)(k-1) = **CALI\_WAITING**)

and ([NewBeaconObtained](#NewBeaconObtained)(k) == **True**)

and ([TrackMap](#TrackMap).[BeaconBelongsToCalibrationCouple](#BeaconBelongsToCalibrationCouple)([BeaconMessage](#BeaconMessage).Id(k)) == **True**)

and ([SlipSlideDetected](#SlipSlideDetected)(k) == **False**))

CalibrationMeasurementStartPositionMin = [CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k)

CalibrationMeasurementStartPositionMax = [CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k)

CalibrationMeasurementStartBeacon = [BeaconMessage](#BeaconMessage).Id(k)

CalibrationEnd1RunningForward = [End1RunningForward](#End1RunningForward)(k)

[CalibrationState](#CalibrationState) = **CALI\_MEASURING**

#

@

当读到线路地图中与之前MTIB1匹配的MTIB2时，根据经过的齿数值，在[ATPsetting](#ATPsetting)中查表得到最大最小齿距，并比较测得的齿距结果：如果测得的最大最小齿距在理论值范围内，则齿距校准状态从**CALI\_MEASURING**到**CALI\_VALIDATING**；如果在理论范围外，则状态从**CALI\_MEASURING**转入**CALI\_WAITING**。其中最大最小测得齿距是根据校准过程中测得齿数，在离线工具计算的齿数-齿距对照表中查到的。其中CaliMinRatio和CaliMaxRatio是离线工具计算的该校准信标所对应的MTIB结构的相关属性，表示该对MTIB的间距与标准间距（21米）的比率，该结构定义见[REF11]。

$

If ATP is in the state of **CALI\_MEASURING and** following conditions fulfilled: a valid beacon has been received and the beacon and first memorized calibration beacon is one of possible dedicated couple of calibration,and train kinematic was valid,and no excessive slip/slide effect was detected,and sign of train motion is still identical to thus detected on first beacon signaling,

Then,if resulting calibration range is included in default calibration range, then ATP shall: memorize: position of the wheel before and after top location signal of received beacon, the id of received beacon,and shall consider itself as **CALI\_VALIDATING**. else: ATP shall consider that calibration process has failed and back to **CALI\_WAITING**. Among them, the measured maximum and minimum calibration fetches from the offline-generated counter-calibration table, based on the calculated cog counter. The CaliMinRation and CaliMaxRation are elements of structure MTIB generated by offline tool for each couple of calibration beacons. Refer to [REF11] for the definition of MTIB.

%

if (([CalibrationState](#CalibrationState)(k-1) == **CALI\_MEASURING)**

and ([NewBeaconObtained](#NewBeaconObtained)(k) == **True**)

and ([TrackMap](#TrackMap).[AreNeighbouredBeacons](#AreNeighbouredBeacons)([BeaconMessage](#BeaconMessage).Id(k),

CalibrationMeasurementStartBeacon(k)) == **True**)

and ([SlipSlideDetected](#SlipSlideDetected)(k) == **False**)

and ([CalibrationEnd1RunningForward](#CalibrationEnd1RunningForward)(k-1) == [End1RunningForward](#End1RunningForward)(k)))

if (([ATPsetting](#ATPsetting).OdoCaliDefaultCogLengthMax >= [MaxCogCalibrationMeasured](#MaxCogCalibrationMeasured))

and ([ATPsetting](#ATPsetting).OdoCaliDefaultCogLengthMin <= [MinCogCalibrationMeasured](#MinCogCalibrationMeasured)))

CalibrationValidationStartPositionMin = [CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k)

CalibrationValidationStartPositionMax = [CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k)

CalibrationValidationStartBeacon = [BeaconMessage](#BeaconMessage).Id(k))

CalibrationResultMin = [MinCogCalibrationMeasured](#MinCogCalibrationMeasured)

CalibrationResultMax = [MaxCogCalibrationMeasured](#MaxCogCalibrationMeasured)

[CalibrationState](#CalibrationState) = **CALI\_VALIDATING**

else:

[CalibrationState](#CalibrationState) = **CALI\_WAITING**

MinCogCalibrationMeasured

= CaliMinRatio \* [ATPsetting](#ATPsetting).MeterCaliMaxMinCalibration[1]

[|[CalibrationMeasurementStartPositionMin](#CalibrationMeasurementStartPositionMin)(k)-[CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k)|

- [ATPsetting](#ATPsetting).OdoCaliCogCounterMin]

MaxCogCalibrationMeasured

= CaliMaxRatio \* [ATPsetting](#ATPsetting).MeterCaliMaxMinCalibration[0]

[|[CalibrationMeasurementStartPositionMax](#CalibrationMeasurementStartPositionMax)(k)-[CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k)|

- [ATPsetting](#ATPsetting).OdoCaliCogCounterMin]

#

@

如果校准过程中发生下列情况，则从**CALI\_MEASURING**回到**CALI\_WAITING**

$

If ATP is in the state of **CALI\_MEASURING and** following conditions fulfilled:

train kinematic has been detected not valid,or excessive slip/slide effect has been detected,or [WheelMinimumMovement](#WheelMinimumMovement) sign is in the opposite direction of thus observed on first beacon or becomes null,or an unexpected beacon has been received. That is, a beacon not belonging calibration measurement couple.Then, ATP shall abort calibration process and back to **CALI\_WAITING**.

%

if (([CalibrationState](#CalibrationState)(k-1) = **CALI\_MEASURING**)

and (([ValidTrainKinematic](#ValidTrainKinematic)(k) == **False**)

or ([OdometerState](#OdometerState)(k) is **INVALID**)

or ([SlipSlideDetected](#SlipSlideDetected)(k) == **True**)

or ([CalibrationEnd1RunningForward](#CalibrationEnd1RunningForward)(k-1) != [End1RunningForward](#End1RunningForward)(k))

or ([TrainFilteredStopped](#TrainFilteredStopped)(k) == **True**)

or (([NewBeaconObtained](#NewBeaconObtained)(k) == **True**)

and [TrackMap](#TrackMap).BeaconBelongsToCalibrationCouple([BeaconMessage](#BeaconMessage).Id(k)) == **False**)))

[CalibrationState](#CalibrationState) = **CALI\_WAITING**

#

@

当发生以下情况时，认为校准失败，从**CALI\_VALIDATING**回到**CALI\_WAITING**

$

If ATP is in the state of calibration validation in progress and following conditions fulfilled:train kinematic has been detected not valid,or excessive slip/slide effect has been detected,or [WheelMinimumMovement](#WheelMinimumMovement) sign is in the opposite direction of thus observed on first beacon or becomes null,or an unexpected beacon has been received. That is, a beacon not belonging calibration validation couple.In such case, then ATP shall consider that calibration process as not sable and so back to **CALI\_WAITING**.

%

if (([CalibrationState](#CalibrationState)(k-1) = **CALI\_VALIDATING**)

and (([ValidTrainKinematic](#ValidTrainKinematic)(k) == **False**)

or ([OdometerState](#OdometerState)(k) is **INVALID**)

or ([SlipSlideDetected](#SlipSlideDetected)(k) == **True**)

or (([CalibrationEnd1RunningForward](#CalibrationEnd1RunningForward)(k-1) != [End1RunningForward](#End1RunningForward)(k))

or ([TrainFilteredStopped](#TrainFilteredStopped)(k) == **True**)

or (([NewBeaconObtained](#NewBeaconObtained)(k) == **True**)

and ([TrackMap](#TrackMap).[IsCalibrationValidationBeacon](#IsCalibrationValidationBeacon)([BeaconMessage](#BeaconMessage).Id(k),

[CalibrationValidationStartBeacon](#CalibrationValidationStartBeacon)(k)) == **False**))))

[CalibrationState](#CalibrationState) = **CALI\_WAITING**

#

@

当读到有效的验证信标，并判断之前测得齿距在有效范围内时，认为校准成功，转入**CALI\_COMPLETED**状态；否则，校准失败，返回**CALI\_WATING**状态。

$

If ATP is in the state of calibration validation in progress and following conditions fulfilled: a valid beacon has been received and this beacon is one of possible confirmation beacons related to second beacon signaled of possible confirmation beacons related to second beacon signaled of calibration measurement,and train kinematic was valid,and no excessive slip/slide effect was detected,and sign of train motion is still identical to thus detected on first beacon signaling,and [WheelMinimumMovement](#WheelMinimumMovement) is not null. Then,if resulting calibration range **fully includes** the calibration range in track map, then ATP shall: update ATP minimum and maximum calibration with last calibration computed on beacons,and shall consider that calibration process is **CALI\_COMPLETED**.else: ATP shall consider that calibration process is not usable and so back to **CALI\_WATING** waiting for new measurement calibration beacons.

%

if ([CalibrationState](#CalibrationState)(k-1) = **CALI\_VALIDATING**)

and ([[NewBeaconObtained](#NewBeaconObtained)](#BeaconMessageReceive)(k) == **True**)

and ([TrackMap](#TrackMap).[IsCalibrationValidationBeacon](#IsCalibrationValidationBeacon)([BeaconMessage](#BeaconMessage).Id(k),

[CalibrationValidationStartBeacon](#CalibrationValidationStartBeacon)(k)))

and ([SlipSlideDetected](#SlipSlideDetected)(k) == **False**)

and ([CalibrationEnd1RunningForward](#CalibrationEnd1RunningForward)(k-1) == [End1RunningForward](#End1RunningForward)(k)))

if (([TrackMap](#TrackMap).[CalibrationCoupleMaxDistance](#CalibrationCoupleMaxDistance)([BeaconMessage](#BeaconMessage).Id(k),

[CalibrationValidationStartBeacon](#CalibrationValidationStartBeacon)(k)) <= [MaxDistanceRanForValidation](#MaxDistanceRanForValidation))

and [TrackMap](#TrackMap).[CalibrationCoupleMinDistance](#CalibrationCoupleMinDistance)([BeaconMessage](#BeaconMessage).Id(k),

[CalibrationValidationStartBeacon](#CalibrationValidationStartBeacon)(k)) >= [MinDistanceRanForValidation](#MinDistanceRanForValidation))

[MinCogCalibration](#MinCogCalibration) = [CalibrationResultMin](#CalibrationResultMin)(k)

[MaxCogCalibration](#MaxCogCalibration) = [CalibrationResultMax](#CalibrationResultMax)(k)

[CalibrationState](#CalibrationState) = **CALI\_COMPLETED**

else:

[CalibrationState](#CalibrationState) = **CALI\_WAITING**

MaxDistanceRanForValidation

=([abs](#abs)([CalibrationValidationStartPositionMin](#CalibrationValidationStartPositionMin)(k)— [CogPositionAfterTopLoc](#CogPositionAfterTopLoc)(k)) + 1)

\* [CalibrationResultMax](#CalibrationResultMax)(k)

MinDistanceRanForValidation

=([abs](#abs)([CalibrationValidationStartPositionMax](#CalibrationValidationStartPositionMax)(k)— [CogPositionBeforeTopLoc](#CogPositionBeforeTopLoc)(k)) - 1)

\* [CalibrationResultMin](#CalibrationResultMin)(k)

#

@

End2OrientationByBeacon，当定位初始化时，通过经过的信标，判断**END\_2**驾驶室所面对的运营方向。

$

The orientation of the train END means the **UP** or **DOWN** orientation which this END toward to. When a pair of consecutive beacon read, ATP can determine the orientation for each train END according to the direction of these beacons in track map and the direction of train movement.

%

def [End2OrientationByBeacon](#End2OrientationByBeacon)(k):

if (Initialization

or not [MovingInitialByBeacon](#MovingInitialByBeacon)(k)):

return **DOT\_UNKNOWN**

elif ([ATPsetting](#ATPsetting).PolarizedTrain):

return [ATPsetting](#ATPsetting).End2Orientation

elif ([NewBeaconObtained](#NewBeaconObtained)(k)

and [TrackMap](#TrackMap).[AreNeighbouredBeacons](#AreNeighbouredBeacons)([BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k),

[BeaconLastObtained](#BeaconLastObtained)(k))):

if ([End2RunningForward](#End2RunningForward)(k)):

return ([TrackMap](#TrackMap).[OrientationOfNeighbouredBeacons](#OrientationOfNeighbouredBeacons)([BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k),

[BeaconLastObtained](#BeaconLastObtained)(k)))

else:

return ([TrackMap](#TrackMap).[OrientationOfNeighbouredBeacons](#OrientationOfNeighbouredBeacons)([BeaconLastObtained](#BeaconLastObtained)(k),

[BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k)))

else:

return [End2OrientationByBeacon](#End2OrientationByBeacon)(k-1)

#

@

BeaconLocation，如果本周期读到了重定位信标（无论是否已经在定位状态），则ATP需根据该信标在线路地图中的坐标计算读到信标时刻**END\_2**车头的位置：

$

If a valid beacon read, ATP shall calculate actual maximum and minimum location of the **END\_2** according to the beacon location in track map, the distance from beacon antenna to the **END\_2**, the distance after top-loc, the orientation of **END\_2** and the direction of train movement. The beacon location indicate the actual external location of the **END\_2**.

%

def [BeaconLocation](#BeaconLocation)(k):

if ([NewBeaconObtained](#NewBeaconObtained)(k)):

if ([End2OrientationByBeacon](#End2OrientationByBeacon)(k) is not **None**):

[BeaconLocation](#BeaconLocation).Ext2 = ([TrackMap.LocationUpdateExt2](#LocationUpdateExt2)

([End2RunningForward](#End2RunningForward)(k),

[End2OrientationByBeacon](#End2OrientationByBeacon)(k),

[BeaconLastObtained](#BeaconLastObtained).Location(k),

[MaxMotionOfEnd2](#MaxMotionOfEnd2)(k),

[MinMotionOfEnd2](#MinMotionOfEnd2)(k)))

elif ([TrainLocalized](#TrainLocalized)(k-1)):

[BeaconLocation](#BeaconLocation).Ext2 = ([TrackMap.LocationUpdateExt2](#LocationUpdateExt2)

([End2RunningForward](#End2RunningForward)(k),

[TrainLocation](#TrainLocation)(k-1).Ext2.Ort,

[BeaconLastObtained](#BeaconLastObtained).Location(k),

[MaxMotionOfEnd2](#MaxMotionOfEnd2)(k),

[MinMotionOfEnd2](#MinMotionOfEnd2)(k)))

else:

[BeaconLocation](#BeaconLocation).Ext2 = **None**

[BeaconLocation](#BeaconLocation).Uncertainty = (2 \* [BeaconLastObtained](#BeaconLastObtained)(k).PositionTolerance

+ [abs](#abs)([DistLastBeaconMax](#DistLastBeaconMax)(k) - [DistLastBeaconMin](#DistLastBeaconMin)(k)))

[BeaconLocation](#BeaconLocation).Int2 = [UpdateInt2FromExt2](#UpdateInt2FromExt2)

[BeaconLocation](#BeaconLocation).Ext1 = [UpdateExt1FromExt2](#UpdateExt1FromExt2)

[BeaconLocation](#BeaconLocation).Int1 = [UpdateInt1FromExt2](#UpdateInt1FromExt2)

else:

[BeaconLocation](#BeaconLocation) = **None**

return [BeaconLocation](#BeaconLocation)

def MaxMotionOfEnd2(k)

return ([DistLastBeaconMax](#DistLastBeaconMax)(k)

- [ATPsetting](#ATPsetting).CCcoreEnd2BeaconAntennaDistance[[CoreId](#CoreId)(k)]

- [BeaconLastObtained](#BeaconLastObtained)(k).PositionTolerance)

def MinMotionOfEnd2(k)

return ([DistLastBeaconMin](#DistLastBeaconMin)(k)

- [ATPsetting](#ATPsetting).CCcoreEnd2BeaconAntennaDistance[[CoreId](#CoreId)(k)]

- [BeaconLastObtained](#BeaconLastObtained)(k).PositionTolerance)

#

@

MovingInitialByBeacon，是否在信标初始化定位过程中。

%

def [MovingInitialByBeacon](#MovingInitialByBeacon)(k):

if (Initialization

or [TrainLocalized](#TrainLocalized)(k-1)

or [TrainFilteredStopped](#TrainFilteredStopped)(k)

or ([End1RunningForward](#End1RunningForward)(k) and not [End1RunningForward](#End1RunningForward)(k-1))

or ([End2RunningForward](#End2RunningForward)(k) and not [End2RunningForward](#End2RunningForward)(k-1))

or [abs](#abs)([DistLastBeaconMax](#DistLastBeaconMax)(k)) >= [ATPsetting](#ATPsetting).BeaconPairMaxDistance):

return **False**

elif ([NewBeaconObtained](#NewBeaconObtained)(k)):

return **True**

else:

return [MovingInitialByBeacon](#MovingInitialByBeacon)(k-1)

#

@

TrainLocatedOnBeacon，列车通过信标进行初始化定位（该值仅在处理信标的周期为**True**）。如果在定位初始化阶段读到信标，且能够根据该信标的位置计算出列车的定位，（即车身范围内没有轨道边界或未知状态的道岔），即认为初始化定位成功。

%

def [TrainLocatedOnBeacon](#TrainLocatedOnBeacon)(k):

return ([MovingInitialByBeacon](#MovingInitialByBeacon)(k)

and [NewBeaconObtained](#NewBeaconObtained)(k)

and ([ATPsetting](#ATPsetting).PolarizedTrain

or ([MovingInitialByBeacon](#MovingInitialByBeacon)(k-1)

and [TrackMap](#TrackMap).[AreNeighbouredBeacons](#AreNeighbouredBeacons)([BeaconBeforeLastObtained](#BeaconBeforeLastObtained)(k),

[BeaconLastObtained](#BeaconLastObtained)(k))))

and [BeaconLocation](#BeaconLocation)(k) is not **None**)

#

@

MemorizedLocationAuthorized，项目配置是否授权使用记忆定位

%

def [MemorizedLocationAuthorized](#MemorizedLocationAuthorized)(k):

return [Offline.GetMemorizedLocationAuthorized](#GetMemorizedLocationAuthorized)(k)

#

@

MemorizedLocationEnable，仅在刚上电车还未动时允许使用记忆定位

%

def [MemorizedLocationEnable](#MemorizedLocationEnable)(k):

return (not [TrainHasMoved](#TrainHasMoved)(k))

#

@

MemorizedLocationAvailable，记忆定位是否可用

%

def [MemorizedLocationAvailable](#MemorizedLocationAvailable)(k):

return ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and [ValidTrainKinematic](#ValidTrainKinematic)(k)

and [MemorizedLocationAuthorized](#MemorizedLocationAuthorized)(k)

and [MemorizedLocationEnable](#MemorizedLocationEnable)(k)

and Message.[VitalChecksumValid](#VitalChecksumValid)([MemLocation](#MemLocation))

and [MemLocation](#MemLocation).MemLocVersion == **MEM\_LOCATION\_VERSION**

and [MemLocation](#MemLocation).TrainType == [TrainType](#TrainType)(k)

and [MemLocation](#MemLocation).TrainId == [SubSystemId](#SubSystemId)(k)

and [MemLocation](#MemLocation).SleepAreaId == ([TrackMap](#TrackMap).[ExistZoneLocationIncluded](#ExistZoneLocationIncluded)

(**SGL\_SLEEPING\_ZONE**, [MemLocation](#MemLocation)(k).Ext2).Id)

and [MemLocation](#MemLocation).SleepAreaVersion == ([TrackMap](#TrackMap).[ExistZoneLocationIncluded](#ExistZoneLocationIncluded)

(**SGL\_SLEEPING\_ZONE**, [MemLocation](#MemLocation)(k).Ext2).Version)

and [MemLocation](#MemLocation).TrainLength == [ATPsetting](#ATPsetting).LocationTrainLength)

#

@

MotionSinceMemorizedLocation，记录自唤醒后运行了多少距离

%

def [MotionSinceMemorizedLocation](#MotionSinceMemorizedLocation)(k):

if ([TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k-1)):

return [OtherATP](#OtherATP)(k).MotionSinceMemLoc

elif ([MemorizedLocationAvailable](#MemorizedLocationAvailable)(k)

or not [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k-1)):

return 0

else:

return ([MotionSinceMemorizedLocation](#MotionSinceMemorizedLocation)(k-1) + [MaximumTrainMotion](#MaximumTrainMotion)(k))

#

@

TrainPresumablyLocalized，列车使用记忆定位，但还未读到确认信标的状态。待已经通过信标重定位，或者失位后，清除该值。

%

def [TrainPresumablyLocalized](#TrainPresumablyLocalized)(k):

if ([MemorizedLocationAvailable](#MemorizedLocationAvailable)(k)

and not [TrainPresumablyLocalized](#TrainPresumablyLocalized)(k-1)):

return **True**

elif ([TrainPresumablyLocalized](#TrainPresumablyLocalized)(k-1)

and (not [TrainLocalized](#TrainLocalized)(k-1)

or [TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k-1)):

return **False**

else:

return [TrainPresumablyLocalized](#TrainPresumablyLocalized)(k-1)

#

@

TrainStoppedStartTime，记录开始停车的时间

%

def [TrainStoppedStartTime](#TrainStoppedStartTime)(k):

if (Initialization

or (not [TrainFilteredStopped](#TrainFilteredStopped)(k-1)

and [TrainFilteredStopped](#TrainFilteredStopped)(k)):

return [ATPtime](#ATPtime)(k)

else:

return [TrainStoppedStartTime](#TrainStoppedStartTime)(k-1)

#

@

TrainLocatedOnOtherATP，本端和远端都在停车状态时，才有可能使用远端定位

$

Only when ATP and redundant ATP are all in filtered stopped state, can ATP use redundant ATP location for initialization.

%

def [TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k):

if ([OtherATP](#OtherATP)(k).LocatedOnKnownPath

and [TrainFilteredStopped](#TrainFilteredStopped)(k)

and [OtherATP](#OtherATP)(k).TrainFilteredStopped

and [Message.IsMoreRecent](#IsMoreRecent)([OtherATP](#OtherATP)(k).LatestTimeOtherCore, [TrainStoppedStartTime](#TrainStoppedStartTime)(k))

and not [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k-1)

and not [TrainPresumablyLocalized](#TrainPresumablyLocalized)(k)):

[TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP) = **True**

else:

[TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP) = **False**

return [TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)

#

@

TrainInitialLocation，记录列车通过远端ATP、记忆定位、或信标初始化时的位置。如果列车失位，则清除该位置；如果列车保持定位，则保留该位置。

$

ATP determine the initial train location by redundant ATP, memorized location and beacon location in order. If train delocalized, the train location should be clear.

%

def [TrainInitialLocation](#TrainInitialLocation)(k):

if ([TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k)):

return [OtherATP](#OtherATP)(k).Location

elif ([TrainPresumablyLocalized](#TrainPresumablyLocalized)(k)

and not [TrainHasMoved](#TrainHasMoved)(k)):

return [MemLocation](#MemLocation)(k)

elif ([TrainLocatedOnBeacon](#TrainLocatedOnBeacon)(k)):

return [BeaconLocation](#BeaconLocation)(k)

elif ([**TrainLocalized**](#TrainLocalized)(k-1)):

return [TrainInitialLocation](#TrainInitialLocation)(k-1)

else:

return **None**

#

@

LocationBeforeReloc，上周期列车已定位的情况下，使用里程计测得的位移来更新列车定位。

$

If train has localized on the track map, according to the orientation of **END\_2**, ATP using the maximum and minimum train motion to update the external or internal location of the **END\_2**.

%

def [LocationBeforeReloc](#LocationBeforeReloc)(k):

if ([TrainLocalized](#TrainLocalized)(k-1) and [ValidTrainKinematic](#ValidTrainKinematic)(k)):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

if ([End2RunningForward](#End2RunningForward)(k)):

[LocationBeforeReloc](#LocationBeforeReloc).Uncertainty = ([TrainLocation](#TrainLocation)(k-1).Uncertainty

- ([MaximumTrainMotion](#MaximumTrainMotion)(k) - [MinimumTrainMotion](#MinimumTrainMotion)(k)))

else:

[LocationBeforeReloc](#LocationBeforeReloc).Uncertainty = ([TrainLocation](#TrainLocation)(k-1).Uncertainty

+ ([MaximumTrainMotion](#MaximumTrainMotion)(k) - [MinimumTrainMotion](#MinimumTrainMotion)(k)))

else:

[LocationBeforeReloc](#LocationBeforeReloc).Uncertainty = ([TrainLocation](#TrainLocation)(k-1).Uncertainty

+ abs([MaximumTrainMotion](#MaximumTrainMotion)(k))

+ abs([MinimumTrainMotion](#MinimumTrainMotion)(k)))

[LocationBeforeReloc](#LocationBeforeReloc).Ext2 = ([TrackMap.LocationUpdateExt2](#LocationUpdateExt2)([End2RunningForward](#End2RunningForward)(k),

[TrainLocation](#TrainLocation)(k-1).Ext2.Ort,

[TrainLocation](#TrainLocation)(k-1).Ext2,

[MaximumTrainMotion](#MaximumTrainMotion)(k),

[MinimumTrainMotion](#MinimumTrainMotion)(k)))

[LocationBeforeReloc](#LocationBeforeReloc).Int2 = [UpdateInt2FromExt2](#UpdateInt2FromExt2)

[LocationBeforeReloc](#LocationBeforeReloc).Ext1 = [UpdateExt1FromExt2](#UpdateExt1FromExt2)

[LocationBeforeReloc](#LocationBeforeReloc).Int1 = [UpdateInt1FromExt2](#UpdateInt1FromExt2)

else:

[LocationBeforeReloc](#LocationBeforeReloc) = **None**

return [LocationBeforeReloc](#LocationBeforeReloc)

#

@

LocationUntravelable，判断车身范围内是否有线路边界或者状态不符的道岔。当上周期列车定位，并满足以下条件时，本周期设置为**True**。如果车尾最小定位到车头最大定位之间存在状态未知的道岔(包括发散或汇聚节点)；或者，如果车尾最小定位到车头最大定位之间存在变量状态与之前列车位置不符的发散汇聚节点；或者，轨道边界在列车定位范围内；否则，设置该值为**False**。

$

ATP shall determine whether there is an unknown-status point intersecting with the train location.If the train has localized at the previous cycle, and fulfills one of the following conditions: There is an unknown-status divergence of convergence located in the range from train tail to train head, then ATP shall set as **True**; Or else:, if there is a convergence with reverse route located in the range from train tail to train head, the ATP shall set as **True**;Or else, train crossed the boundary of ATC area, shall set as **True**.Otherwise, set as **False**.

%

def [LocationUntravelable](#LocationUntravelable)(k):

return ([TrainLocalized](#TrainLocalized)(k-1)

and (not ([TrackMap](#TrackMap).ReachableBetweenTwoLocations

([LocationBeforeReloc](#LocationBeforeReloc)(k).Ext2, [LocationBeforeReloc](#LocationBeforeReloc)(k).Ext1,

([abs](#abs)([MaximumTrainMotion](#MaximumTrainMotion)) + [ATPsetting](#ATPsetting).LocationTrainLength

+ [LocationBeforeReloc](#LocationBeforeReloc)(k).Uncertainty)))

or not ([TrackMap](#TrackMap).ReachableBetweenTwoLocations

([LocationBeforeReloc](#LocationBeforeReloc)(k).Ext1, [LocationBeforeReloc](#LocationBeforeReloc)(k).Ext2,

([abs](#abs)([MaximumTrainMotion](#MaximumTrainMotion)) + [ATPsetting](#ATPsetting).LocationTrainLength

+ [LocationBeforeReloc](#LocationBeforeReloc)(k).Uncertainty)))))

#

@

InverseLocation，判断**END\_2**车头的外侧和内侧定位顺序是否正确。

$

ATP shall determine the correct order of the external and internal location of train END.

%

def [InverseLocation](#InverseLocation)(k):

return ([LocationBeforeReloc](#LocationBeforeReloc)(k).Uncertainty < 0)

#

@

LocationUncertaintyExceed，列车定位状态下，每周期计算列车外侧定位和内侧之间的距离是否超过最大允许误差。

$

ATP shall calculate the uncertain distance between the external and internal locations of train **END\_2**.

%

def [LocationUncertaintyExceed](#LocationUncertaintyExceed)(k):

return ([LocationBeforeReloc](#LocationBeforeReloc)(k).Uncertainty > [ATPsetting](#ATPsetting).LocationMaxUncertaintyConfirmed)

#

@

TrainRealignmentOnBeacon，是否在信标上重定位成功

%

def [TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k):

return ([TrainLocalized](#TrainLocalized)(k-1)

and [NewBeaconObtained](#NewBeaconObtained)(k)

and ([TrackMap](#TrackMap).[IntersectionOfTwoZones](#IntersectionOfTwoZones)([BeaconLocation](#BeaconLocation).Int2, [BeaconLocation](#BeaconLocation).Ext2,

[LocationBeforeReloc](#LocationBeforeReloc).Int2, [LocationBeforeReloc](#LocationBeforeReloc).Ext2)

is not **None**))

#

@

RealignmentFailed，上周期定位状态下，如果读到新的重定位信标，但ATP根据位移计算的最大最小定位，与通过读到信标位置计算的最大最小定位之间没有交集，则认为重定位失败。

%

def [RealignmentFailed](#RealignmentFailed)(k):

return ([TrainLocalized](#TrainLocalized)(k-1)

and [NewBeaconObtained](#NewBeaconObtained)(k)

and ([TrackMap](#TrackMap).[IntersectionOfTwoZones](#IntersectionOfTwoZones)([BeaconLocation](#BeaconLocation).Int2, [BeaconLocation](#BeaconLocation).Ext2,

[LocationBeforeReloc](#LocationBeforeReloc).Int2, [LocationBeforeReloc](#LocationBeforeReloc).Ext2)

is **None**))

#

@

LocationAfterReloc，经过信标重定位后的列车定位

%

def [LocationAfterReloc](#LocationAfterReloc)(k):

if ([TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k)):

([LocationAfterReloc](#LocationAfterReloc).Int2,

[LocationAfterReloc](#LocationAfterReloc).Ext2) = ([TrackMap](#TrackMap).[IntersectionOfTwoZones](#IntersectionOfTwoZones)([LocationBeforeReloc](#LocationBeforeReloc)(k).Int2,

[LocationBeforeReloc](#LocationBeforeReloc)(k).Ext2,

[BeaconLocation](#BeaconLocation)(k).Int2,

[BeaconLocation](#BeaconLocation)(k).Ext2))

[LocationAfterReloc](#LocationAfterReloc).Uncertainty = ([TrackMap](#TrackMap).[DistanceBtwTwoLocs](#DistanceBtwTwoLocs)([LocationAfterReloc](#LocationAfterReloc).Int2,

[LocationAfterReloc](#LocationAfterReloc).Ext2,

[ATPsetting](#ATPsetting).LocationMaxUncertaintyConfirmed))

[LocationAfterReloc](#LocationAfterReloc).Int1 = [UpdateInt1FromExt2](#UpdateInt1FromExt2)

[LocationAfterReloc](#LocationAfterReloc).Ext1 = [UpdateExt1FromExt2](#UpdateExt1FromExt2)

elif (not [TrainLocalized](#TrainLocalized)(k-1)):

[LocationAfterReloc](#LocationAfterReloc) = **None**

else:

[LocationAfterReloc](#LocationAfterReloc) = [LocationAfterReloc](#LocationAfterReloc)(k-1)

return [LocationAfterReloc](#LocationAfterReloc)

#

@

LocPermanentFailure，在列车已定位，且未使用非确认的BM变量情况下，若发生重定位失败，则永久失位。

%

def [LocPermanentFailure](#LocPermanentFailure)(k):

return ([LocPermanentFailure](#LocPermanentFailure)(k-1)

or ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k-1)

and [RealignmentFailed](#RealignmentFailed)(k)))

#

@

MotionSinceLastReloc，记录自上次信标重定位后的运行距离绝对值

%

def [MotionSinceLastReloc](#MotionSinceLastReloc)(k):

if ([TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k)):

return [OtherATP](#OtherATP)(k).[MotionSinceLastReloc](#MotionSinceLastReloc)

elif ([TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k)):

return [abs](#abs)([DistLastBeaconMax](#DistLastBeaconMax)(k))

elif (not [TrainLocalized](#TrainLocalized)(k-1)):

return 0

else:

return ([MotionSinceLastReloc](#MotionSinceLastReloc)(k-1) + [abs](#abs)([MaximumTrainMotion](#MaximumTrainMotion)(k)))

#

@

CBTCwithoutKnownPath，在CBTC下，若上周期定位path为假，则失位

%

def [CBTCwithoutKnownPath](#CBTCwithoutKnownPath)(k):

return ([TrainLocalized](#TrainLocalized)(k-1)

and not [LocationPathKnown](#LocationPathKnown)(k-1)

and not [BlockModeUsed](#BlockModeUsed)(k))

#

@

TrainUnitIntegrity，任一端车头能保证列车完整性，则认为车辆完整性能被保证。如果该项目未配置列车完整性采集，则认为列车完整性已由车辆保证。其状态来自于项目可配置的列车输入采集。

$

If either of ends can ensure the train integrity, ATP shall set [TrainUnitIntegrity](#TrainUnitIntegrity) as **True**. If the project is not configured with the capture of train integrity, it is sure that the train can guarantees the integrity.

%

def [TrainUnitIntegrity](#TrainUnitIntegrity)(k):

return [Offline](#Offline).[GetTrainUnitIntegrity](#GetTrainUnitIntegrity)(k)

#

@

LocalizationFault用于表明是否发生定位错误。判断条件如下:

$

If the localization state is not **NOT\_LOCALIZED**, ATP shall determine whether the localization fault happens or not, according to the following pseudo-codes:

%

def [LocalizationFault](#LocalizationFault)(k):

return (not [TrainUnitIntegrity](#TrainUnitIntegrity)(k)

or not [ValidTrainKinematic](#ValidTrainKinematic)(k)

or [LocationUntravelable](#LocationUntravelable)(k)

or [InverseLocation](#InverseLocation)(k)

or [LocationUncertaintyExceed](#LocationUncertaintyExceed)(k)

or [RealignmentFailed](#RealignmentFailed)(k)

or [LocPermanentFailure](#LocPermanentFailure)(k)

or ([MotionSinceLastReloc](#MotionSinceLastReloc)(k) > [ATPsetting](#ATPsetting).LocationBeaconValidityDistance)

or ([MemLocationNotConfirmed](#MemLocationNotConfirmed)(k-1)

and [abs](#abs)([MotionSinceMemorizedLocation](#MotionSinceMemorizedLocation)(k)) >= [ATPsetting](#ATPsetting).MemLocValidityDistance)

or [CBTCwithoutKnownPath](#CBTCwithoutKnownPath)(k)

or [CoupledTypeInconsistent](#CoupledTypeInconsistent)(k))

#

@

TrainLocalized，表示当前列车是否定位。当列车定位初始化后，ATP可根据里程计测得并经打滑补偿和轴锁判断处理的列车位移，每周期更新列车在线路地图中的位置。如果再读到信标，则ATP可根据该信标的位置对之前的定位进行重新校正。考虑到安全，ATP需维护列车每端车头的外侧和内侧两组定位信息。

$

Only the localization state is **LOCALIZED**, ATP shall consider the train has localized. When the train passed the continuous two beacons, ATP can judge the initial location and direction according to the position and the sequences of above-mentioned beacons in track map. Later, ATP can update the train location in the track map in each cycle based on the train movement combined with sliding overestimation and wheel block consideration. If ATP received a new beacon, it will realign the train location according to this beacon. For safety, ATP needs to maintain the location information from the external and internal side of each train

end.

%

def [TrainLocalized](#TrainLocalized)(k):

if (Initialization

or [LocalizationFault](#LocalizationFault)(k)):

return **False**

elif (not [TrainLocalized](#TrainLocalized)(k-1)

and ([TrainPresumablyLocalized](#TrainPresumablyLocalized)(k)

or [TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k)

or [TrainLocatedOnBeacon](#TrainLocatedOnBeacon)(k))):

return **True**

else:

return [TrainLocalized](#TrainLocalized)(k-1)

#

@

TrainLocation，列车End1和End2端定位。分为以下四种情况：本周期非定位；本周期刚初始化；本周期经过信标重定位；本周期使用位移累加定位。

%

def [TrainLocation](#TrainLocation)(k):

if (not [TrainLocalized](#TrainLocalized)(k)):

return **None**

elif (not [TrainLocalized](#TrainLocalized)(k-1)):

return [TrainInitialLocation](#TrainInitialLocation)(k)

elif ([TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k)):

return [LocationAfterReloc](#LocationAfterReloc)(k)

else:

return [LocationBeforeReloc](#LocationBeforeReloc)(k)

#

@

DriverInCab\_1或DriverInCab\_2，如果采集到某端的驾驶室被激活，则ATP认为司机在该端驾驶室。其状态来自于项目可配置的列车输入采集。

$

ATP shall consider the driver is in this cab if it captures that either end of cab activated, which shown by the data from [DriverInCab\_1](#DriverInCab_1) or [DriverInCab\_2](#DriverInCab_2).

%

def [DriverInCab\_1](#DriverInCab_1)(k):

return [Offline](#Offline).[GetDriverInCab\_1](#GetDriverInCab_1)(k)

def [DriverInCab\_2](#DriverInCab_2)(k):

return [Offline](#Offline).[GetDriverInCab\_2](#GetDriverInCab_2)(k)

#

@

DriverInTrain，当前是否有司机在车内

$

If the active status is different between two ENDs of the train, ATP consider there is a driver in train.

%

def [DriverInTrain](#DriverInTrain)(k):

return ([DriverInCab\_1](#DriverInCab_1)(k) is not [DriverInCab\_2](#DriverInCab_2)(k))

#

@

NonVitalSelectedFrontEnd，来自CCNV的车头选择信息

￥

[NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd) represents the train front choice from CCNV.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd) = [NonVitalRequest](#NonVitalRequest).SelectedFrontEnd(k)

else:

[NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd) = **UNKNOW**

#

@

TrainFrontEnd，判断司机在**END\_1**还是**END\_2**还是由CCNV选择。

$

If there is a driver in the train, the train front is the activated END. or else: the front determined by CCNV.Otherwise, the train front is the default one or the front one when train is moving.

%

def [TrainFrontEnd](#TrainFrontEnd)(k):

if (Initialization):

return **END\_2**

elif ([DriverInTrain](#DriverInTrain)(k)):

if ([DriverInCab\_1](#DriverInCab_1)(k)):

return **END\_1**

else:

return **END\_2**

elif ([NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd)(k) is **END\_1**

or [NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd)(k) is **END\_2**):

return [NonVitalSelectedFrontEnd](#NonVitalSelectedFrontEnd)(k)

elif ([WheelFilteredStopped](#WheelFilteredStopped)(k)):

return [TrainFrontEnd](#TrainFrontEnd)(k-1)

elif (not [End2RunningForward](#End2RunningForward)(k)):

return **END\_1**

else:

return **END\_2**

#

@

TrainFrontOrientation，列车运营方向.

$

The train front orientation is the orientation of the active train END.

%

def [TrainFrontOrientation](#TrainFrontOrientation)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**):

return [TrainLocation](#TrainLocation).Ext2.Ort(k)

else:

return [TrainLocation](#TrainLocation).Ext1.Ort(k)

#

@

TrainFrontLocation，车头定位的更新:

$

ATP updates the train front location according to the active train END.

%

def [TrainFrontLocation](#TrainFrontLocation)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

[TrainFrontLocation](#TrainFrontLocation).Max = [TrainLocation](#TrainLocation).Ext1

[TrainFrontLocation](#TrainFrontLocation).Min = [TrainLocation](#TrainLocation).Int1

else:

[TrainFrontLocation](#TrainFrontLocation).Max = [TrainLocation](#TrainLocation).Ext2

[TrainFrontLocation](#TrainFrontLocation).Min = [TrainLocation](#TrainLocation).Int2

return [TrainFrontLocation](#TrainFrontLocation)

#

@

TrainRearLocation，车尾定位的更新:

$

ATP updates the train rear locations according to the active train END.

%

def [TrainRearLocation](#TrainRearLocation)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

[TrainRearLocation](#TrainRearLocation).Max = [TrainLocation](#TrainLocation).Int2

[TrainRearLocation](#TrainRearLocation).Min = [TrainLocation](#TrainLocation).Ext2

else:

[TrainRearLocation](#TrainRearLocation).Max = [TrainLocation](#TrainLocation).Int1

[TrainRearLocation](#TrainRearLocation).Min = [TrainLocation](#TrainLocation).Ext1

return [TrainRearLocation](#TrainRearLocation)

#

@

LocationPathKnown，判断列车在Block模式没有移动授权条件下是否经过道岔导致非确认定位。

$

In BM mode, if train localized but EOA is invalid, [LocationPathKnown](#LocationPathKnown) cannot be **TRUE** when train cross switch. After train crossed switch and relocalized by beacon successfully, [LocationPathKnown](#LocationPathKnown) can be set to **TRUE**.

%

def [LocationPathKnown](#LocationPathKnown)(k):

if (Initialization

or ([BlockModeUsed](#BlockModeUsed)(k)

and [TrainLocalized](#TrainLocalized)(k)

and not [BlockModeEOAvalid](#BlockModeEOAvalid)(k-1)

and not [TrainFilteredStopped](#TrainFilteredStopped)(k)

and [TrackMap.ExistSwitchBtwTwoLocs](#ExistSwitchBtwTwoLocs)([TrainRearLocation](#TrainRearLocation)(k).Min,

[TrainFrontLocation](#TrainFrontLocation)(k).Max))):

return **False**

elif ([TrainLocatedOnBeacon](#TrainLocatedOnBeacon)(k)

or [TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k)

or [TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k)

or ([TrainPresumablyLocalized](#TrainPresumablyLocalized)(k) and not [TrainHasMoved](#TrainHasMoved)(k))):

return **True**

else:

return [LocationPathKnown](#LocationPathKnown)(k-1)

#

@

TrainLocatedOnKnownPath，判断列车是否定位并已知[LocationPathKnown](#LocationPathKnown)。

%

def [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k):

return ([TrainLocalized](#TrainLocalized)(k)

and [LocationPathKnown](#LocationPathKnown)(k))

#

@

LocalizationState，列车的定位状态，用于用于维护诊断功能。

%

def [LocalizationState](#LocalizationState)(k):

if ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and not [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k)):

return **LOCALIZED\_CONFIRMED**

elif ([TrainLocalized](#TrainLocalized)(k)):

return **LOCALIZED\_NOT\_CONFIRMED**

elif ([MovingInitialByBeacon](#MovingInitialByBeacon)(k)):

return **MOVING\_INIT**

else:

return **NOT\_LOCALIZED**

#

@

MemLocationNotConfirmed，是否通过本端或远端的记忆定位初始化列车定位，但还未通过重定位信标确认定位。

%

def [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k):

if (Initialization

or not [TrainLocalized](#TrainLocalized)(k)

or [TrainRealignmentOnBeacon](#TrainRealignmentOnBeacon)(k)):

return **False**

elif ([TrainPresumablyLocalized](#TrainPresumablyLocalized)(k)):

return **True**

elif ([TrainLocatedOnOtherATP](#TrainLocatedOnOtherATP)(k)):

return [OtherATP](#OtherATP).LocatedWithMemLocation(k)

else:

return [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k-1)

#

@

LocationUncertaintyExceedTime，记录超过最大定位误差的时间

%

def [LocationUncertaintyExceedTime](#LocationUncertaintyExceedTime)(k):

if (Initialization):

return 0

elif ([TrainLocalized](#TrainLocalized)(k)

and (([MemLocationNotConfirmed](#MemLocationNotConfirmed)(k) or not [LocationPathKnown](#LocationPathKnown)(k))

and ([TrainLocation](#TrainLocation)(k).Uncertainty

> [ATPsetting](#ATPsetting).LocationMaxUncertaintyNotConfirmed))):

return ([ATPtime](#ATPtime)(k) + [ATPsetting](#ATPsetting).LocReportValidityTime)

else:

return [LocationUncertaintyExceedTime](#LocationUncertaintyExceedTime)(k-1)

#

@

LocationNotUncertaintyExceed，判断是否还处在最大定位误差的确认时间内

%

def [LocationNotUncertaintyExceed](#LocationNotUncertaintyExceed)(k):

if (Initialization):

return **False**

elif ([TrainLocalized](#TrainLocalized)(k)):

return [Message.IsMoreRecent](#IsMoreRecent)([ATPtime](#ATPtime)(k), [LocationUncertaintyExceedTime](#LocationUncertaintyExceedTime)(k))

else:

return [LocationNotUncertaintyExceed](#LocationNotUncertaintyExceed)(k-1)

#

@

LocalizedAuthorizationForSweepping，发给ZC的是否定位信息。

$

ATP shall send the current localization status to the [ZC](#ZC).

%

def [LocalizedAuthorizationForSweepping](#LocalizedAuthorizationForSweepping)(k):

return ([TrainLocalized](#TrainLocalized)(k)

and [LocationNotUncertaintyExceed](#LocationNotUncertaintyExceed)(k))

#

@

TrainConfirmedLocalized，发给ZC的是否确认定位信息。

$

­­ATP shall send the status of the localization status whether confirmed.

%

def [TrainConfirmedLocalized](#TrainConfirmedLocalized)(k):

return ([LocationPathKnown](#LocationPathKnown)(k)

and not [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k))

#

@

TrainIncludedInSleepingZone，列车停车后定位完全所在的Sleeping zone

%

def [TrainIncludedInSleepingZone](#TrainIncludedInSleepingZone)(k):

if (not [TrainFilteredStopped](#TrainFilteredStopped)(k)):

return **None**

else:

for SleepZone in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_SLEEPING\_ZONE**,

[TrackMap.BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainFrontLocation](#TrainFrontLocation)(k).Max)):

if ([TrackMap](#TrackMap).[LocationInZone](#LocationInZone)([TrainFrontLocation](#TrainFrontLocation)(k).Max,

SleepZone.Location,

SleepZone.Length,

SleepZone.Orientation)

and [TrackMap](#TrackMap).[LocationInZone](#LocationInZone)([TrainRearLocation](#TrainRearLocation)(k).Min,

SleepZone.Location,

SleepZone.Length,

SleepZone.Orientation)):

return SleepZone

else:

continue:

return **None**

#

@

WritingMemLocRequest，是否写入记忆定位。

$

Only when train has moved and filtered stopped in sleeping zone, can ATP writing memorized location information.

%

def [WritingMemLocRequest](#WritingMemLocRequest)(k):

return ([TrainHasMoved](#TrainHasMoved)(k)

and [TrainIncludedInSleepingZone](#TrainIncludedInSleepingZone)(k) is not **None**

and [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k))

#

@

ClearingMemLocRequest，是否清除记忆定位

$

When train has moved and does not fulfill the condition of writing memory location, ATP shall clear memorized location information.

%

def [ClearingMemLocRequest](#ClearingMemLocRequest)(k):

return (not [WritingMemLocRequest](#WritingMemLocRequest)(k) and [TrainHasMoved](#TrainHasMoved)(k))

#

@

MemLocWritten，写入记忆定位的内容

%

def [MemLocWritten](#MemLocWritten)(k):

if ([WritingMemLocRequest](#WritingMemLocRequest)(k)):

[MemLocWritten](#MemLocWritten).MemLocVersion = **MEM\_LOCATION\_VERSION**

[MemLocWritten](#MemLocWritten).SleepAreaId = [TrainIncludedInSleepingZone](#TrainIncludedInSleepingZone)(k).Id

[MemLocWritten](#MemLocWritten).SleepAreaVersion = [TrainIncludedInSleepingZone](#TrainIncludedInSleepingZone)(k).Version

[MemLocWritten](#MemLocWritten).TrainType = [TrainType](#TrainType)(k)

[MemLocWritten](#MemLocWritten).TrainId = [SubSystemId](#SubSystemId)(k)

[MemLocWritten](#MemLocWritten).Ext2 = [TrainLocation](#TrainLocation).Ext2

[MemLocWritten](#MemLocWritten).Ext1 = [TrainLocation](#TrainLocation).Ext1

[MemLocWritten](#MemLocWritten).Uncertainty = [TrainLocation](#TrainLocation).Uncertainty

[MemLocWritten](#MemLocWritten).TrainLength = [ATPsetting](#ATPsetting).LocationTrainLength

elif ([ClearingMemLocRequest](#ClearingMemLocRequest)(k)):

[MemLocWritten](#MemLocWritten).MemLocVersion = **None**

[MemLocWritten](#MemLocWritten).SleepAreaId = **None**

[MemLocWritten](#MemLocWritten).SleepAreaVersion = **None**

[MemLocWritten](#MemLocWritten).TrainType = **None**

[MemLocWritten](#MemLocWritten).TrainId = **None**

[MemLocWritten](#MemLocWritten).Ext2 = **None**

[MemLocWritten](#MemLocWritten).Ext1 = **None**

[MemLocWritten](#MemLocWritten).Uncertainty = **None**

[MemLocWritten](#MemLocWritten).TrainLength = None

else:

[MemLocWritten](#MemLocWritten) = [MemLocWritten](#MemLocWritten)(k-1)

return [MemLocWritten](#MemLocWritten)

#

@

RestrictiveSignalOverrun，BM模式下，本周期列车车头最大定位是否冒进限制状态的信号机。当满足以下所有条件时，ATP认为列车冒进了限制状态的信号机，需设置RestrictiveSignalOverrun为**True**。本周期列车已定位，即[TrainLocalized](#TrainLocalized)为**True**；本周期使用BM变量；上周期RestrictiveSignalOverrun为**False**；本周期列车位移[MaximumTrainMotion](#MaximumTrainMotion)向激活的驾驶室方向运行；本周期列车车头最大定位[TrainFrontLocation](#TrainFrontLocation)经过了一个信号机奇点；该信号机为限制状态，或者建立了Overlap的状态。否则，设置RestrictiveSignalOverrun为**False**。

$

[RestrictiveSignalOverrun](#RestrictiveSignalOverrun), ATP shall determine whether the location of maximum train head overruns a restricted signal in BLOCK mode.When all of the following conditions fulfilled, ATP considers the train has overrun a restricted signal in this cycle, and set [RestrictiveSignalOverrun](#RestrictiveSignalOverrun) as **True**.Train has localized;And the current type of EOA is **BLOCK\_MODE\_EOA**; And [RestrictiveSignalOverrun](#RestrictiveSignalOverrun) was **False** at the last cycle;And the moving direction in current cycle is toward on the train front end; And the maximum location of train front end passes the position of the signal in this cycle;And the status of the signal is restriction or overlap established.Otherwise, ATP set [RestrictiveSignalOverrun](#RestrictiveSignalOverrun) as **False.**

%

def [RestrictiveSignalOverrun](#RestrictiveSignalOverrun)(k):

sing = [TrackMap](#TrackMap).[ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(**SGL\_SIGNAL**, [TrainFrontLocation](#TrainFrontLocation)(k-1).Max,

[TrainFrontLocation](#TrainFrontLocation)(k).Max)

return (sing is not **None**

and [BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)

and (([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2** and [End2RunningForward](#End2RunningForward)(k))

or ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1** and [End1RunningForward](#End1RunningForward)(k)))

and not [BMvariantValue](#BMvariantValue)(sing.Variant, k))

#

@

TrainInBMinitialZone，车头最小定位在在BM初始化区域内。

%

def [TrainInBMinitialZone](#TrainInBMinitialZone)(k):

NewBlock = [TrackMap.ExistSingularityInZone](#ExistSingularityInZone)(**SGL\_NEW\_BLOCK**, TrainFrontLocation(k).Min,

[ATPsetting](#ATPsetting).BMinitAreaLength)

Signal = ([TrackMap.ExistSingularityInReverseZone](#ExistSingularityInReverseZone)(**SGL\_SIGNAL**,

NewBlock.Location,

[ATPsetting](#ATPsetting).BMinitAreaLength))

if (Signal is not **None**

and Signal.BmInitialization):

return Signal

else:

return **None**

#

@

TrainEnteredInBMinitialZoneAge， 如果列车在BM初始化区域内，则记录已在该区域内运行的时间

%

def [TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge)(k):

if ([TrainInBMinitialZone](#TrainInBMinitialZone)(k) is **None**):

[TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge) = 0

else:

[TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge) = [TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge)(k-1) + 1

return [TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge)

#

@

StopAssuredPointCrossed，本周期是否通过了信号机前方的BMCP点

%

def [StopAssuredPointCrossed](#StopAssuredPointCrossed)(Cbi, k):

Bmcp = [TrackMap.ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(**SGL\_BMCP**, [TrainFrontLocation](#TrainFrontLocation)(k-1).Max,

[TrainFrontLocation](#TrainFrontLocation)(k).Max)

return (Bmcp is not **None**

and cbi == Bmcp.CbiId)

#

@

CBIminProductionAgeSinceSSAcrossing，记录从通过上个BMCP点开始到现在已经过了多长时间

%

def [CBIminProductionAgeSinceSSAcrossing](#CBIminProductionAgeSinceSSAcrossing)(Cbi, k):

if (Initialization

or [CBIminProductionAgeSinceSSAcrossing](#CBIminProductionAgeSinceSSAcrossing)(k-1)>= **REPORT\_AGE\_MAX**):

return **REPORT\_AGE\_MAX**

elif ([StopAssuredPointCrossed](#StopAssuredPointCrossed)(Cbi, k)):

return [ATPsetting](#ATPsetting).VariantsBMALSpresenceTimer

else:

return ([CBIminProductionAgeSinceSSAcrossing](#CBIminProductionAgeSinceSSAcrossing)(Cbi, k-1) + 1)

#

@

CBIminProductionAge，对于每个联锁，ATP维护最后收到其变量消息时联锁的最小时间，到现在经过的时间。

%

def [CBIminProductionAge](#CBIminProductionAge)(cbi, k):

return [min](#min)([CBIminProductionAgeSinceSSAcrossing](#CBIminProductionAgeSinceSSAcrossing)(Cbi, k),

[CBIvariantReportLastAge](#CBIvariantReportLastAge)(Cbi, k))

#

@

ReceivedVariantsAfterEnteredBMinitialZone，先进入BM初始化区，再收到无线或者信标的变量

%

def [ReceivedVariantsAfterEnteredBMinitialZone](#ReceivedVariantsAfterEnteredBMinitialZon)(k):

CbiId = [TrackMap.CbiId](#CbiId)([TrainInBMinitialZone](#TrainInBMinitialZone)(k).Block)

return ([TrainInBMinitialZone](#TrainInBMinitialZone)(k) is not **None**

and (([CBIvariantMoreAvailableThanBeacon](#CBIvariantMoreAvailableThanBeacon)(CbiId, k)

and (([CBIvariantReportLastAge](#CBIvariantReportLastAge)(CbiId, k)

+ [ATPsetting](#ATPsetting).VariantsBMproductionLatencyRadio)

< [TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge)(k)))

or ([BMbeaconReadAge](#BMbeaconReadAge)(k) + [ATPsetting](#ATPsetting).VariantsBMproductionLatencyBeacon

< [TrainEnteredInBMinitialZoneAge](#TrainEnteredInBMinitialZoneAge)(k))))

#

@

BlockModeEOAvalid，BM下的移动授权是否可用

%

def [BlockModeEOAvalid](#BlockModeEOAvalid)(k):

if (Initialization

or not [BlockModeUsed](#BlockModeUsed)(k)

or [TrainFrontEnd](#TrainFrontEnd)(k) is not [TrainFrontEnd](#TrainFrontEnd)(k-1)

or not [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

or [HazardousMotionOnNonExclusiveRoute](#HazardousMotionOnNonExclusiveRoute)(k)

or [RestrictiveSignalOverrun](#RestrictiveSignalOverrun)(k)):

return **False**

elif (not [BlockModeEOAvalid](#BlockModeEOAvalid)(k-1)

and [TrainInBMinitialZone](#TrainInBMinitialZone)(k) is not **None**

and [BMvariantValue](#BMvariantValue)([TrainInBMinitialZone](#TrainInBMinitialZone).Variant(k), k)

and [ReceivedVariantsAfterEnteredBMinitialZone](#ReceivedVariantsAfterEnteredBMinitialZon)(k)):

return **True**

else:

return [BlockModeEOAvalid](#BlockModeEOAvalid)(k-1)

#

@

TrainInSMIzone，判断当车头最大定位在SMI区域内，且车速小于SMI限速时，可使用ZC的EOA消息中的WithoutSpacingEoa进行监控。

%

def [TrainInSMIzone](#TrainInSMIzone)(k):

Smi = [TrackMap](#TrackMap).[ExistZoneLocationIncluded](#ExistZoneLocationIncluded)(**SGL\_SMI\_ZONE**, [TrainFrontEnd](#TrainFrontEnd)(k).Max)

return (Smi is not **None**

and [TrainMaxSpeed](#TrainMaxSpeed)(k) < Smi.SpeedLimit(k))

#

@

CBTCmodeEOAvalid，CBTC模式下判断来自ZC的EOA是否有效。如果在SMI区域内且车速小于SMI限速，则应当使用WithoutSpacingEOA；否则，应当使用普通的EOA

%

def [CBTCmodeEOAvalid](#CBTCmodeEOAvalid)(k):

return (not [BlockModeUsed](#BlockModeUsed)(k)

and [**ReceivedEOAreport**](#ReceivedEOAreport).TrainFrontEnd == [TrainFrontEnd](#TrainFrontEnd)(k)

and (([TrainInSMIzone](#TrainInSMIzone)(k)

and ([Message.IsMoreRecent](#IsMoreRecent)([ReceivedEOAreport](#ReceivedEOAreport)(k).WithoutSpacing.ValidityTime,

[ATPtime](#ATPtime)(k)))

and ([ReceivedEOAreport](#ReceivedEOAreport)(k).WithoutSpacing.Location.Block != 0)

and ([**TrackMap**](#TrackMap).[DistanceBtwTwoLocs](#DistanceBtwTwoLocs)([**TrainFrontLocation**](#TrainFrontLocation)(k).Min,

[ReceivedEOAreport](#ReceivedEOAreport)(k).WithoutSpacing.Location,

[ATPsetting](#ATPsetting).EOAmaxDistance) is not **None**))

or (not [TrainInSMIzone](#TrainInSMIzone)(k)

and [TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and ([Message.IsMoreRecent](#IsMoreRecent)([ReceivedEOAreport](#ReceivedEOAreport)(k).Classic.ValidityTime,

[ATPtime](#ATPtime)(k)))

and ([ReceivedEOAreport](#ReceivedEOAreport)(k).Classic.Location.BlockId != 0)

and ([**TrackMap**](#TrackMap).[DistanceBtwTwoLocs](#DistanceBtwTwoLocs)([**TrainFrontLocation**](#TrainFrontLocation)(k).Min,

[ReceivedEOAreport](#ReceivedEOAreport)(k).Eoa.Location,

[ATPsetting](#ATPsetting).EOAmaxDistance) is not **None**))))

#

@

CBTCmodeEOAlocation，CBTC下的EOA位置。

%

def [CBTCmodeEOAlocation](#CBTCmodeEOAlocation)(k):

if ([CBTCmodeEOAvalid](#CBTCmodeEOAvalid)(k)):

if ([TrainInSMIzone](#TrainInSMIzone)(k)):

return [ReceivedEOAreport](#ReceivedEOAreport).WithoutSpacing.Location

else:

return [ReceivedEOAreport](#ReceivedEOAreport).Classic.Location

else:

return **None**

#

@

EndOfAuthorityValid，统一BM或CBTC下的EOA是否可用。

%

def [EndOfAuthorityValid](#EndOfAuthorityValid)(k):

if ([BlockModeUsed](#BlockModeUsed)(k)):

return [BlockModeEOAvalid](#BlockModeEOAvalid)(k)

else:

return [CBTCmodeEOAvalid](#CBTCmodeEOAvalid)(k)

#

@

TractionAuthorisedSenseEnd1，如果EOA有效且在**END\_1**方向，则ATP授权列车向**END\_1**方向运行。

$

If current EOA is valid and whose orientation is **END\_1**, ATP shall authorize the train can move toward **END\_1**.

%

def [TractionAuthorisedSenseEnd1](#TractionAuthorisedSenseEnd1)(k):

if ([EndOfAuthorityValid](#EndOfAuthorityValid)(k)

and [TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

[TractionAuthorisedSenseEnd1](#TractionAuthorisedSenseEnd1) = **True**

else:

[TractionAuthorisedSenseEnd1](#TractionAuthorisedSenseEnd1) = **False**

return [TractionAuthorisedSenseEnd1](#TractionAuthorisedSenseEnd1)

#

@

TractionAuthorisedSenseEnd2，如果EOA有效且在**END\_2**方向，则ATP授权向驾驶室2方向运行。

$

If current EOA is valid and whose orientation is **END\_2**, ATP shall authorize the train can move toward **END\_2**.

%

def [TractionAuthorisedSenseEnd2](#TractionAuthorisedSenseEnd2)(k):

if ([EndOfAuthorityValid](#EndOfAuthorityValid)(k)

and [TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**):

[TractionAuthorisedSenseEnd2](#TractionAuthorisedSenseEnd2) = **True**

else:

[TractionAuthorisedSenseEnd2](#TractionAuthorisedSenseEnd2) = **False**

return [TractionAuthorisedSenseEnd2](#TractionAuthorisedSenseEnd2)

#

@

BMoverlapReleasableSendable，在BM下且未被ATC切除状态下，通过无线发给CI解锁信息。

%

def [BMoverlapReleasableSendable](#BMoverlapReleasableSendable)(k):

return [Offline.GetBMoverlapReleasableSendable](#GetBMoverlapReleasableSendable)(k)

#

@

OverlapReleasable，可发送Overlap解锁信息

%

def [OverlapReleasable](#OverlapReleasable)(k):

return ([BMoverlapReleasableSendable](#BMoverlapReleasableSendable)(k)

and [TrainFilteredStopped](#TrainFilteredStopped)(k)

and [BlockModeEOAvalid](#BlockModeEOAvalid)(k)

and [NonVitalRequest](#NonVitalRequest).OverlapRelease(k))

#

@

CrossedOverlapTimerInitialSignal，即本周期通过一个overlap timer初始化信号机时，返回该信号机奇点

%

def [CrossedOverlapTimerInitialSignal](#CrossedOverlapTimerInitialSignal)(k):

Signal = [TrackMap.ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(**SGL\_SIGNAL**, [TrainFrontLocation](#TrainFrontLocation)(k-1).Max,

[TrainFrontLocation](#TrainFrontLocation)(k).Max)

if (Signal is not **None**

and Signal.BmOverlapTimerInit):

return Signal

else:

return **None**

#

@

OverlapTimer，当经过具有Overlap初始化属性的信号机时，将OverlapTimer设置为当时信号机的变量有效期。

%

def [OverlapTimer](#OverlapTimer)(k):

if (not [BlockModeEOAvalid](#BlockModeEOAvalid)(k)

or [OverlapReleasable](#OverlapReleasable)(k)):

return 0

elif ([BMvariantValidWhileTemporallyValid](#BMvariantValidWhileTemporallyValid)(k)

and [CrossedOverlapTimerInitialSignal](#CrossedOverlapTimerInitialSignal)(k) is not **None**):

return [[BMvariantRemainingTime](#BMvariantRemainingTime)](#BMvariantRemaingTime)([CrossedOverlapTimerInitialSignal](#CrossedOverlapTimerInitialSignal)(k).CBIvariant.Id, k)

else:

return [OverlapTimer](#OverlapTimer)(k-1) - 1

#

@

OverlapTimerPermissive，用于判断是否在BM下强制Overlap状态建立.

%

def [OverlapTimerPermissive](#OverlapTimerPermissive)(k):

return ([OverlapTimer](#OverlapTimer)(k) > 0)

#

@

NotCoercedRestrictive\_1，非强制限制1

%

def NotCoercedRestrictive\_1(k):

return [Offline](#Offline).[GetNotCoercedRestrictive\_1](#GetNotCoercedRestrictive)(k)

#

@

NotCoercedRestrictive\_2，非强制限制2

%

def NotCoercedRestrictive\_2(k):

return [Offline](#Offline).[GetNotCoercedRestrictive\_2](#GetNotCoercedRestrictive)(k)

#

@

NotCoercedRestrictive\_3，非强制限制3

%

def NotCoercedRestrictive\_3(k):

return [Offline](#Offline).[GetNotCoercedRestrictive\_3](#GetNotCoercedRestrictive)(k)

#

@

NotCoercedRestrictive\_4，非强制限制4

%

def NotCoercedRestrictive\_4(k):

return [Offline](#Offline).[GetNotCoercedRestrictive\_4](#GetNotCoercedRestrictive)(k)

#

@

CoercedRestrictive，等于相应的“非强制限制”取反。

%

def [CoercedRestrictive](#CoercedRestrictive)(ncr, k):

if (ncr is **NOT\_COERCED\_RESTRICTIVE\_1**):

[CoercedRestrictive](#CoercedRestrictive) = not [NotCoercedRestrictive\_1](#NotCoercedRestrictive_1)(k)

elif (ncr is **NOT\_COERCED\_RESTRICTIVE\_2**):

[CoercedRestrictive](#CoercedRestrictive) = not [NotCoercedRestrictive\_2](#NotCoercedRestrictive_2)(k)

elif (ncr is **NOT\_COERCED\_RESTRICTIVE\_3**):

[CoercedRestrictive](#CoercedRestrictive) = not [NotCoercedRestrictive\_3](#NotCoercedRestrictive_3)(k)

elif (ncr is **NOT\_COERCED\_RESTRICTIVE\_4**):

[CoercedRestrictive](#CoercedRestrictive) = not [NotCoercedRestrictive\_4](#NotCoercedRestrictive_4)(k)

elif (ncr is **VARIANTS\_RECEIVED\_FROM\_CBI\_ID**):

[CoercedRestrictive](#CoercedRestrictive) = not [CBIvariantLowValidity](#CBIvariantLowValidity)(**VARIANTS\_RECEIVED\_FROM\_CBI\_ID**, k)

else:

[CoercedRestrictive](#CoercedRestrictive) = **False**

return [CoercedRestrictive](#CoercedRestrictive)

#

@

CoercedPermissive\_1，强制允许输入1

%

def [CoercedPermissive\_1](#CoercedPermissive_1) (k):

[CoercedPermissive\_1](#CoercedPermissive_1) = [Offline](#Offline).[GetCoercedPermissive\_1](#GetCoercedPermissive)(k)

return [CoercedPermissive\_1](#CoercedPermissive_1)

#

@

CoercedPermissive\_2，强制允许输入2

%

def [CoercedPermissive\_2](#CoercedPermissive_2)(k):

[CoercedPermissive\_2](#CoercedPermissive_2) = [Offline](#Offline).[GetCoercedPermissive\_2](#GetCoercedPermissive)(k)

return [CoercedPermissive\_2](#CoercedPermissive_2)

#

@

CoercedPermissive\_3，强制允许输入3

%

def [CoercedPermissive\_3](#CoercedPermissive_3)(k):

[CoercedPermissive\_3](#CoercedPermissive_3) = [Offline](#Offline).[GetCoercedPermissive\_3](#GetCoercedPermissive)(k)

return [CoercedPermissive\_3](#CoercedPermissive_3)

#

@

CoercedPermissive\_4，强制允许输入4

%

def [CoercedPermissive\_4](#CoercedPermissive_4)(k):

[CoercedPermissive\_4](#CoercedPermissive_4) = [Offline](#Offline).[GetCoercedPermissive\_4](#GetCoercedPermissive)(k)

return [CoercedPermissive\_4](#CoercedPermissive_4)

#

@

CoercedPermissive，返回采集到的“强制允许”结果

%

def [CoercedPermissive](#CoercedPermissive)(cr, k):

if (cr is **COERCED\_PERMISSIVE\_1**):

[CoercedPermissive](#CoercedPermissive) = [CoercedPermissive\_1](#CoercedPermissive_1)(k)

elif (cr is **COERCED\_PERMISSIVE\_2**):

[CoercedPermissive](#CoercedPermissive) = [CoercedPermissive\_2](#CoercedPermissive_2)(k)

elif (cr is **COERCED\_PERMISSIVE\_3**):

[CoercedPermissive](#CoercedPermissive) = [CoercedPermissive\_3](#CoercedPermissive_3)(k)

elif (cr is **COERCED\_PERMISSIVE\_4**):

[CoercedPermissive](#CoercedPermissive) = [CoercedPermissive\_4](#CoercedPermissive_4)(k)

elif (cr is **VARIANTS\_OVERLAP\_PERMISSIVE**):

[CoercedPermissive](#CoercedPermissive) = [OverlapTimerPermissive](#OverlapTimerPermissive)(k)

else:

[CoercedPermissive](#CoercedPermissive) = **False**

#

@

TSRreportReceived，收到TSR消息

%

def [TSRreportReceived](#TSRreportReceived)(lcId, k):

return [Message.Received](#Received)([TSRdownloadContent](#TSRdownloadContent)(lcId), k)

#

@

TSRreportAvailable，TSR消息可用

%

def [TSRreportAvailable](#TSRreportAvailable)(lcId, k):

return Message.[Available](#Available)([TSRreportReceived](#TSRreportReceived)(lcId, k),

[TSRdownloadContent](#TSRdownloadContent).CcLoopHour,

[ATPsetting](#ATPsetting).TSRvalidityTime,

[LastTSRreportAge](#LastTSRreportAge)(lcId, k-1),

k)

#

@

LastTSRreportAge，记录当前使用的TSR消息已经过了多长时间。

%

def [LastTSRreportAge](#LastTSRreportAge)(lcId, k):

return Message.[LastAge](#LastAge)([TSRreportAvailable](#TSRreportAvailable)(lcId, k),

[TSRdownloadContent](#TSRdownloadContent).CcLoopHour,

[LastTSRreportAge](#LastTSRreportAge)(lcId, k-1),

k)

#

@

ReceivedTSRdatabase，将LC发送的TSR消息报文映射到BLOCK数组中。对于线路上的每个BLOCK，判断其是否有对应的TSR，若有，则更新其首末点坐标和限速值，其中需将TSR消息中的坐标和速度单位转化为ATP软件使用的坐标和速度单位。

$

ATP shall map the TSR message received from LC to structure of block. It need to judge whether there is corresponding TSR for each BLOCK in the track map. If yes, ATP shall update the abscissa of the starting and ending points, as well as the restriction speed. During the process, it need to transfer the abscissa and speed unit of TSR message to the corresponding one used in ATP.

%

def [ReceivedTSRdatabase](#ReceivedTSRdatabase)(lc, k):

if (Initialization

or (Message.[Exists](#Exists)([DateSynchronizationReport](#DateSynchronizationReport)(lc), k)

and Message.[Exists](#Exists)([VersionAuthorization](#VersionAuthorization)(lc), k)

and not Message.[Exists](#Exists)([TSRdownloadContent](#TSRdownloadContent)(lc), k))

or (not Message.[IsMoreRecent](#IsMoreRecent)([ReceivedTSRdatabase](#ReceivedTSRdatabase)(lc, k-1).ValidityTime, [ATPtime](#ATPtime)(k))

and not [TSRreportAvailable](#TSRreportAvailable)(k))):

[SetAllBlockAsDefaultTsr](#SetAllBlockAsDefaultTsr)

elif ([TSRreportAvailable](#TSRreportAvailable)(lc, k)):

NewValidity = 0

if ([Message.ReplyLocalCC](#ReplyLocalCC)([TSRdownloadContent](#TSRdownloadContent)(lc).CcLoopHour)):

NewValidity = ([TSRdownloadContent](#TSRdownloadContent)(lc).CcLoopHour + [ATPsetting](#ATPsetting).TSRvalidityTime)

else:

NewValidity = ([ATPtime](#ATPtime)(k) + [ATPsetting](#ATPsetting).TSRvalidityTime

- ([OtherATPmaxTime](#OtherATPmaxTime)(k) - [TSRdownloadContent](#TSRdownloadContent)(lc).CcLoopHour))

[ReceivedTSRdatabase](#ReceivedTSRdatabase).ValidityTime = NewValidity

for tsr in [range](#range)(0, [TSRdownloadContent](#TSRdownloadContent)(lc).NumberOfTsr):

[SetTsrInFirstBlock](#SetTsrInFirstBlock)

[SetTsrInLastBlock](#SetTsrInLastBlock)

for blk in [range](#range)([TSRdownloadContent](#TSRdownloadContent)(lc).Tsr[tsr].FirstBlockId + 1,

[TSRdownloadContent](#TSRdownloadContent)(lc).Tsr[tsr].LastBlockId):

[SetTsrInIntermediateBlock](#SetTsrInIntermediateBlock)

else:

[ReceivedTSRdatabase](#ReceivedTSRdatabase) = [ReceivedTSRdatabase](#ReceivedTSRdatabase)(lc, k-1)

return [ReceivedTSRdatabase](#ReceivedTSRdatabase)

#

@

TrainEnergy，计算EB施加时刻的列车动能，作为能量监控使用的列车能量。

$

ATP shall calculate the train energy where EB indeed applied. The calculation shall consider the kinetic energy and the error of the potential energy. The [ATPsetting](#ATPsetting).MPauthAltitudeMaxErrorEnergy means an algorithm error caused by offline tool to calculate the compensation gradients.

%

[TrainEnergy](#TrainEnergy) = [V2EbApplied](#V2EbApplied) \* [V2EbApplied](#V2EbApplied)

+ [ATPsetting](#ATPsetting).MPauthAltitudeMaxErrorEnergy

#

@

ZoneVSLNotExceedTrainSpeedLimit，ATP应始终将项目配置的限速值为[ATPsetting](#ATPsetting).MPauthLimitSpeed作为安全速度限制区域。限制区能量

%

def [ZoneVSLNotExceedTrainSpeedLimit](#ZoneVSLNotExceedTrainSpeedLimit)(k):

return ([**TrainEnergy**](#TrainEnergy)(k) < [pow](#pow)([**ATPsetting**](#ATPsetting).MPauthLimitSpeed))

#

@

ZoneVSLnotExceedPSR，PSR作为区域型限速的情形，ATP应将以下两种类型的PSR作F为限制区域进行监控：该PSR是车尾最小定位上游的第一个PSR（即从该PSR所在位置到车尾最小定位之间没有其他PSR），如Figure 5‑18中的PSR2；该PSR位于车尾最小定位下游到EB实际位置之间，如Figure 5‑18中的PSR2,PSR3和PSR4。

%

def [ZoneVSLnotExceedPSR](#ZoneVSLnotExceedPSR)(k):

for Psr in [TrackMap.AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_PSR**,

[TrackMap.BlockOrigin](#TrackMap.BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k))):

if ([**TrainEnergy**](#TrainEnergy)(k) >= [pow](#pow)(Psr.SpeedLimit)

and (not [TrackMap.LocationBtwTwoLocs](#LocationBtwTwoLocs)(Psr.Location,

[TrackMap.BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainRearLocation](#TrainRearLocation)(k).Min)

or ([TrackMap.ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(**SGL\_PSR**, Psr.Location,

[TrainRearLocation](#TrainRearLocation)(k).Min) is **None**))):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedPSR，PSR作为点型限速的情形。其中，Energy.AccumulationPotentialEnergy表示根据限制点所在坡度或EB最小保障率累加计算目标位置的势能，EB最小保障率应根据所在位置的Grip值（Normal或Reduce）选取[ATPsetting](#ATPsetting).EBguaranteedAccNormalGrip或[ATPsetting](#ATPsetting).EBguaranteedAccReducedGrip。能量计算的原理和方法见[REF10]。当车尾在一个较低的PSR（或TSR）中时，若当前车速小于该PSR限速，而计算出的V2速度大于该PSR限速，按照上述处理方式，也会导致EB，尽管当列车运行到X2位置时，列车也许已经离开了该PSR区域。

$

If the train tail intersected with a PSR (or TSR) area, and the speed of train is lower but the [V2EbApplied](#V2EbApplied) is higher than the limitation. In accordance with the above approach will result in EB, although when the train runs to the EB applied position, the train may have left the PSR area.

%

def PointVSLnotExceedPSR(k):

for Psr in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_PSR**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if ([**TrainEnergy**](#TrainEnergy)(k) >= ([pow](#pow)(Psr.SpeedLimit)

+ ([[Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Psr.Location)))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedBSR，车身范围内有BSR的情形

%

def ZoneVSLnotExceedBSR(k):

for Block in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_NEW\_BLOCK**,

[TrackMap](#TrackMap).[BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)))):

if (Block.Bsr is not **None**

and not [CoercedPermissive](#CoercedPermissive)(Block.[CoercedPermissive](#CoercedPermissive), k)

and not [VariantValue](#VariantValue)(Block.Bsr.Variant, k)

and [TrainEnergy](#TrainEnergy)(k) >= [pow](#pow)(Block.Bsr.Speed)):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedBSR，列车下游有BSR的情形

%

def [PointVSLnotExceedBSR](#PointVSLnotExceedBSR)(k):

for Block in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_NEW\_BLOCK**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if (Block.Bsr is not **None**

and not [CoercedPermissive](#CoercedPermissive)(Block.Bsr.[CoercedPermissive](#CoercedPermissive), k)

and not [VariantValue](#VariantValue)(Block.Bsr.Variant, k)

and [TrainEnergy](#TrainEnergy)(k) >= ([pow](#pow)(Block.Bsr.Speed)

+ ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Block.Bsr.Position)))):

return **False**

else:

continue

else:

return **True**

#

@

TSRcontrolInhibition，不处理TSR信息。其状态来自于项目可配置的列车输入采集。

$

According to the status of TSRcontrollinhibition, ATP can judge whether it is necessary to handle TSR information.

%

def [TSRcontrolInhibition](#TSRcontrolInhibition)(k):

return [Offline.GetTSRcontrolInhibition](#GetTSRcontrolInhibition)(k)

#

@

ZoneVSLnotExceedTSR，TSR作为区域型限速的情形。即对于从车尾所在Block起始点到EB施加位置内的所有Block，当满足以下条件时，认为列车超过了TSR限速：未禁止处理TSR信息；且该Block存在TSR；且列车定位与该TSR区域有交集；且计算的列车能量大于上述TSR的限制能量。其中TSRonBlock表示获取指定Block上TSR的值。

%

def ZoneVSLnotExceedTSR(k):

for Blk in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_NEW\_BLOCK**,

[TrackMap](#TrackMap).[BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min)

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)))):

Tsr = [TSRonBlock](#TSRonBlock)(Blk, [TrackMap.OppositeOrientation](#OppositeOrientation)([TrainFrontOrientation](#TrainFrontOrientation)(k)), k)

if (not [TSRcontrolInhibition](#TSRcontrolInhibition)(k)

and Tsr is not **None**

and not [TrackMap](#TrackMap).[LocationBtwTwoLocs](#LocationBtwTwoLocs)(Tsr.Position,

[TrackMap](#TrackMap).[BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainRearLocation](#TrainRearLocation)(k).Min)

and [TrainEnergy](#TrainEnergy)(k) >= [pow](#pow)(Tsr.Value)):

return **False**

else:

continue

return **True**

def [TSRonBlock](#TSRonBlock)(blockId, direction, k):

if (not [ReceivedTSRdatabase](#ReceivedTSRdatabase).[Blocks](#Blocks)[blockId].NotRestrictionApplication):

if (direction is UP):

return ([**ReceivedTSRdatabase**](#ReceivedTSRdatabase).[Blocks](#Blocks)[blockId].Position[0],

[ReceivedTSRdatabase](#ReceivedTSRdatabase).[Blocks](#Blocks)[blockId].Value)

else:

return ([ReceivedTSRdatabase](#ReceivedTSRdatabase).[Blocks](#Blocks)[blockId].Position[1],

[ReceivedTSRdatabase](#ReceivedTSRdatabase).[Blocks](#Blocks)[blockId].Value)

else:

return **None**

#

@

PointVSLnotExceedTSR，TSR作为点型限速的情形

%

def [PointVSLnotExceedTSR](#PointVSLnotExceedTSR)(k):

for Blk in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_NEW\_BLOCK**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

Tsr = [TSRonBlock](#TSRonBlock)(Blk, [TrainFrontOrientation](#TrainFrontOrientation)(k), k)

if (not [TSRcontrolInhibition](#TSRcontrolInhibition)(k)

and Tsr is not **None**

and [TrainEnergy](#TrainEnergy)(k) >= ([pow](#pow)(Tsr.Value)

+ ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Tsr.Position)))):

return **False**

else:

continue

else:

return **True**

#

@

ZoneVSLnotExceedOTE，Open track end作为区域型限速的情形

%

def ZoneVSLnotExceedOTE(k):

if ([**TrackMap**.ExistSingularityInZone](#ExistSingularityInZone)(SGL\_OPEN\_TRACK\_END, [TrainFrontLocation](#TrainFrontLocation)(k).Max,

[X2EbApplied](#X2EbApplied)(k)) is not **None**)):

return **False**

else:

return **True**

#

@

PointVSLnotExceedOTE，Open track end作为点型限速的情形

%

def PointVSLnotExceedOTE(k):

Ote = ([TrackMap](#TrackMap).[ExistSingularityInZone](#ExistSingularityInZone)(SGL\_OPEN\_TRACK\_END,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

[ATPsetting](#ATPsetting).EOAmaxDistance)

if (Ote is not **None**)

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Ote.Location))):

return **False**

else:

return **True**

#

@

ZoneVSLnotExceedCTE，Close track end作为区域型限速的情形

%

def ZoneVSLnotExceedCTE(k):

cte = ([**TrackMap**](#TrackMap).[ExistSingularityInZone](#ExistSingularityInZone)(**SGL\_CLOSE\_TRACK\_END**, [TrainFrontLocation](#TrainFrontLocation)(k).Max,

[X2EbApplied](#X2EbApplied)(k)))

if (cte is not **None**

and [TrainEnergy](#TrainEnergy)(k) >= [pow](#pow)(cte.SpeedLimit)):

return **False**

else:

return **True**

#

@

PointVSLnotExceedCTE，Close track end作为点型限速的情形

%

def PointVSLnotExceedCTE(k):

cte = ([**TrackMap**](#TrackMap).[ExistSingularityInZone](#ExistSingularityInZone)(**SGL\_CLOSE\_TRACK\_END**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

[ATPsetting](#ATPsetting).EOAmaxDistance))

if (cte is not **None**

and [TrainEnergy](#TrainEnergy)(k) >= ([[pow](#pow)](#pow)(cte.SpeedLimit)

+ ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

cte.Location)))):

return **False**

else:

return **True**

#

@

ZoneVSLnotExceedSignal，信号机作为区域型限速的情形

%

def ZoneVSLnotExceedSignal(k):

for Sig in [TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_SIGNAL**, [TrainFrontLocation](#TrainFrontLocation)(k).Max, [X2EbApplied](#X2EbApplied)(k)):

if (([CoercedRestrictive](#CoercedRestrictive)(Sig.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Sig.Variant, k))

and (not [CoercedPermissive](#CoercedPermissive)(Sig.[CoercedPermissive](#CoercedPermissive), k)

and not [VariantValue](#VariantValue)(Sig.OverlapVariant, k))):

return **False**

else:

continue

else:

return **True**

#

@

PointVSLnotExceedSignal，信号机作为点型限速的情形

%

def PointVSLnotExceedSignal(k):

for Sig in [TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_SIGNAL**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if (([**CoercedRestrictive**](#CoercedRestrictive)(Sig.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Sig.Variant, k))

and (not [CoercedPermissive](#CoercedPermissive)(Sig.[CoercedPermissive](#CoercedPermissive), k)

and not [VariantValue](#VariantValue)(Sig.OverlapVariant, k))):

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Sig.Location))):

return **False**

else:

continue

else:

return **True**

#

@

ZoneVSLnotExceedOverlap，Overlap作为区域型限速的情形

%

def ZoneVSLnotExceedOverlap(k):

for Overlap in [TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_OVERLAP\_END**, [TrainFrontLocation](#TrainFrontLocation)(k).Max,

[X2EbApplied](#X2EbApplied)(k)):

Signal = [TrackMap.ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(SGL\_SIGNAL, [TrainFrontLocation](#TrainFrontLocation)(k).Max,

Overlap.Location)

if (Signal is not **None**

and ([**CoercedRestrictive**](#CoercedRestrictive)(Signal.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Signal.Variant, k))

and ([CoercedPermissive](#CoercedPermissive)(Signal.[CoercedPermissive](#CoercedPermissive), k)

or [VariantValue](#VariantValue)(Overlap.Variant, k))):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedOverlap，Overlap作为点型限速的情形

%

def PointVSLnotExceedOverlap(k):

for Overlap in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_OVERLAP\_END**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

Signal = ([TrackMap.ExistSingBtwTwoLocs](#ExistSingBtwTwoLocs)(SGL\_SIGNAL,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

Overlap.Location))

if (Signal is not **None**

and ([**CoercedRestrictive**](#CoercedRestrictive)(Signal.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Signal.Variant, k))

and ([**CoercedPermissive**](#CoercedPermissive)(Signal.[CoercedPermissive](#CoercedPermissive), k)

or [VariantValue](#VariantValue)(Overlap.Variant, k))

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Overlap.Location))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedSwitch，非受控道岔作为区域型限速的情形

%

def ZoneVSLnotExceedSwitch(k):

for Switch in [TrackMap.AllSwitchesInZone](#AllSwitchesInZone)([TrainFrontLocation](#TrainFrontLocation)(k).Max, [X2EbApplied](#X2EbApplied)(k)):

if ([VariantValue](#VariantValue)(Switch.Variant1, k) == [VariantValue](#VariantValue)(Switch.Variant2, k)):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedSwitch，非受控道岔作为点型限速的情形

%

def PointVSLnotExceedSwitch(k):

for Switch in ([TrackMap.AllSwitchesInZone](#AllSwitchesInZone)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if ([VariantValue](#VariantValue)(Switch.Variant1, k) == [VariantValue](#VariantValue)(Switch.Variant2, k)

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Switch.Location))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedPZ，PZ作为区域型限速。ATP应监控与列车定位有以下两种关系的限制状态保护区：该保护区的起始点在车尾最小定位到紧急制动施加位置之间；或，该保护区起始点在车尾最小定位上游，但车尾最小定位在该保护区范围内。

%

def ZoneVSLnotExceedPZ(k):

for Pz in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_PROTECTION\_ZONE**,

[TrackMap.BlockOrigin](#TrackMap.BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)))):

if ((not [TrackMap.LocationBtwTwoLocs](#LocationBtwTwoLocs)(Pz.Location,

[TrackMap.BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainRearLocation](#TrainRearLocation)(k).Min)

or [TrackMap.LocationInZone](#LocationInZone)([TrainRearLocation](#TrainRearLocation)(k).Min, Pz.Location, Pz.Length))

and not [CoercedPermissive](#CoercedPermissive)(Pz.[CoercedPermissive](#CoercedPermissive), k)

and ([CoercedRestrictive](#CoercedRestrictive)(Pz.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Pz.Variant, k))):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedPZ，PZ作为点型限速的情形

%

def PointVSLnotExceedPZ(k):

for Pz in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_PROTECTION\_ZONE**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if (not [CoercedPermissive](#CoercedPermissive)(Pz.[CoercedPermissive](#CoercedPermissive), k)

and ([CoercedRestrictive](#CoercedRestrictive)(Pz.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Pz.Variant, k))

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Pz.Location))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedPSD，PSD作为区域型限速。ATP应监控与列车定位有以下两种关系的限制状态屏蔽门区域：该屏蔽门区的起始点在车尾最小定位到紧急制动施加位置之间；或，该屏蔽门区起始点在车尾最小定位上游，但车尾最小定位在该屏蔽门区范围之内。

%

def ZoneVSLnotExceedPSD(k):

for Psd in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_PSD\_ZONE**,

[TrackMap.BlockOrigin](#TrackMap.BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)))):

if ((not [TrackMap.LocationBtwTwoLocs](#LocationBtwTwoLocs)(Psd.Location,

[TrackMap.BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainRearLocation](#TrainRearLocation)(k).Min)

or [TrackMap.LocationInZone](#LocationInZone)([TrainRearLocation](#TrainRearLocation)(k).Min, Psd.Location, Psd.Length))

and not [CoercedPermissive](#CoercedPermissive)(Psd.[CoercedPermissive](#CoercedPermissive), k)

and ([CoercedRestrictive](#CoercedRestrictive)(Psd.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Psd.Variant, k))):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedPSD，PSD作为点型限速的情形

%

def PointVSLnotExceedPSD(k):

for Psd in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_PSD\_ZONE**,

[[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k) )](#EBAlocation),

ATPsetting.EOAmaxDistance)):

if (not [CoercedPermissive](#CoercedPermissive)(Psd.[CoercedPermissive](#CoercedPermissive), k)

and ([CoercedRestrictive](#CoercedRestrictive)(Psd.[NotCoercedRestrictive](#NotCoercedRestrictive), k)

or not [VariantValue](#VariantValue)(Psd.Variant, k))

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

Psd.Location))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedZC，非授权ZC作为区域型限速的情形

%

def ZoneVSLnotExceedZC(k):

for block in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(**SGL\_NEW\_BLOCK**, [TrainRearLocation](#TrainRearLocation)(k).Min,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)))):

if (not [BlockModeUsed](#BlockModeUsed)(k)

and not [VersionAuthorizedByLC](#VersionAuthorizedByLC)([TrackMap](#TrackMap).[ZCId](#ZCId)(block.Id), k)):

return **False**

else:

continue

return **True**

#

@

PointVSLnotExceedZC，非授权ZC边界作为点型限速的情形

%

def PointVSLnotExceedZC(k):

for NewBlock in ([TrackMap.AllSingsInZone](#AllSingsInZone)(**SGL\_NEW\_BLOCK**,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k)),

ATPsetting.EOAmaxDistance)):

if (not [BlockModeUsed](#BlockModeUsed)(k)

and not [VersionAuthorizedByLC](#VersionAuthorizedByLC)([TrackMap](#TrackMap).[ZCId](#ZCId)(NewBlock.Id), k)

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

NewBlock.Location))):

return **False**

else:

continue

return **True**

#

@

ZoneVSLnotExceedEOA，CBTC下EOA作为区域型限速的情形

%

def ZoneVSLnotExceedEOA(k):

if ([[CBTCmodeEOAvalid](#CBTCmodeEOAvalid)](#EndOfAuthorityValid)(k)

and ([TrackMap](#TrackMap).[LocationBtwTwoLocs](#LocationBtwTwoLocs)([[CBTCmodeEOAlocation](#CBTCmodeEOAlocation)](#EOAlocation)(k),

[TrainFrontLocation](#TrainFrontLocation)(k).Min,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k))))):

return **False**

else:

return **True**

#

@

PointVSLnotExceedEOA，CBTC下EOA作为点型限速的情形

%

def PointVSLnotExceedEOA(k):

if ([[CBTCmodeEOAvalid](#CBTCmodeEOAvalid)](#EndOfAuthorityValid)(k)

and not ([TrackMap](#TrackMap).[LocationBtwTwoLocs](#LocationBtwTwoLocs)([[CBTCmodeEOAlocation](#CBTCmodeEOAlocation)](#EOAlocation)(k),

[TrainFrontLocation](#TrainFrontLocation)(k).Min,

[TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max, [X2EbApplied](#X2EbApplied)(k))))

and [TrainEnergy](#TrainEnergy)(k) >= ([Energy.AccumulationPotentialEnergy](#AccumulationPotentialEnergy)

([TrackMap.CalculateZoneBorder](#CalculateZoneBorder)([TrainFrontEnd](#TrainFrontEnd)(k).Max,

[X2EbApplied](#X2EbApplied)(k)),

[[CBTCmodeEOAlocation](#CBTCmodeEOAlocation)](#EOAlocation)(k)))):

return **False**

else:

return **True**

#

@

ZoneVSLNotExceed，判断有无限制区域超能。

$

ATP shall determine whether train exceeds the vital speed limitation of the restrictive zone, by comparing the energy between the energy of the train and the energy of the zone.

%

def [ZoneVSLNotExceed](#ZoneVSLNotExceed)(k):

return ([EndOfAuthorityValid](#EndOfAuthorityValid)(k)

and [ZoneVSLNotExceedTrainSpeedLimit](#ZoneVSLNotExceedTrainSpeedLimit)(k)

and [ZoneVSLnotExceedPSR](#ZoneVSLnotExceedPSR)(k)

and [ZoneVSLnotExceedBSR](#ZoneVSLnotExceedBSR)(k)

and [ZoneVSLnotExceedTSR](#ZoneVSLnotExceedTSR)(k)

and [ZoneVSLnotExceedOTE](#ZoneVSLnotExceedOTE)(k)

and [ZoneVSLnotExceedCTE](#ZoneVSLnotExceedCTE)(k)

and [ZoneVSLnotExceedSignal](#ZoneVSLnotExceedSignal)(k)

and [ZoneVSLnotExceedOverlap](#ZoneVSLnotExceedOverlap)(k)

and [ZoneVSLnotExceedSwitch](#ZoneVSLnotExceedSwitch)(k)

and [ZoneVSLnotExceedPZ](#ZoneVSLnotExceedPZ)(k)

and [ZoneVSLnotExceedPSD](#ZoneVSLnotExceedPSD)(k)

and [ZoneVSLnotExceedZC](#ZoneVSLnotExceedZC)(k)

and [ZoneVSLnotExceedEOA](#ZoneVSLnotExceedEOA)(k))

#

@

PointVSLNotExceed，判断有无限制点超能。

$

ATP shall determine whether train exceeds the vital speed limitation of the restrictive point, by comparing the energy between the energy of the train and the kinetic added potential energy of the point.

%

def [PointVSLNotExceed](#PointVSLNotExceed)(k):

return ([EndOfAuthorityValid](#EndOfAuthorityValid)(k)

and [PointVSLnotExceedPSR](#PointVSLnotExceedPSR)(k)

and [PointVSLnotExceedBSR](#PointVSLnotExceedBSR)(k)

and [PointVSLnotExceedTSR](#PointVSLnotExceedTSR)(k)

and [PointVSLnotExceedOTE](#PointVSLnotExceedOTE)(k)

and [PointVSLnotExceedCTE](#PointVSLnotExceedCTE)(k)

and [PointVSLnotExceedSignal](#PointVSLnotExceedSignal)(k)

and [PointVSLnotExceedOverlap](#PointVSLnotExceedOverlap)(k)

and [PointVSLnotExceedSwitch](#PointVSLnotExceedSwitch)(k)

and [PointVSLnotExceedPZ](#PointVSLnotExceedPZ)(k)

and [PointVSLnotExceedPSD](#PointVSLnotExceedPSD)(k)

and [PointVSLnotExceedZC](#PointVSLnotExceedZC)(k)

and [PointVSLnotExceedEOA](#PointVSLnotExceedEOA)(k))

#

@

MotionProtectionInhibition，表示ATP不负责列车位置的监控。其状态来自于项目可配置的列车输入采集。

%

def [MotionProtectionInhibition](#MotionProtectionInhibition)(k):

return [Offline.GetMotionProtectionInhibition](#GetMotionProtectionInhibition)(k)

#

@

TrainPossiblyInOverEnergy，列车能量大于限制点或限制区能量，即超能。

$

If the train energy exceeds the zone of point vital speed limitation, ATP shall consider the train possibly over energy.

%

def [TrainPossiblyInOverEnergy](#TrainPossiblyInOverEnergy)(k):

return (not [ZoneVSLNotExceed](#ZoneVSLNotExceed)(k)

or not [PointVSLNotExceed](#PointVSLNotExceed)(k))

#

@

TrainEnergyControlDisabled，在RM模式下不报超能。

$

If the RMF or RMR mode selected, ATP shall not monitor the train energy.

%

def [TrainEnergyControlDisabled](#TrainEnergyControlDisabled)(k):

return [MotionProtectionInhibition](#MotionProtectionInhibition)(k)

#

@

EBforOverEnergy，超能后是否输出EB

$

ATP shall request emergency braking if train is possibly in over-energy and train speed control enabled and if following conditions fulfilled:the train is not detected at filtered stop,or the train is detected at filtered stop and: safe immobilization customization setting for this control indicates to use emergency brake，or safe immobilization customization setting for this control indicates to use emergency brake when it was already applied.

%

def [EBforOverEnergy](#EBforOverEnergy)(k):

return ([**TrainPossiblyInOverEnergy**](#TrainPossiblyInOverEnergy)(k)

and not [TrainEnergyControlDisabled](#TrainEnergyControlDisabled)(k)

and (not [TrainFilteredStopped](#TrainFilteredStopped)(k)

or ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([**ATPsetting**](#ATPsetting).MPauthImmoBehaviourAtFS

is **IB\_APPLY\_EMERGENCY\_BRAKE**)

or (([**ATPsetting**](#ATPsetting).MPauthImmoBehaviourAtFS

is **IB\_APPLY\_EMERGENCY\_BRAKE\_WHEN\_TRIGGERED**)

and not [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1)))))

#

@

PBforOverEnergy，超能停车后是否继续输出PB

$

ATP shall request parking braking if train is possibly in over-energy and train speed control enabled and if following conditions fulfilled:the train is detected at filtered stop,and safe immobilization customization setting for this control indicates to use parking brake.

%

def [PBforOverEnergy](#PBforOverEnergy)(k):

return ([**TrainPossiblyInOverEnergy**](#TrainPossiblyInOverEnergy)(k)

and not [TrainEnergyControlDisabled](#TrainEnergyControlDisabled)

and ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([**ATPsetting**](#ATPsetting).MPauthImmoBehaviourAtFS is **IB\_APPLY\_PARKING\_BRAKE**)))

#

@

NotOnRestrictiveMoralTimeArea\_1，当列车定位时，ATP需判断**END\_1**端车头的内外侧 定位是否与该端车头朝向的“限制状态”信号机下游的模糊时间区有无交集。其中模糊时间区定义为信号机下游长度为[ATPsetting](#ATPsetting).MTdistance的一段范围。当满足下列所有条件时，设置[NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1)为**True**：列车已确认定位；并且：**END\_1**端车头的内外侧定位与**END\_1**端车头朝向的信号机下游模糊区没有交集；或者，**END\_1**端车头的内外侧定位与**END\_1**端车头朝向的信号机下游模糊区有交集，但该信号机是允许状态。否则，应设置[NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1)为**False**。

%

def [NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1)(k):

Signal = [TrackMap.ExistSingularityInReverseZone](#ExistSingularityInReverseZone)(**SGL\_SIGNAL**,

[TrainLocation](#TrainLocation).Ext1,

[ATPsetting](#ATPsetting).MTdistance + [TrainLocation](#TrainLocation)(k).Uncertainty)

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and (Signal is **None**

or not Signal.BmMoralTime

or [VariantValue](#VariantValue)(Signal.Variant, k)))

#

@

NotOnRestrictiveMoralTimeArea\_2，当列车定位时，ATP需判断**END\_2**端车头的内外侧定位是否与该端车头朝向的“限制状态”信号机下游的模糊时间区有无交集。当满足下列所有条件时，设置[NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2)为**True**：列车已确认定位；并且：**END\_2**端车头的内外侧定位与**END\_2**端车头朝向的信号机下游模糊区没有交集；或者，**END\_2**端车头的内外侧定位与**END\_2**端车头朝向的信号机下游模糊区有交集，但该信号机是允许状态。否则，应设置[NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2)为**False**。

%

def [NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2)(k):

Signal = [TrackMap.ExistSingularityInReverseZone](#ExistSingularityInReverseZone)(**SGL\_SIGNAL**,

[TrainLocation](#TrainLocation).Ext2,

[ATPsetting](#ATPsetting).MTdistance + [TrainLocation](#TrainLocation)(k).Uncertainty)

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and (Signal is **None**

or not Signal.BmMoralTime

or [VariantValue](#VariantValue)(Signal.Variant, k)))

#

@

RouteExclusivityGuaranted\_1，如果列车在车头1对应方向且限制状态的模糊时间区内超过项目设定时间，则ATP应将该值设为限制状态。其中MoralTimeTimer\_1为记录列车在车头1对应方向的限制状态模糊时间区内的时间。

$

If ATP cannot determine train is [NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1), and if this situation lasts more than [ATPsetting](#ATPsetting). MTtimeout cycles, ATP shall consider that route exclusivity is not guaranteed and [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) shall be set to **False**. If ATP detects that train is [NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1), route exclusivity shall consider as guaranteed for that direction of travel and [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) shall set to **True**

%

def [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1)(k):

if (Initialization):

[RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) = **False**

elif ([RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1)(k-1)

and not [NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1)(k)):

if (MoralTimeTimer\_1(k-1) < round.floor([ATPsetting](#ATPsetting).MTtimeout)):

MoralTimeTimer\_1 = MoralTimeTimer\_1(k-1) + 1

else:

[RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) = **False**

elif ([NotOnRestrictiveMoralTimeArea\_1](#NotOnRestrictiveMoralTimeArea_1)(k)):

MoralTimeTimer\_1 = 1

[RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) = **True**

else:

[RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) = [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1)(k-1)

return [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1)

#

@

RouteExclusivityGuaranted\_2，如果列车在车头2对应方向且限制状态的模糊时间区内超过项目设定时间，则ATP应将该值设置为限制状态，其中MoralTimeTimer\_2为记录列车在车头2对应方向限制状态模糊时间区内的时间。

$

If ATP cannot determine train is [NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2), and if this situation lasts more than [ATPsetting](#ATPsetting).MTtimeout cycles, ATP shall consider that route exclusivity is not guaranteed and [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) shall set to **False**. If ATP detects that train is [NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2), route exclusivity shall consider as guaranteed for that direction of travel and [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) shall set to **True**

%

def [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2)(k):

if (Initialization):

[RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) = **False**

elif (RouteExclusivityGuaranted 2(k-1)

and not [NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2)(k)):

if (MoralTimeTimer\_2(k-1) < round.floor([ATPsetting](#ATPsetting).MTtimeout)):

MoralTimeTimer\_2 = MoralTimeTimer\_2(k-1) + 1

else:

[RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) = **False**

elif ([NotOnRestrictiveMoralTimeArea\_2](#NotOnRestrictiveMoralTimeArea_2)(k)):

MoralTimeTimer\_2 = 1

[RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) = **True**

else:

[RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) = [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2)(k-1)

return [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2)

#

@

HazardousMotionOnNonExclusiveRoute，非RM的BM模式下，如果列车在激活端车头方向的限制状态的Moral Time区停止超时预设时间，则ATP认为当前处于“非独占进路”的风险之中。

$

If [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1) is **False**, ATP shall request emergency braking if and only if:[TrainFrontEnd](#TrainFrontEnd) is not **END\_2**,RM forward nor RM reverse are not selected,and block mode is not selected. If [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2) is **False**, ATP shall request emergency braking if and only if:[TrainFrontEnd](#TrainFrontEnd) is not **END\_1**,RM forward nor RM reverse driving mode are not selected,and block mode is not selected.

%

def [HazardousMotionOnNonExclusiveRoute](#HazardousMotionOnNonExclusiveRoute)(k):

return (((not [RouteExclusivityGuaranted\_1](#RouteExclusivityGuaranted_1)(k) and ([TrainFrontEnd](#TrainFrontEnd)(k)!= **END\_2**))

or (not [RouteExclusivityGuaranted\_2](#RouteExclusivityGuaranted_2)(k) and ([TrainFrontEnd](#TrainFrontEnd)(k)!= **END\_1**)))

and not [MotionProtectionInhibition](#MotionProtectionInhibition)(k)

and [BlockModeUsed](#BlockModeUsed)(k))

#

@

PBonNonExclusiveRoute，当由于MoralTime监控导致的停车后，是否保持输出停车制动的取决于项目配置。

$

ATP shall request parking braking if train considered too near from a non-exclusive route and if following conditions are fulfilled:the train is detected at filtered stop,safe immobilization customization setting for this control indicates to use parking brake.

%

[PBonNonExclusiveRoute](#PBonNonExclusiveRoute)(k)

= [HazardousMotionOnNonExclusiveRoute](#HazardousMotionOnNonExclusiveRoute)(k)

and [TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([ATPsetting](#ATPsetting).MTimmoBehaviourAtFS == **IB\_APPLY\_PARKING\_BRAKE**)

#

@

EBonNonExclusiveRoute，如果当前处于“非独占进路”的风险中，且列车在移动，则ATP应当输出EB；如果当前已停车，则是否继续输出EB取决于项目配置。

$

ATP shall request emergency braking if train considered too near from a non-exclusive route and if following conditions are fulfilled:the train is not detected at filtered stop,or the train is detected at filtered stop and:safe immobilization customization setting for this control indicates to use emergency brake,or safe immobilization customization setting for this control indicates to use emergency brake when it was already applied.

%

[EBonNonExclusiveRoute](#EBonNonExclusiveRoute)(k)

= （[HazardousMotionOnNonExclusiveRoute](#HazardousMotionOnNonExclusiveRoute)(k)

and (not [TrainFilteredStopped](#TrainFilteredStopped)(k)

or ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and (([ATPsetting](#ATPsetting).MTimmoBehaviourAtFS == **IB\_APPLY\_EMERGENCY\_BRAKE**)

or (([ATPsetting](#ATPsetting).MTimmoBehaviourAtFS == **IB\_APPLY\_EMERGENCY\_BRAKE\_WHEN\_TRIGGERED**)

and not [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1))))))

#

@

PermissiveZoneLogicalInput，允许区逻辑输入。其中，列车定位完全包含在vital zone中的条件如下：

%

def [PermissiveZoneLogicalInput](#PermissiveZoneLogicalInput)(PzType, k):

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and IncludedInVitalZone(PzType, k))

def IncludedInVitalZone(PzType, k):

for Vz in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(SGL\_VITAL\_ZONE,

[TrackMap](#TrackMap).[BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainFrontLocation](#TrainFrontLocation)(k).Max)):

if (Vz.[PermissiveZoneLogicalInput](#PermissiveZoneLogicalInput) is PzType

and [TrackMap](#TrackMap).[LocationInZone](#LocationInZone)([TrainFrontLocation](#TrainFrontLocation)(k).Max,

Vz.Location,

Vz.Length,

Vz.Orientation)

and [TrackMap](#TrackMap).[LocationInZone](#LocationInZone)([TrainRearLocation](#TrainRearLocation)(k).Min,

Vz.Location,

Vz.Length,

Vz.Orientation)

and (Vz.Variant is **None**

or ([VariantValue](#VariantValue)(Vz.Variant, k)))):

return **True**

else:

continue

return **False**

#

@

NotRestrictiveZoneLogicalInput，非限制区逻辑输入。其中，列车与vital zone没有交集的判别条件如下：

%

def [NotRestrictiveZoneLogicalInput](#NotRestrictiveZoneLogicalInput)(NrzType, k):

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and NotIntersectVitalZone(NrzType, k))

def NotIntersectVitalZone(NrzType, k):

for Vz in ([TrackMap](#TrackMap).[AllSingsBtwTwoLocs](#AllSingsBtwTwoLocs)(SGL\_VITAL\_ZONE,

[TrackMap](#TrackMap).[BlockOrigin](#BlockOrigin)([TrainRearLocation](#TrainRearLocation)(k).Min),

[TrainFrontLocation](#TrainFrontLocation)(k).Max)):

if (Vz.[NotRestrictiveZoneLogicalInput](#NotRestrictiveZoneLogicalInput) is NrzType

and ([TrackMap](#TrackMap).[IntersectionOfTwoZones](#IntersectionOfTwoZones)([TrainRearLocation](#TrainRearLocation)(k).Min,

[TrainFrontLocation](#TrainFrontLocation)(k).Max,

vz.Location,

[TrackMap](#TrackMap).[CalculateZoneBorder](#CalculateZoneBorder)(vz.Location, vz.Length))

is not **None**)

and (Vz.Variant is **None**

or not [VariantValue](#VariantValue)(Vz.Variant, k))):

return **False**

else:

continue

return **True**

#

@

NoUndetectableDanger\_1，已监控向**END\_1**方向的运行，其状态来自于项目可配置的列车输入采集。

$

The No Undetectable Danger in Extremity 1 shall be consider as permissive status according to project configuration.

%

def [NoUndetectableDanger\_1](#NoUndetectableDanger_1)(k):

return [Offline](#Offline).[NoUndetectableDanger\_1](#NoUndetectableDanger_1)(k)

#

@

NoUndetectableDanger\_2，其状态来自于项目可配置的列车输入采集。

$

The "No Undetectable Danger in Extremity 2" shall be consider as permissive status according to project configuration.

%

def [NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k):

return [Offline](#Offline).[NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k)

#

@

UndetectableDangerRiskForNoNUDE，当前两端车头都没有NUDE输入，则认为列车存在“无法侦测的风险”。

$

If there is neither No Undetectable Danger in Extremity 1 nor No Undetectable Danger in Extremity 2 inputs, ATP shall consider the train is possible under the risk of undetectable danger.

%

def [UndetectableDangerRiskForNoNUDE](#UndetectableDangerRiskForNoNUDE)(k):

return (not [NoUndetectableDanger\_1](#NoUndetectableDanger_1)(k)

and not [NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k))

#

@

PBforUndetectableDangerRisk，当停车且存在“无法侦测的风险”时，如果项目配置为输出停车制动，则ATP应当输出停车制动。

$

ATP shall request a parking braking if the possibility of an undetected danger has proven to be and if following conditions are fulfilled:the train is detected at filtered stop,safe immobilization customization setting for this control indicates to use parking brake.

%

[PBforUndetectableDangerRisk](#PBforUndetectableDangerRisk)(k)

= ([UndetectableDangerRiskForNoNUDE](#UndetectableDangerRiskForNoNUDE)(k)

and [TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([ATPsetting](#ATPsetting).NUDEimmoBehaviourAtFS == **IB\_APPLY\_PARKING\_BRAKE**))

#

@

NUDEdistanceAccount\_1，监控当司机未授权向**END\_1**方向运行时，列车向**END\_1**方向运行的距离，该值为非负数，若在初始化阶段，或NUDE1为**True**，或已经EB并停车，则等于0；否则，当测速无效时，将其设置为默认值; 否则，当里程计已初始化后：如果[MaximumTrainMotion](#MaximumTrainMotion)大于0，则等于上周期累加距离加上本周期最大位移，最小取0。而如果[MaximumTrainMotion](#MaximumTrainMotion)小于等于0，则使用上周期值加最小位移（实际上就是减小该累加值，倒车），最小取0。否则，保持累加距离不变。

$

When the driver does not authorize the train running toward the **END\_1**, ATP shall accumulate the distance of the train running toward to the **END\_1**.If in initialization, or the [NoUndetectableDanger\_1](#NoUndetectableDanger_1) is **True**, or the train has triggered EB and has stopped, ATP set this distance to 0;Else if train kinematic has invalid, ATP set this distance to the default value.Else if the odometer has initialized:If the [MaximumTrainMotion](#MaximumTrainMotion)is greater than 0, ATP accumulate the maximum movement in this cycle with the distance of last cycle;Or if the [MaximumTrainMotion](#MaximumTrainMotion)is less than or equal to 0, ATP use the minimum movement of this cycle plus to the distance last cycle (in fact, decrease the accumulated distance). The minimum of this accumulated distance is 0.Otherwise, keep the distance unchanged.

%

def [NUDEdistanceAccount\_1](#NUDEdistanceAccount_1)(k):

if (INTIALIZATION

or [NoUndetectableDanger\_1](#NoUndetectableDanger_1)(k)

or ([EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE)(k-1) and [TrainFilteredStopped](#TrainFilteredStopped)(k)))

return 0

elif ([ValidTrainKinematic](#ValidTrainKinematic)(k) != **True**)

return [ATPsetting](#ATPsetting).NUDEdistanceWithoutMotionAvailable

elif ([OdometerState](#OdometerState)(k) is **INITIALIZED**)

if ([End1RunningForward](#End1RunningForward)(k))

return [max](#max)(0, [NUDEdistanceAccount\_1](#NUDEdistanceAccount_1)(k-1)+ [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [max](#max)(0, [NUDEdistanceAccount\_1](#NUDEdistanceAccount_1)(k-1)+ [MinimumTrainMotion](#MinimumTrainMotion)(k))

else:

return [NUDEdistanceAccount\_1](#NUDEdistanceAccount_1)(k-1)

#

@

NUDEdistanceAccount\_2，监控当司机未授权向**END\_2**方向运行时，列车向**END\_2**方向运行的距离，该值为非正数，若在初始化阶段，或NUDE2为**True**，或已经EB并停车，则等于0；否则，当测速无效时，将其设置为默认值；否则，当里程计已经初始化后：若 [MaximumTrainMotion](#MaximumTrainMotion)小于0，则等于上周期累加距离加上本周期最大位移，最大取0。若[MaximumTrainMotion](#MaximumTrainMotion)大于等于0，则使用上周期值加最小位移，最大取0。否则，保持累加距离不变。

$

When the driver does not authorize the train running toward the **END\_2**, ATP shall accumulate the distance of the train running toward to the **END\_2**.If in initialization, or the [NoUndetectableDanger\_2](#NoUndetectableDanger_2) is **True**, or the train has triggered EB and has stopped, ATP set this distance to 0;Else if train kinematic has invalid, ATP set this distance to the default value.Else if the odometer has initialized:if the [MaximumTrainMotion](#MaximumTrainMotion) is less than 0, ATP accumulate the maximum movement in this cycle with the distance of last cycle;Else: if the [MaximumTrainMotion](#MaximumTrainMotion) is greater than or equal to 0, ATP use the minimum movement of this cycle plus to the distance last cycle (in fact, decrease the accumulated distance). The minimum of this accumulated distance is 0.Otherwise, keep the distance unchanged.

%

def [NUDEdistanceAccount\_2](#NUDEdistanceAccount_2)(k):

if (INTIALIZATION

or [NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k)

or ([EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE)(k-1) and [TrainFilteredStopped](#TrainFilteredStopped)(k)))

return 0

elif ([ValidTrainKinematic](#ValidTrainKinematic)(k) != **True**)

return -1 \* [ATPsetting](#ATPsetting).NUDEdistanceWithoutMotionAvailable(k)

elif ([OdometerState](#OdometerState)(k) is **INITIALIZED**)

if ([End2RunningForward](#End2RunningForward)(k))

return [min](#min)(0, [NUDEdistanceAccount\_2](#NUDEdistanceAccount_2)(k-1)+ [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [min](#min)(0, [NUDEdistanceAccount\_2](#NUDEdistanceAccount_2)(k-1)+ [MinimumTrainMotion](#MinimumTrainMotion)(k))

else:

return [NUDEdistanceAccount\_2](#NUDEdistanceAccount_2)(k-1)

#

@

UndetectDangerMotionWithoutNUDE，列车运行超过限定距离，但仍有车头未检测到NUDE。

$

When the train has moved without NUDE more than project-restricted distance, ATP shall set this value to **True**.

%

[UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE)(k)

= (not [NoUndetectableDanger\_1](#NoUndetectableDanger_1)(k)

and ([NUDEdistanceAccount\_1](#NUDEdistanceAccount_1)(k)> [ATPsetting](#ATPsetting).NUDElimitDistance))

or (not [NoUndetectableDanger\_2](#NoUndetectableDanger_2)(k)

and ([NUDEdistanceAccount\_2](#NUDEdistanceAccount_2)(k)< -1 \* [ATPsetting](#ATPsetting).NUDElimitDistance)))

#

@

EBappliedForMotionWithoutNUDE，保证由NUDE导致的EB会延迟一段时间。即：当[UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE)为**True**时，设置EBappliedForMotionWithoutNUDE为**True**；当[UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE)由**True**变为**False**后，还需保持EBappliedForMotionWithoutNUDE 在[ATPsetting](#ATPsetting).NUDEtrainStopDurationBeforeEBrelease时间内为**True**；超过上述时间后，该值为**False**。

$

The EB request shall be maintained to **True** during the application time [ATPsetting](#ATPsetting).NUDEtrainStopDurationBeforeEBrelease, if the train has moved without NUDE more than project restricted distance. When [UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE) is **True**, ATP shall set [EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE) to **True**;When [UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE) change from **True** to **False**, ATP shall

maintain [EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE) to **True** in period [ATPsetting](#ATPsetting).NUDEtrainStopDurationBeforeEBrelease；Over the time, set this value to **False**.

%

def [EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE)(k):

if ([UndetectDangerMotionWithoutNUDE](#UndetectDangerMotionWithoutNUDE)(k)):

NudeEBreleaseCounter = 0

return **True**

elif (NudeEBreleaseCounter < [ATPsettings](#ATPsettings).NUDEtrainStopDurationBeforeEBrelease):

NudeEBreleaseCounter = NudeEBreleaseCounter(k-1) + 1

return **True**

else:

return **False**

#

@

EBforUndetectableDangerRisk，由“无法侦测的危险”导致EB并停车后，ATP应当根据项目配置判断是否输出EB。

$

When the train has triggered emergency brake causing by the "undetectable danger risk" and has stopped, ATP shall determine whether keeping the EB output according to the project configuration.

%

[EBforUndetectableDangerRisk](#EBforUndetectableDangerRisk)(k)

= ([EBappliedForMotionWithoutNUDE](#EBappliedForMotionWithoutNUDE)(k)

or ([UndetectableDangerRiskForNoNUDE](#UndetectableDangerRiskForNoNUDE)(k)

and (not [TrainFilteredStopped](#TrainFilteredStopped)(k)

or ([ATPsetting](#ATPsetting).NUDEimmoBehaviourAtFS == **IB\_APPLY\_EMERGENCY\_BRAKE**)

or (([ATPsetting](#ATPsetting).NUDEimmoBehaviourAtFS == **IB\_APPLY\_EMERGENCY\_BRAKE\_WHEN\_TRIGGERED**)

and (not [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1))))))

#

@

ConditionForRMlimitSpeed，当前应用哪种RM限速。ATP最多支持项目配置**MAX\_RM\_CONDITION\_NB**种RM限速。

%

def [ConditionForRMlimitSpeed](#ConditionForRMlimitSpeed)(i, k):

return [Offline.GetConditionForRMlimitSpeed](#GetConditionForRMlimitSpeed)(i, k)

#

@

RMlimitSpeedApplied，根据列车输入，判断当前应当监控的RM限速

%

def [RMlimitSpeedApplied](#RMlimitSpeedApplied)(k):

for i in [range](#range)(0, **MAX\_RM\_CONDITION\_NB**):

if ([ConditionForRMlimitSpeed](#ConditionForRMlimitSpeed)(i, k)):

return [ATPsetting](#ATPsetting).MPinhibitionLimitSpeed[i]

else:

continue

return 0

#

@

NoDangerForRMoverSpeed，列车速度小于等于RM模式下的限速。

$

ATP estimates that current train maximum speed not exceeds the RM limit speed.

%

def [NoDangerForRMoverSpeed](#NoDangerForRMoverSpeed)(k):

return ([ValidTrainKinematic](#ValidTrainKinematic)(k)

and [TrainMaxSpeed](#TrainMaxSpeed)(k) <= [RMlimitSpeedApplied](#RMlimitSpeedApplied)(k))

#

@

EBforRMoverSpeed，若在RM模式下，列车速度大于RM模式限速，则将输出EB。

%

def [EBforRMoverSpeed](#EBforRMoverSpeed)(k):

return (not [NoDangerforRMoverSpeed](#NoDangerforRMoverSpeed)(k)

and [MotionProtectionInhibition](#MotionProtectionInhibition)(k))

#

@

NoDangerForMemorizedLocationOverSpeed，在使用记忆定位而还未读到确认信标时，ATP监控列车速度是否超过项目限制值。

%

def [NoDangerForMemorizedLocationOverSpeed](#NoDangerForMemorizedLocationOverSpeed)(k):

return (not [MemLocationNotConfirmed](#MemLocationNotConfirmed)(k)

or [[TrainMaxSpeed](#TrainMaxSpeed)](#MaximumTrainSpeed) < [ATPsetting](#ATPsetting).MemLocLimitSpeed)

#

@

EBforMemorizedLocationOverSpeed，在使用记忆定位而还未读到确认信标时，ATP应确保列车速度不超过项目限制值。

%

def [EBforMemorizedLocationOverSpeed](#EBforMemorizedLocationOverSpeed)(k):

return (not [NoDangerforMemorizedLocationOverSpeed](#NoDangerforMemorizedLocationOverSpeed)(k)

and not [MotionProtectionInhibition](#MotionProtectionInhibition)(k))

#

@

RMRselectedDrivingMode，是否选择了RMR倒车模式。其状态来自于项目可配置的列车输入采集。

$

[RMRselectedDrivingMode](#RMRselectedDrivingMode) represents the choice of RMR.

%

def [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k):

return [Offline](#Offline).[GetRMRselectedDrivingMode](#GetRMRselectedDrivingMode)(k)

#

@

RollbackDistanceAccount\_1，累计回溜的距离（负值表示在回溜）：初始化时设置该值为0；否则，如果列车运动学无效，则设置为配置参数的默认值；否则，在**END\_1**激活且未选择RMR模式的前提下：若里程计已初始化，且列车向**END\_1**方向运行，则累加最小位移，若超过0则取0，否则是一个负值。否则，若里程计齿数齿号匹配，则累加列车最大位移; 否则，即里程计未初始化，则保持累计距离不变。其他情况，保持累计距离不变。

$

When train front extremity is **END\_1** and traction effort is supposed to be in the direction of travel, [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1) is the estimated maximum distance which separates current front extremity 1 position to last most forward position reached by this extremity. ATP shall evaluate [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1) in order to control that speed does not exceed [ATPsetting](#ATPsetting).MPnotAuthLimitSpeed .

%

def [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k):

if (Initialization)

return 0

elif (not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

return (-1 \* [ATPsetting](#ATPsetting).MPnotAuthDistWithoutMotionAvailable)

elif ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**

and not [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

if ([End1RunningForward](#End1RunningForward)(k)):

return [min](#min)(0, [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k-1) + [MinimumTrainMotion](#MinimumTrainMotion)(k))

else:

return ([RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k-1) + [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k-1)

else:

return [RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k-1)

#

@

RollbackDistanceAccount\_2，累计回溜的距离（负值表示在回溜）：初始化时设置该值为0；否则，如果列车运动学无效，则设置为配置参数的默认值；否则，在**END\_2**激活且未选择RMR模式的前提下：若里程计已初始化，且列车向**END\_2**方向运行，则减去最小位移，若超过0则取0，否则是一个负值。否则，若里程计已初始化，则减去列车最大位移;否则，即里程计还未初始化，则保持累计距离不变。其他情况，保持累计距离不变。

$

When train front extremity is **END\_2** and traction effort is supposed to be in the direction of travel, [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2) is the estimated maximum distance which separates current front extremity 2 position to last most forward position reached by this extremity. ATP shall evaluate [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2) in order to control that speed does not exceed [ATPsetting](#ATPsetting).MPnotAuthLimitSpeed.

%

def [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k):

if (Initialization)

return 0

elif (not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

return (-1 \* [ATPsetting](#ATPsetting).MPnotAuthDistWithoutMotionAvailable)

elif ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**

and not [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

if ([End2RunningForward](#End2RunningForward)(k)):

return [min](#min)(0, [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k-1) - [MinimumTrainMotion](#MinimumTrainMotion)(k))

else:

return ([RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k-1) — [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k-1)

else:

return [RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k-1)

#

@

UnrecoverableRollbackOverSpeed，如果ATP检测到列车已经回退超过项目限制的最大距离，则设置永久回退超速

$

From ATP power-up, [UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed) shall initialize to **False**. [UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed) shall be set to **True** if and only if following conditions are fulfilled:driving selector indicates that traction effort is supposed to be in the direction of travel,train front extremity is **END\_2** or **END\_1**,and rollback limit speed currently applicable is null for this direction of travel.Once [UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed) set as **True**, it shall stay at state **True** while ATP is not reboot.

%

def [UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed)(k):

return ([UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed)(k-1)

or (not [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)

and (([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**

and ([abs](#abs)([RollbackDistanceAccount\_2](#RollbackDistanceAccount_2)(k))

> [ATPsetting](#ATPsetting).MPnotAuthLimitDistance))

or ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**

and ([abs](#abs)([RollbackDistanceAccount\_1](#RollbackDistanceAccount_1)(k))

> [ATPsetting](#ATPsetting).MPnotAuthLimitDistance)))))

#

@

RollbackOverSpeed，下列任一条件满足，认为回退超速若车头2激活，位移为**END\_1**方向，未选择RMR模式;车速大于当前回退距离所在限速;若车头1激活，位移为**END\_2**方向，未选择RMR模式;车速大于当前回退距离所在限速;列车运动学无效;已发生了永久回退超速错误

$

[RollbackOverSpeed](#RollbackOverSpeed) shall be **True** if and only if following conditions are fulfilled:driving selector indicates that traction effort is supposed to be in the direction of travel,train front extremity is **END\_2** or **END\_1**,and movement observed is in the opposite direction of travel, and over-estimated train speed is greater than [ATPsetting](#ATPsetting).MPnotAuthLimitSpeed currently applicable for this direction of travel and rollback speed restrictions is not null.Or: train has reached a position due a rollback movement which is unrecoverable,Or: train kinematic is invalid,

%

def [RollbackOverSpeed](#RollbackOverSpeed)(k):

return (not [ValidTrainKinematic](#ValidTrainKinematic)(k)

or [UnrecoverableRollbackOverSpeed](#UnrecoverableRollbackOverSpeed)(k)

or (not [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)

and ([TrainMaxSpeed](#TrainMaxSpeed)(k) >= [ATPsetting](#ATPsetting).MPnotAuthLimitSpeed

and (([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**

and [End1RunningForward](#End1RunningForward)(k) and not [End2RunningForward](#End2RunningForward)(k))

or ([TrainFrontEnd](#TrainFrontEnd) is **END\_1**

and [End2RunningForward](#End2RunningForward)(k) and not [End1RunningForward](#End1RunningForward)(k))))))

#

@

EBforRollbackOverSpeed，如果ATP检测到回溜超速，则输出EB

$

ATP shall request emergency braking if a reverse speed limit is over-run for unwilling rollback or excessive reverse motion.

%

[EBforRollbackOverSpeed](#EBforRollbackOverSpeed) = [RollbackOverSpeed](#RollbackOverSpeed)(k)

#

@

LongDistanceReverseAuthorized，长距离倒车模式是否授权，其状态来自于项目可配置的列车输入采集。

$

[LongDistanceReverseAuthorized](#LongDistanceReverseAuthorized) represents the authorization of long distance reverse.

%

def [LongDistanceReverseAuthorized](#LongDistanceReverseAuthorized)(k):

return [Offline.GetLongDistanceReverseAuthorized](#GetLongDistanceReverseAuthorized)(k)

#

@

ReverseDistanceAccount\_1，累加RMR模式下的倒车距离（负值表示倒车）：

初始化时设置该值为0；否则，如果列车运动学无效，则设置为配置参数的默认值；否则，在**END\_1**激活且非长距离倒车授权的前提下：若里程计已初始化，且列车向**END\_1**方向运行，则减小倒车距离绝对值，大于零则等于0。否则，如果里程计已初始化，且选择RMR模式，则累加倒车距离。否则，即里程计还未初始化，则保持距离不变。其他情况，保持累计距离不变。

$

When train front extremity is **END\_1** and traction effort is supposed to be in the opposite direction of travel, [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1) is the estimated maximum distance which separates current front extremity 1 position to last most forward position reached by this extremity. ATP shall evaluate [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1) in order to control that speed does not exceed reverse speed limit function.

%

def [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k):

if (Initialization)

return 0

elif (not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

return [ATPsetting](#ATPsetting).ReverseDistWithoutMotionAvailable

elif ([TrainFrontEnd](#TrainFrontEnd) is **END\_1**

and not [LongDistanceReverseAuthorized](#LongDistanceReverseAuthorized)(k)):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

if ([End1RunningForward](#End1RunningForward)(k)):

return [min](#min)(0, [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k-1) + [MinimumTrainMotion](#MinimumTrainMotion)(k))

elif ([RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)):

return ([ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k-1) + [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k-1)

else:

return [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k-1)

else:

return [ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k-1)

#

@

ReverseDistanceAccount\_2，累加RMR模式下的倒车距离（负值表示倒车）：初始化时设置该值为0；否则，如果列车运动学无效，则设置为配置参数的默认值；否则，在**END\_2**激活且非长距离倒车授权的前提下：如果里程计已初始化，且列车向**END\_2**方向运行，则减小倒车距离绝对值，大于零则等于0。否则，如果里程计已初始化，且选择RMR模式时累加倒车距离。否则，即里程计未初始化，则保持距离不变；其他情况，保持累计距离不变。

$

When train front extremity is **END\_2** and traction effort is supposed to be in the opposite direction of travel, [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2) is the estimated maximum distance which separates current front extremity 2 position to last most forward position reached by this extremity. ATP shall evaluate [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2) in order to control that speed does not exceed [ReverseSpeedRestrictions](#ReverseSpeedRestrictions) reverse speed limit function.

%

def [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k):

if (Initialization)

return 0

elif (not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

return [ATPsetting](#ATPsetting).ReverseDistWithoutMotionAvailable

elif ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**

and not [LongDistanceReverseAuthorized](#LongDistanceReverseAuthorized)(k)):

if ([OdometerState](#OdometerState)(k) is **INITIALIZED**):

if ([End2RunningForward](#End2RunningForward)(k))

return [min](#min)(0, [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k-1) - [MinimumTrainMotion](#MinimumTrainMotion)(k))

elif ([RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)):

return ([ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k-1) — [MaximumTrainMotion](#MaximumTrainMotion)(k))

else:

return [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k-1)

else:

return [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k-1)

else:

return [ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k-1)

#

@

ReverseOverSpeed，超过RMR模式限速的条件：

$

[ReverseOverSpeed](#ReverseOverSpeed) shall be **True** if following conditions fulfilled:driving selector indicates that traction effort is supposed to be in the opposite direction of travel,train front extremity is **END\_2** or **END\_1**,and movement observed is the opposite direction of travel,and:over-estimated train speed is greater than reverse speed restrictions currently applicable for this direction of travel,or else: if reverse speed restrictions currently applicable is null for this direction of travel,Or else: train kinematic is invalid.

%

def [ReverseOverSpeed](#ReverseOverSpeed)(k):

if (not [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)

or [LongDistanceReverseAuthorized](#LongDistanceReverseAuthorized)(k)):

return **False**

elif (not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

return **True**

else:

return (([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_2**):

and (([End1RunningForward](#End1RunningForward)(k) and not [End2RunningForward](#End2RunningForward)(k)

and ([TrainMaxSpeed](#TrainMaxSpeed)(k)

> [ReverseSpeedRestrictions](#ReverseSpeedRestrictions)([ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k))))

or ([ReverseSpeedRestrictions](#ReverseSpeedRestrictions)([ReverseDistanceAccount\_2](#ReverseDistanceAccount_2)(k)) == 0)))

or ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**

and (([End2RunningForward](#End2RunningForward)(k) and not [End1RunningForward](#End1RunningForward)(k)

and ([TrainMaxSpeed](#TrainMaxSpeed)(k) > [ReverseSpeedRestrictions](#ReverseSpeedRestrictions)([ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k))))

or ([ReverseSpeedRestrictions](#ReverseSpeedRestrictions)([ReverseDistanceAccount\_1](#ReverseDistanceAccount_1)(k)) == 0))))

#

@

EBforReverseOverSpeed，由于RMR下倒车超速而导致EB

$

ATP shall request emergency braking if a reverse speed limit is over-run for unwilling rollback or excessive reverse motion.

%

[EBforReverseOverSpeed](#EBforReverseOverSpeed) = [ReverseOverSpeed](#ReverseOverSpeed)(k)

#

@

TrainEmergencyBrakeApplied，列车是否施加了紧急制动。其状态来自于项目可配置的列车输入采集。

$

[TrainEmergencyBrakeApplied](#TrainEmergencyBrakeApplied) shows that whether the train has applied emergency brake. If the project is not configured, ATP shall consider the emergency brake has not applied by the train. Otherwise, if either of the end is in emergency brake, ATP considers the emergency brake has applied.

%

def [TrainEmergencyBrakeApplied](#TrainEmergencyBrakeApplied)(k):

return [Offline.GetTrainEmergencyBrakeApplied](#GetTrainEmergencyBrakeApplied)(k)

#

@

TrainParkingBrakeApplied，任一端车头已施加停车制动，则认为停车制动已施加。其状态来自于项目可配置的列车输入采集。

$

The term [TrainParkingBrakeApplied](#TrainParkingBrakeApplied) stands for that either of the train ends is in parking brake.

%

def [TrainParkingBrakeApplied](#TrainParkingBrakeApplied)(k):

return [Offline.GetTrainParkingBrakeApplied](#GetTrainParkingBrakeApplied)(k)

#

@

TrainSafelyImmobilised，判断是否已经安全停车

$

ATP shall consider that train safely immobilized if:Train brake has detected safely applied, or train parking brake is detected;And train is detected at filtered stop.

%

def [TrainSafelyImmobilised](#TrainSafelyImmobilised)(k):

return (([TrainEmergencyBrakeApplied](#TrainEmergencyBrakeApplied)(k)

or [TrainParkingBrakeApplied](#TrainParkingBrakeApplied)(k))

and [TrainFilteredStopped](#TrainFilteredStopped)(k))

#

@

NoVitalCorrectlyDocked，CCNV判断列车是否停在开门授权区内

%

def [NoVitalCorrectlyDocked](#NoVitalCorrectlyDocked)(k):

return ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest)(k).TrainInCorrectlyDockedZone)

#

@

LocalATPenableDoorOpening\_A，本ATP是否在站内允许开A侧车门：

$

ATP shall determine whether train doors on side A opening. The rules are following:

%

def [LocalATPenableDoorOpening\_A](#LocalATPenableDoorOpening_A)(k):

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and [TrainIncludedInVPEZ\_A](#TrainIncludedInVPEZ_A)(k)

and [NoVitalCorrectlyDocked](#NoVitalCorrectlyDocked)(k)

and [TrainSafelyImmobilised](#TrainSafelyImmobilised)(k))

#

@

LocalATPenableDoorOpening\_B，本ATP是否授权开启B侧车门：

$

ATP shall determine whether train doors on side B opening. The rules are following:

%

def [LocalATPenableDoorOpening\_B](#LocalATPenableDoorOpening_B)(k):

return ([TrainLocatedOnKnownPath](#TrainLocatedOnKnownPath)(k)

and [TrainIncludedInVPEZ\_B](#TrainIncludedInVPEZ_B)(k)

and [NoVitalCorrectlyDocked](#NoVitalCorrectlyDocked)(k)

and [TrainSafelyImmobilised](#TrainSafelyImmobilised)(k))

#

@

EnableDoorOpening\_A，结合远端ATP结果的开门授权信息。

%

def [EnableDoorOpening\_A](#EnableDoorOpening_A)(k):

return ([LocalATPenableDoorOpening\_A](#LocalATPenableDoorOpening_A)(k)

or [OtherATP](#OtherATP).EnableDoorOpening\_A)

#

@

EnableDoorOpening\_B，结合远端ATP结果的开门授权信息。

%

def [EnableDoorOpening\_B](#EnableDoorOpening_B)(k):

return ([LocalATPenableDoorOpening\_B](#LocalATPenableDoorOpening_B)(k)

or [OtherATP](#OtherATP)(k).EnableDoorOpening\_B)

#

@

PSDoperation\_A和PSDoperation\_B，其结构为ST\_PSD\_OPERATION，用于获取来自CCNV的屏蔽门控制指令。

$

[PSDoperation\_A](#PSDoperation_A) and [PSDoperation\_B](#PSDoperation_B) structured as ST\_PSD\_OPERATION, used to obtain the PSD controlling order from CCNV.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[PSDoperation\_A](#PSDoperation_A)(k)= [NonVitalRequest](#NonVitalRequest).[PSDoperation\_A](#PSDoperation_A)(k)

[PSDoperation\_B](#PSDoperation_B)(k)= [NonVitalRequest](#NonVitalRequest).[PSDoperation\_B](#PSDoperation_B)(k)

else:

[PSDoperation\_A](#PSDoperation_A)(k).Id = **None**

[PSDoperation\_B](#PSDoperation_B)(k).Id = **None**

#

@

CommunicateWithPSD，ATP根据CCNV的请求，判断是否与联锁建立通信。当本周期来自CCNV的[PSDoperation\_A](#PSDoperation_A)或[PSDoperation\_B](#PSDoperation_B)不全为**None**时，设置CommunicateWithPSD为**True**；否则，设置CommunicateWithPSD为**False**。

$

ATP shall determine whether to establish communication with the correlative CI according to request from CCNV:When there is at least one id of PSDoperation\_A or PSDoperation\_B is not none, ATP shall set CommunicatedWithPSD to **True**:Otherwise, set CommunicatedWithPSD to **False**.

%

def [CommunicateWithPSD](#CommunicateWithPSD)(k):

return ([PSDoperation\_A](#PSDoperation_A)(k).Id is not **None**

or [PSDoperation\_B](#PSDoperation_B)(k).Id is not **None**)

#

@

UsingPSDstatusFromCI，只有当列车定位与PSD区域有交集，且列车静止或刚发车时，ATP使用来自CI的PSD状态信息。

$

Only when the train fulfilled the following conditions, ATP shall use the PSD status from the CI:The train location intersects with a PSD zone;And the train is filtered stopped or just started moving.

%

[UsingPSDstatusFromCI](#UsingPSDstatusFromCI) = (([AlignPSDzone\_A](#AlignPSDzone_A)(k) or [AlignPSDzone\_B](#AlignPSDzone_B)(k))

and ([TrainFilteredStopped](#TrainFilteredStopped)(k)

or ([TrainFilteredStopped](#TrainFilteredStopped)(k-1)

and not [TrainFilteredStopped](#TrainFilteredStopped)(k))))

#

@

MasterCCcore，来自CCNV的当前是否为主控CC信息

$

[MasterCCcore](#MasterCCcore) shows whether the status from CCNV is the main controlled CC.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[MasterCCcore](#MasterCCcore) = [NonVitalRequest](#NonVitalRequest).MasterCcCore(k)

else:

[MasterCCcore](#MasterCCcore) = **False**

#

@

[PSDzoneStatus\_A](#PSDzoneStatus_A), 如果ATP所在为主控CC，则对A侧PSD状态的更新规则如下：

$

If the ATP is the master CC, then the A-side PSD state updating rules are as follows:

%

if ([MasterCCcore](#MasterCCcore)(k) == **True**)

if (([PSDoperation\_A](#PSDoperation_A).Id == **None**) or ([PSDoperation\_A](#PSDoperation_A).Id != [PSDid\_A](#PSDid_A)(k)))

[PSDzoneStatus\_A](#PSDzoneStatus_A).Id(k)= **None**

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = **0**

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed = **False**

else:

[PSDzoneStatus\_A](#PSDzoneStatus_A)(k).Id = [PSDoperation\_A](#PSDoperation_A)(k).Id

if ([UsingPSDstatusFromCI](#UsingPSDstatusFromCI)(k))

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k) = [TableOfPSDPlatform](#TableOfPSDPlatform)[[PSDid\_A](#PSDid_A)(k)].DoorStatusValidityTime

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k) = [TableOfPSDPlatform](#TableOfPSDPlatform)[[PSDid\_A](#PSDid_A)(k)].DoorClosed

elif ([CoercedPermissive](#CoercedPermissive)([TrackMap](#TrackMap).PSDs[[PSDid\_A](#PSDid_A)(k)].CoercedPermissive, k))

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = **REPORT\_AGE\_MAX**

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed = **True**

elif ([CoercedRestrictive](#CoercedRestrictive)([TrackMap](#TrackMap).PSDs[[PSDid\_A](#PSDid_A)(k)].NotCoercedRestrictive, k))

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = **REPORT\_AGE\_MAX**

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed = **False**

else:

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k) = [ReceivedVariantReport](#ReceivedVariantReport)[LineSectionOfPSD].ValidityTime

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k)= [VariantValue](#VariantValue)([TrackMap](#TrackMap).PSDs[[PSDid\_A](#PSDid_A)(k)].Variant, k)

#

@

[PSDzoneStatus\_B](#PSDzoneStatus_B), 如果ATP所在为主控CC，则对B侧PSD状态的更新规则如下：

$

If the ATP is the master CC, then the B-side PSD state updating rules are as follows:

%

if ([MasterCCcore](#MasterCCcore)(k) == **True**)

if (([PSDoperation\_B](#PSDoperation_B).Id == **None**) or ([PSDoperation\_B](#PSDoperation_B).Id != [PSDid\_B](#PSDid_B)(k)))

[PSDzoneStatus\_B](#PSDzoneStatus_B).Id(k)= **None**

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity = 0

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed = **False**

else:

[PSDzoneStatus\_B](#PSDzoneStatus_B).Id(k)= [PSDoperation\_B](#PSDoperation_B).Id

if ([UsingPSDstatusFromCI](#UsingPSDstatusFromCI)(k))

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k) = [TableOfPSDPlatform](#TableOfPSDPlatform)[[PSDid\_B](#PSDid_B)(k)].DoorStatusValidityTime

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k) = [TableOfPSDPlatform](#TableOfPSDPlatform)[[PSDid\_B](#PSDid_B)(k)].DoorClosed

elif ([CoercedPermissive](#CoercedPermissive)([TrackMap](#TrackMap).PSDs[[PSDid\_B](#PSDid_B)(k)].CoercedPermissive, k))

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity = **REPORT\_AGE\_MAX**

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed = **True**

elif ([CoercedRestrictive](#CoercedRestrictive)([TrackMap](#TrackMap).PSDs[[PSDid\_B](#PSDid_B)(k)].NotCoercedRestrictive, k))

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity =  **REPORT\_AGE\_MAX**

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed = **False**

else:

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k) = [ReceivedVariantReport](#ReceivedVariantReport)[LineSectionOfPSD].ValidityTime

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k)= [VariantValue](#VariantValue)([TrackMap](#TrackMap).PSDs[[PSDid\_B](#PSDid_B)(k)].Variant, k)

#

@

如果ATP所在为备机CC，则对A侧PSD状态的更新规则如下：

$

If the ATP is not the master CC, then the A-side PSD state updating rules are as follows:

%

if ([MasterCCcore](#MasterCCcore) != **True**)

if ([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

[PSDzoneStatus\_A](#PSDzoneStatus_A).Id = [OtherATP](#OtherATP).PsdIdSide\_A

if ([PSDzoneStatus\_A](#PSDzoneStatus_A).Id(k) != **None**)

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k) = ([OtherATP](#OtherATP).PsdValiditySide\_A

- [Message.ModularSub](#ModularSub)([ATPtime](#ATPtime)(k), [OtherATP](#OtherATP).LatestTimeOtherCore))

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k) = ([OtherATP](#OtherATP).PsdClosedSide\_A

and ([PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k) > 0))

else:

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = 0

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed = **False**

elif ([PSDzoneStatus\_A](#PSDzoneStatus_A).Id(k-1) != **None**)

[PSDzoneStatus\_A](#PSDzoneStatus_A).Id = [PSDzoneStatus\_A](#PSDzoneStatus_A).Id(k-1)

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = [PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k-1)- 1

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k) = ([PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k-1)

and ([PSDzoneStatus\_A](#PSDzoneStatus_A).Validity(k) > 0))

else:

[PSDzoneStatus\_A](#PSDzoneStatus_A).Validity = 0

[PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed = **False**

#

@

如果ATP所在为备机CC，则对B侧PSD状态的更新规则如下：

$

If the ATP is not the master CC, then the B-side PSD state updating rules are as follows:

%

if ([MasterCCcore](#MasterCCcore) != **True**)

if ([OtherATPmessageAvailable](#OtherATPmessageAvailable)(k) == **True**)

[PSDzoneStatus\_B](#PSDzoneStatus_B).Id = [OtherATP](#OtherATP).PsdIdSide\_B

if ([PSDzoneStatus\_B](#PSDzoneStatus_B).Id(k) != **None**)

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k) = ([OtherATP](#OtherATP).PsdValiditySide\_B

- [Message.ModularSub](#ModularSub)([ATPtime](#ATPtime)(k), [OtherATP](#OtherATP).LatestTimeOtherCore))

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k)

= [OtherATP](#OtherATP).PsdClosedSide\_B

and ([PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k) > 0)

else:

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity = 0

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed = **False**

elif ([PSDzoneStatus\_B](#PSDzoneStatus_B).Id(k-1) != **None**)

[PSDzoneStatus\_B](#PSDzoneStatus_B).Id = [PSDzoneStatus\_B](#PSDzoneStatus_B).Id(k-1)

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity = [PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k-1) - 1

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k)

= [PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k-1)

and ([PSDzoneStatus\_B](#PSDzoneStatus_B).Validity(k) > 0)

else:

[PSDzoneStatus\_B](#PSDzoneStatus_B).Validity = 0

[PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed = **False**

#

@

PSDstatusNonVital\_A，用于CCNV发送给DMI显示的A侧PSD状态

%

def [PSDstatusNonVital\_A](#PSDstatusNonVital_A)(k):

if ([PSDzoneStatus\_A](#PSDzoneStatus_A)(k).Id is **None**

or [PSDzoneStatus\_A](#PSDzoneStatus_A)(k).Validity <= 0):

return **PSD\_STATE\_UNKNOWN**

elif ([PSDzoneStatus\_A](#PSDzoneStatus_A)(k).AllPSDclosed):

return **PSD\_STATE\_CLOSED**

else:

return **PSD\_STATE\_OPENED**

#

@

PSDstatusNonVital\_B，用于CCNV发送给DMI显示的B侧PSD状态

%

def [PSDstatusNonVital\_B](#PSDstatusNonVital_B)(k):

if ([PSDzoneStatus\_B](#PSDzoneStatus_B)(k).Id is **None**

or [PSDzoneStatus\_B](#PSDzoneStatus_B)(k).Validity <= 0):

return **PSD\_STATE\_UNKNOWN**

elif ([PSDzoneStatus\_B](#PSDzoneStatus_B)(k).AllPSDclosed):

return **PSD\_STATE\_CLOSED**

else:

return **PSD\_STATE\_OPENED**

#

@

PSDmanagerOrder\_A，A侧PSD的控制命令信息，其结构为ST\_PSD\_MANAGE。其中如果来自CCNV的A侧PSD标识不等于ATP读取SGD中A侧的标识，则禁止使用CCNV的标识开门。

$

The rules to generate the PSD manage order on side A shall follow the pseudo-codes. In which if the PSD id from CCNV is not equal to the id in ATP's track map, ATP shall prohibit the PSD opening.

%

def [PSDmanagerOrder\_A](#PSDmanagerOrder_A)(k):

[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id = [PSDoperation\_A](#PSDoperation_A).Id(k)

if ([PSDoperation\_A](#PSDoperation_A).Id(k) == [PSDid\_A](#PSDid_A)(k)

and [PSDoperation\_A](#PSDoperation_A).Id(k) is not **None**

and not [PSDoperation\_A](#PSDoperation_A).ClosingOrder(k)

and [PSDoperation\_A](#PSDoperation_A).OpeningOrder(k)

and [EnableDoorOpening\_A](#EnableDoorOpening_A)(k)):

[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order = Open\_PSD\_Configuration

elif (not [PSDoperation\_A](#PSDoperation_A).OpeningOrder(k)

and [PSDoperation\_A](#PSDoperation_A).ClosingOrder(k)):

[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order = Close\_PSD\_Of\_Platform

else:

[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order = **None**

return [PSDmanagerOrder\_A](#PSDmanagerOrder_A)

#

@

PSDmanagerOrder\_B，B侧PSD的控制命令信息，其结构为ST\_PSD\_MANAGE。其中如果来自CCNV的B侧PSD标识不等于ATP读取SGD中B侧的标识，则禁止使用来自CCNV的标识开门。

$

The rules to generate the PSD manage order on side B shall follow the pseudo-codes. In which if the PSD id from CCNV is not equal to the id in ATP's track map, ATP shall prohibit the PSD opening.

%

def [PSDmanagerOrder\_B](#PSDmanagerOrder_B)(k):

[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id = [PSDoperation\_B](#PSDoperation_B).Id(k)

if ([PSDoperation\_B](#PSDoperation_B).Id(k) == [PSDid\_B](#PSDid_B)(k)

and [PSDoperation\_B](#PSDoperation_B).Id(k) is not **None**

and not [PSDoperation\_B](#PSDoperation_B).ClosingOrder(k)

and [PSDoperation\_B](#PSDoperation_B).OpeningOrder(k)

and [EnableDoorOpening\_B](#EnableDoorOpening_B)(k)):

[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order = Open\_PSD\_Configuration

elif (not [PSDoperation\_B](#PSDoperation_B).OpeningOrder(k)

and [PSDoperation\_B](#PSDoperation_B).ClosingOrder(k)):

[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order = Close\_PSD\_Of\_Platform

else:

[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order = **None**

return [PSDmanagerOrder\_B](#PSDmanagerOrder_B)

#

@

PSDplatformManagerOpeningOrder，本ATP是否发了开门命令.

$

ATP shall determine whether itself opening the PSD in this cycle.

%

if (Initialization)

[PSDplatformManagerOpeningOrder](#PSDplatformManagerOpeningOrder) = **False**

elif (([PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order(k) == Open\_PSD\_Configuration)

or ([PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order(k) == Open\_PSD\_Configuration))

[PSDplatformManagerOpeningOrder](#PSDplatformManagerOpeningOrder) = **True**

else:

[PSDplatformManagerOpeningOrder](#PSDplatformManagerOpeningOrder) = **False**

#

@

PSDopeningCommand，本ATP或者冗余端ATP当前是否在发送开PSD命令.

$

ATP shall determine whether itself or the redundant ATP opening the PSD in this cycle.

%

if (([PSDplatformManagerOpeningOrder](#PSDplatformManagerOpeningOrder)(k) == **True**)

or ([OtherATP](#OtherATP).PsdManagerOpeningOrder(k) == **True**))

[PSDopeningCommand](#PSDopeningCommand) = **True**

else:

[PSDopeningCommand](#PSDopeningCommand) = **False**

#

@

在与联锁通信时，如果[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id有效，则根据[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order和离线配置数据设置发送给A侧屏蔽门的控制信息[CIsetting](#CIsetting)[0]。

$

When communicating with the CI, if the [PSDmanagerOrder\_A](#PSDmanagerOrder_A) which comes from CCNV was valid, ATP shall set the [CIsetting](#CIsetting)[0] according to the [PSDmanagerOrder\_A](#PSDmanagerOrder_A) and the configuration of the PSD.

%

if (([CommunicateWithPSD](#CommunicateWithPSD)(k) == **True**)

and ([PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id != **None**))

[CIsetting](#CIsetting)[0].PlatformId = [PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id

if ([PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order == Open\_PSD\_Configuration)

[CIsetting](#CIsetting)[0].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id].DoorOpeningCode

elif ([PSDmanagerOrder\_A](#PSDmanagerOrder_A).Order == Close\_PSD\_Configuration):

[CIsetting](#CIsetting)[0].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id].DoorClosingCode

else:

[CIsetting](#CIsetting)[0].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_A](#PSDmanagerOrder_A).Id].DoorNoActionCode

else:

[CIsetting](#CIsetting)[0].PlatformId = **None**

[CIsetting](#CIsetting)[0].Order = **None**

#

@

在与联锁通信时，如果[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id有效，则根据[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order和离线配置数据设置发送给B侧屏蔽门的控制信息[CIsetting](#CIsetting) [1]。

$

When communicating with the CI, if the [PSDmanagerOrder\_B](#PSDmanagerOrder_B), which comes from CCNV, was valid, ATP shall set the [CIsetting](#CIsetting)[1] according to the [PSDmanagerOrder\_B](#PSDmanagerOrder_B) and the configuration of the PSD.

%

if (([CommunicateWithPSD](#CommunicateWithPSD)(k) == **True**)

and ([PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id != **None**)):

[CIsetting](#CIsetting)[1].PlatformId = [PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id

if ([PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order == Open\_PSD\_Configuration)

[CIsetting](#CIsetting)[1].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id].DoorOpeningCode

elif ([PSDmanagerOrder\_B](#PSDmanagerOrder_B).Order == Close\_PSD\_Configuration):

[CIsetting](#CIsetting)[1].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id].DoorClosingCode

else:

[CIsetting](#CIsetting)[1].Order = [TrackMap](#TrackMap).PSDs[[PSDmanagerOrder\_B](#PSDmanagerOrder_B).Id].DoorNoActionCode

else:

[CIsetting](#CIsetting)[1].PlatformId = **None**

[CIsetting](#CIsetting)[1].Order = **None**

#

@

AllTrainDoorsClosedAndLocked，两端车头有任意一端采到TDCL，即认为两侧车门关闭并锁闭。

$

The [AllTrainDoorsClosedAndLocked](#AllTrainDoorsClosedAndLocked) stands for the condition that either side of both train ends collect TDCL, i.e. both side of door is closed and locked.

%

def [AllTrainDoorsClosedAndLocked](#AllTrainDoorsClosedAndLocked)(k):

return [Offline.GetAllTrainDoorsClosedAndLocked](#GetAllTrainDoorsClosedAndLocked)(k)

#

@

InhibitControlTrainDoorsStatus，不监控车门状态.

$

ATP shall not monitor the status of train doors when

InhibitControlTrainDoorsStatus is selected.

%

def [InhibitControlTrainDoorsStatus](#InhibitControlTrainDoorsStatus)(k):

return [Offline.GetInhibitControlTrainDoorsStatus](#GetInhibitControlTrainDoorsStatus)(k)

#

@

NoDangerForTrainDoorsNotClosedAndLocked，当列车停车，且与PSD区或VPEZ有交集时，且TDCL丢失，则该值为假；否则，该值为真。

%

def [NoDangerForTrainDoorsNotClosedAndLocked](#NoDangerForTrainDoorsNotClosedAndLocked)(k):

return not ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and not [AllTrainDoorsClosedAndLocked](#AllTrainDoorsClosedAndLocked)(k)

and ([AlignPSDzone\_A](#AlignPSDzone_A)(k) or [AlignPSDzone\_B](#AlignPSDzone_B)(k)

or [TrainInterVPEZ\_A](#TrainInterVPEZ_A)(k) or [TrainInterVPEZ\_B](#TrainInterVPEZ_B)(k)))

#

@

PBforTrainDoorsNotClosedAndLocked，列车停车，且车身与PSD区或VPEZ区域有交集时，车门未关时保持PB输出。

$

If the train is aligning in a PSD or intersecting with a vital passage exchange zone, and the RMF or RMR does not selected, ATP shall keep triggering parking brake when the train doors does not closed and locked.

%

def [PBforTrainDoorsNotClosedAndLocked](#PBforTrainDoorsNotClosedAndLocked)(k):

return (not [NoDangerForTrainDoorsNotClosedAndLocked](#NoDangerForTrainDoorsNotClosedAndLocked)(k)

and not [InhibitControlTrainDoorsStatus](#InhibitControlTrainDoorsStatus)(k))

#

@

EBforPBnotAppliedDueToTrainDoors，由于车门开而输出ZVRD，但未检测到ZVBA, 则ATP应当输出EB.

$

If ATP has triggered parking brake for train doors opening, but it does not applied by the rolling stock, ATP shall trigger the emergency brake.

%

[EBforPBnotAppliedDueToTrainDoors](#EBforPBnotAppliedDueToTrainDoors)(k)

= (([PBforTrainDoorsNotClosedAndLocked](#PBforTrainDoorsNotClosedAndLocked)(k) == **True**)

and ([TrainParkingBrakeApplied](#TrainParkingBrakeApplied)(k) != **True**))

#

@

NoDangerForDepartureWithoutTDCL，判断是否未处于上周期停车而本周期开始动车，且车门未关的条件。

$

ATP shall determine whether the train is departure without TDCL.

%

def [NoDangerForDepartureWithoutTDCL](#NoDangerForDepartureWithoutTDCL)(k):

return ([AllTrainDoorsClosedAndLocked](#AllTrainDoorsClosedAndLocked)(k)

or [TrainFilteredStopped](#TrainFilteredStopped)(k)

or not [TrainFilteredStopped](#TrainFilteredStopped)(k-1))

#

@

EBforDepartureWithoutTDCL，若ATP监控发车时丢失TDCL的情况，则输出EB。

$

If ATP needs to monitor the status of train doors, ATP shall trigger EB if train determine without TDCL:

%

def [EBforDepartureWithoutTDCL](#EBforDepartureWithoutTDCL)(k):

return (not [NoDangerForDepartureWithoutTDCL](#NoDangerForDepartureWithoutTDCL)(k)

and not [InhibitControlTrainDoorsStatus](#InhibitControlTrainDoorsStatus)(k))

#

@

InhibitProtectionMovingWithoutTDCL，禁止监控非开门授权情况下车门打开的情形。

$

ATP shall not monitor the status of train doors open without door opening enable if [InhibitProtectionMovingWithoutTDCL](#InhibitProtectionMovingWithoutTDCL) is selected.

%

def [InhibitProtectionMovingWithoutTDCL](#InhibitProtectionMovingWithoutTDCL)(k):

return Offline.[GetInhibitProtectionMovingWithoutTDCL](#GetInhibitProtectionMovingWithoutTDCL)(k)

#

@

NoDangerForMovingWithoutTDCL，监控非授权开门状态下车门打开

%

def [NoDangerForMovingWithoutTDCL](#NoDangerForMovingWithoutTDCL)(k):

return ([AllTrainDoorsClosedAndLocked](#AllTrainDoorsClosedAndLocked)(k)

or [TrainFilteredStopped](#TrainFilteredStopped)(k)

or [EnableDoorOpening\_A](#EnableDoorOpening_A)(k)

or [EnableDoorOpening\_B](#EnableDoorOpening_B)(k))

#

@

EBforMovingWithoutTDCL，禁止监控非开门授权情况下车门打开的情形。

%

def [EBforMovingWithoutTDCL](#EBforMovingWithoutTDCL)(k):

return (not [NoDangerForMovingWithoutTDCL](#NoDangerForMovingWithoutTDCL)(k)

and not [InhibitProtectionMovingWithoutTDCL](#InhibitProtectionMovingWithoutTDCL)(k))

#

@

InhibitControlPSDstatus，项目可配置不监控PSD状态的条件。

$

The conditions ATP does not control PSD can be configured by project.

%

def [InhibitControlPSDstatus](#InhibitControlPSDstatus)(k):

return [Offline.GetInhibitControlPSDstatus](#GetInhibitControlPSDstatus)(k)

#

@

AllPSDclosedAndLocked的判断，上周期或本周期停车，若有PSD且已获取其状态为关闭。

$

If the train stopped or just started moving, and the status of all aligned PSD are closed, ATP shall consider the [AllPSDclosedAndLocked](#AllPSDclosedAndLocked) is **True**.

%

[AllPSDclosedAndLocked](#AllPSDclosedAndLocked)(k)

= (([TrainLocalized](#TrainLocalized)(k) == **True**)

and (([TrainFilteredStopped](#TrainFilteredStopped)(k) == **True**)

or ([TrainFilteredStopped](#TrainFilteredStopped)(k-1) == **True**))

and (([PSDid\_A](#PSDid_A)(k) == 0) and ([PSDid\_B](#PSDid_B)(k) == 0))

or (([PSDid\_A](#PSDid_A)(k) != 0) and ([PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k) == **True**)

and ([PSDid\_B](#PSDid_B)(k) == 0))

or (([PSDid\_B](#PSDid_B)(k) != 0) and ([PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k) == **True**)

and ([PSDid\_A](#PSDid_A)(k) == 0))

or (([PSDid\_A](#PSDid_A)(k) != 0) and ([PSDzoneStatus\_A](#PSDzoneStatus_A).AllPSDclosed(k) == **True**)

and ([PSDid\_B](#PSDid_B)(k) != 0) and ([PSDzoneStatus\_B](#PSDzoneStatus_B).AllPSDclosed(k) == **True**)))

#

@

NoDangerforUnexpectedPSDopening，判断在发车时是否PSD为开门状态.

$

ATP shall determine whether the train starts moving without the aligned PSD closed.

%

def [NoDangerForUnexpectedPSDopening](#NoDangerForUnexpectedPSDopening)(k):

return (not ([AlignPSDzone\_A](#AlignPSDzone_A)(k) or [AlignPSDzone\_B](#AlignPSDzone_B)(k))

or [AllPSDclosedAndLocked](#AllPSDclosedAndLocked)(k)

or [TrainFilteredStopped](#TrainFilteredStopped)(k)

or not [TrainFilteredStopped](#TrainFilteredStopped)(k-1))

#

@

EBforUnexpectedPSDopening，PSD区域内刚发车时PSD门开，则输出EB.

$

If in charge of the PSD control, ATP shall trigger emergency brake when train just started moving but PSD does not closed.

%

def [EBforUnexpectedPSDopening](#EBforUnexpectedPSDopening)(k):

return (not [NoDangerForUnexpectedPSDopening](#NoDangerForUnexpectedPSDopening)(k)

and not [InhibitControlPSDstatus](#InhibitControlPSDstatus)(k))

#

@

NoDangerForPSDnotClosedAndLocked，列车停在PSD区域，且PSD状态为限制时，该值为假；否则，该值为真。

%

def [NoDangerForPSDnotClosedAndLocked](#NoDangerForPSDnotClosedAndLocked)(k):

return not ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and not [AllPSDclosedAndLocked](#AllPSDclosedAndLocked)(k)

and ([AlignPSDzone\_A](#AlignPSDzone_A)(k) or [AlignPSDzone\_B](#AlignPSDzone_B)(k))

#

@

PBforPSDnotClosedAndLocked，车停在PSD区域内，PSD开，且未限制监控该功能时，要求输出ZVRD。

$

If one of the statuses of the aligned PSD does not closed when train stopped, ATP shall trigger parking brake.

%

def [PBforPSDnotClosedAndLocked](#PBforPSDnotClosedAndLocked)(k):

return (not [NoDangerForPSDnotClosedAndLocked](#NoDangerForPSDnotClosedAndLocked)(k)

and not [InhibitControlPSDstatus](#InhibitControlPSDstatus)(k))

#

@

EBforPBnotAppliedDueToPSD，由于PSD开而施加PB，但是未采到ZVBA。

$

If ATP has triggered the parking brake for the PSD opening, but it does not applied by the rolling stock, ATP shall trigger the emergency brake.

%

[EBforPBnotAppliedDueToPSD](#EBforPBnotAppliedDueToPSD)(k)

= (([PBforPSDnotClosedAndLocked](#PBforPSDnotClosedAndLocked)(k) == **True**)

and ([TrainParkingBrakeApplied](#TrainParkingBrakeApplied)(k) != **True**))

#

@

InhibitPSDopeningSupervisedByATP，是否禁止ATP监控发送屏蔽门开启指令时输出PB。

%

def [InhibitPSDopeningSupervisedByATP](#InhibitPSDopeningSupervisedByATP)(k):

return [Offline.GetInhibitPSDopeningSupervisedByATP](#GetInhibitPSDopeningSupervisedByATP)(k)

#

@

PBforPSDopenedAndSupervisedByATP，在PSD开门过程中输出PB

$

If ATP needs to supervise the PSD opening status, ATP shall trigger parking brake when the PSD opening command is valid.

%

def [PBforPSDopenedAndSupervisedByATP](#PBforPSDopenedAndSupervisedByATP)(k):

return ([PSDopeningCommandValid](#PSDopeningCommandValid)(k)

and not [InhibitPSDopeningSupervisedByATP](#InhibitPSDopeningSupervisedByATP))

#

@

EmergencyHandleNotPulledEnd1，**END\_1**逃生门未开。如果该项目未配置驾驶室的逃生门，则认为该逃生门未开。其状态来自于项目可配置的列车输入采集。

$

[EmergencyHandleNotPulledEnd1](#EmergencyHandleNotPulledEnd1) stands for the closed emergency door of **END\_1**. If the train does not allocate with emergency door in the cab, it is certain that the emergency door does not opened.

%

def [EmergencyHandleNotPulledEnd1](#EmergencyHandleNotPulledEnd1)(k):

return [Offline](#Offline).[GetEmergencyHandleNotPulledEnd1](#GetEmergencyHandleNotPulledEnd1)(k)

#

@

EmergencyHandleNotPulledEnd2，End\_2逃生门未开。如果该项目未配置驾驶室的逃生门，则认为该逃生门未开。其状态来自于项目可配置的列车输入采集。

$

[EmergencyHandleNotPulledEnd2](#EmergencyHandleNotPulledEnd2) stands for the closed emergency door of End2. If the train does not allocate with emergency door in the cab, it is certain that the emergency door does not opened.

%

def [EmergencyHandleNotPulledEnd2](#EmergencyHandleNotPulledEnd2)(k):

return [Offline](#Offline).GetEmergencyHandleNotPulledEnd2(k)

#

@

HoldDoorsClosedTrainEnd1，未拉**END\_1**端驾驶室的逃生门紧急手柄，或者车在运动时，锁闭**END\_1**端逃生门。

$

ATP shall keep hold the train **END\_1** door closed when one of the following conditions fulfilled:Train kinematics is valid and the train does not stop;Or the emergency handle of **END\_1** is not pulled;

%

[HoldDoorsClosedTrainEnd1](#HoldDoorsClosedTrainEnd1)(k)

= (([ValidTrainKinematic](#ValidTrainKinematic)(k) and (not [TrainFilteredStopped](#TrainFilteredStopped)(k)))

or [EmergencyHandleNotPulledEnd1](#EmergencyHandleNotPulledEnd1)(k))

#

@

HoldDoorsClosedTrainEnd2，未拉**END\_2**端驾驶室的逃生门紧急手柄，或者车在运动时，锁闭**END\_2**端逃生门。

$

ATP shall keep hold the train **END\_2** door closed when one of the following conditions fulfilled:Train kinematics is valid and the train does not stop;Or the emergency handle of **END\_2** is not pulled;

%

[HoldDoorsClosedTrainEnd2](#HoldDoorsClosedTrainEnd2)(k)

= (([ValidTrainKinematic](#ValidTrainKinematic)(k) and (not [TrainFilteredStopped](#TrainFilteredStopped)(k)))

or [EmergencyHandleNotPulledEnd2](#EmergencyHandleNotPulledEnd2)(k))

#

@

EBforNotAllTrainEndHoldDoorsClosed，驾驶室逃生门手柄拉下.

$

If ATP does not hold the train end door, then trigger emergency brake.

%

[EBforNotAllTrainEndHoldDoorsClosed](#EBforNotAllTrainEndHoldDoorsClosed)(k)

= (not [HoldDoorsClosedTrainEnd1](#HoldDoorsClosedTrainEnd1)(k)

or not [HoldDoorsClosedTrainEnd2](#HoldDoorsClosedTrainEnd2)(k))

#

@

EmergencyDetrainDoorLockingEnd1，要求车辆锁闭End1端驾驶室的紧急逃生门。

%

def [EmergencyDetrainDoorLockingEnd1](#EmergencyDetrainDoorLockingEnd1)(k):

return ([HoldDoorsClosedTrainEnd1](#HoldDoorsClosedTrainEnd1)(k)

or [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1))

#

@

EmergencyDetrainDoorLockingEnd2，要求车辆锁闭End2端驾驶室的紧急逃生门。

%

def [EmergencyDetrainDoorLockingEnd2](#EmergencyDetrainDoorLockingEnd2)(k):

return ([HoldDoorsClosedTrainEnd2](#HoldDoorsClosedTrainEnd2)(k)

or [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1))

#

@

HoldDoorsClosed\_A，A侧车门锁闭.

$

The conditions ATP determining the [HoldDoorsClosed\_A](#HoldDoorsClosed_A) show as following ARDL:

%

if (Initialization

or ([ValidTrainKinematic](#ValidTrainKinematic)(k) != **True**))

[HoldDoorsClosed\_A](#HoldDoorsClosed_A) = **False**

elif ([TrainMaxSpeed](#TrainMaxSpeed)(k) > [ATPsetting](#ATPsetting).DoorTrainLockingSpeed)

[HoldDoorsClosed\_A](#HoldDoorsClosed_A) = **True**

elif ([TrainMaxSpeed](#TrainMaxSpeed)(k) <= [ATPsetting](#ATPsetting).DoorTrainUnlockingSpeed)

[HoldDoorsClosed\_A](#HoldDoorsClosed_A)(k)

= (([EvacuationNotPossible\_A](#EvacuationNotPossible_A)(k) and (not [EvacuationNotPossible\_B](#EvacuationNotPossible_B)(k)))

or ([EvacuationNotPossible\_A](#EvacuationNotPossible_A)(k) and [EvacuationNotPossible\_B](#EvacuationNotPossible_B)(k)

and [ATPsetting](#ATPsetting).EvacuationTrainEnd))

else:

[HoldDoorsClosed\_A](#HoldDoorsClosed_A) = [HoldDoorsClosed\_A](#HoldDoorsClosed_A)(k-1)

#

@

HoldDoorsClosed\_B，B侧车门锁闭

$

The conditions ATP determining the [HoldDoorsClosed\_B](#HoldDoorsClosed_B) show as following ARDL :

%

if (Initialization

or ([ValidTrainKinematic](#ValidTrainKinematic)(k) != **True**))

[HoldDoorsClosed\_B](#HoldDoorsClosed_B) = **False**

elif ([TrainMaxSpeed](#TrainMaxSpeed)(k) > [ATPsetting](#ATPsetting).DoorTrainLockingSpeed)

[HoldDoorsClosed\_B](#HoldDoorsClosed_B) = **True**

elif ([TrainMaxSpeed](#TrainMaxSpeed)(k) <= [ATPsetting](#ATPsetting).DoorTrainUnlockingSpeed)

[HoldDoorsClosed\_B](#HoldDoorsClosed_B)(k)

= (([EvacuationNotPossible\_B](#EvacuationNotPossible_B)(k) and (not [EvacuationNotPossible\_A](#EvacuationNotPossible_A)(k)))

or ([EvacuationNotPossible\_A](#EvacuationNotPossible_A)(k) and [EvacuationNotPossible\_B](#EvacuationNotPossible_B)(k)

and [ATPsetting](#ATPsetting).EvacuationTrainEnd))

else:

[HoldDoorsClosed\_B](#HoldDoorsClosed_B) = [HoldDoorsClosed\_B](#HoldDoorsClosed_B)(k-1)

#

@

TrainDockedInStation，根据开门授权条件判断是否车停在站内。

$

ATP shall determine whether the train has docked in the station correctly according to conditions of train stopping and doors opening enable.

%

def [TrainDockedInStation](#TrainDockedInStation)(k):

return ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and ([EnableDoorOpening\_A](#EnableDoorOpening_A)(k)

or [EnableDoorOpening\_B](#EnableDoorOpening_B)(k)))

#

@

TrainLeavingStation，判断是否在离站过程中。从[TrainDockedInStation](#TrainDockedInStation)由**True**变为**False**开始，如果列车测速有效，累加[MaximumTrainMotion](#MaximumTrainMotion)距离：如果其绝对值在[**0**, [ATPsetting](#ATPsetting).EvacuationStationAreaLength]范围内，则设置TrainLeavingStation为**True**；否则为**False**。即如果列车出站后又倒车回到上述范围内，也应认为是TrainLeavingStation。如果列车运动学无效，则设置该值为**False**并清除累加距离。

$

The train is said to be leaving the station:if since last time train has been detected docked in station ([TrainDockedInStation](#TrainDockedInStation)), the cumulated of the absolute value of [MaximumTrainMotion](#MaximumTrainMotion) is in the range [**0**,[ATPsetting](#ATPsetting).EvacuationStationAreaLength] and no train kinematic invalidation occurs. or else, if the train kinematics is invalid, ATP shall set [TrainLeavingStation](#TrainLeavingStation) as **False** and clear the cumulated distance.

%

def [TrainLeavingStation](#TrainLeavingStation)(k):

if (Initialization

or not [ValidTrainKinematic](#ValidTrainKinematic)(k)):

TrainHasDockedInStation = **False**

LeavingStationDistance = **0**

return **False**

elif ([TrainDockedInStation](#TrainDockedInStation)(k)):

TrainHasDockedInStation = **True**

LeavingStationDistance = **0**

return **False**

elif (not TrainHasDockedInStation(k)):

LeavingStationDistance = **0**

return **False**

else:

LeavingStationDistance = LeavingStationDistance(k-1) + [MaximumTrainMotion](#MaximumTrainMotion)(k)

return ([abs](#abs)(LeavingStationDistance) <= [ATPsetting](#ATPsetting).EvacuationStationAreaLength)

#

@

EmergencyHandleNotPulledSide侧向的紧急手柄未落下。其状态来自于项目可配置的列车输入采集。

$

EmergencyHandleNotPulledSid shows that the emergency handles is not pulled down.

%

def [EmergencyHandleNotPulledSide](#EmergencyHandleNotPulledSide)(k):

return [Offline](#Offline).[GetEmergencyHandleNotPulledSide](#GetEmergencyHandleNotPulledSide)(k)

#

@

EvacuationWhileLeavingStation，未完全出站时丢失车门状态则EB.

$

If the train is just leaving the station and the side doors emergency handles are pulled, ATP shall require [EvacuationWhileLeavingStation](#EvacuationWhileLeavingStation).

%

[EvacuationWhileLeavingStation](#EvacuationWhileLeavingStation)(k)

= (([EmergencyHandleNotPulledSide](#EmergencyHandleNotPulledSide)(k) != **True**)

and ([TrainLeavingStation](#TrainLeavingStation)(k) == **True**)

and ([TrainFilteredStopped](#TrainFilteredStopped)(k) != **True**))

#

@

EvacuationWithTrainStopped，非开门区，停车且乘客紧急手柄拉下

$

If the train does not stop on the doors opening enable area and the side doors emergency handles pulled, ATP shall require [EvacuationWithTrainStopped](#EvacuationWithTrainStopped).

%

[EvacuationWithTrainStopped](#EvacuationWithTrainStopped)(k)

= (([EmergencyHandleNotPulledSide](#EmergencyHandleNotPulledSide)(k) != **True**)

and ([TrainFilteredStopped](#TrainFilteredStopped)(k) == **True**)

and not [EnableDoorOpening\_A](#EnableDoorOpening_A)(k)

and not [EnableDoorOpening\_B](#EnableDoorOpening_B)(k))

#

@

InhibitProtectionEvacuationInDistance，在离站时禁止监控逃生手柄状态

%

def [InhibitProtectionEvacuationInDistance](#InhibitProtectionEvacuationInDistance)(k):

return [Offline.GetInhibitProtectionEvacuationInDistance](#GetInhibitProtectionEvacuationInDistance)(k)

#

@

InhibitProtectionEvacuationWithStop，在站间停车时禁止监控逃生手柄状态。

%

def [InhibitProtectionEvacuationWithStop](#InhibitProtectionEvacuationWithStop)(k):

return [Offline](#Offline).[GetInhibitProtectionEvacuationWithStop](#GetInhibitProtectionEvacuationWithStop)(k)

#

@

EBforEvacuationWhileTrainLeavingStation，出站时的逃生请求EB.

$

If the train leaving station evacuation has been required, ATP shall trigger the emergency brake.

%

def [EBforEvacuationWhileTrainLeavingStation](#EBforEvacuationWhileTrainLeavingStation)(k):

return ([EvacuationWhileLeavingStation](#EvacuationWhileLeavingStation)(k)

and not [InhibitProtectionEvacuationInDistance](#InhibitProtectionEvacuationInDistance)(k))

#

@

EBforEvacuationWithTrainStopped，站间停车时的逃生请求EB.

$

If the train stopped evacuation has been required, ATP shall trigger the emergency brake.

%

def [EBforEvacuationWithTrainStopped](#EBforEvacuationWithTrainStopped)(k):

return ([EvacuationWithTrainStopped](#EvacuationWithTrainStopped)(k)

and not [InhibitProtectionEvacuationWithStop](#InhibitProtectionEvacuationWithStop)(k))

#

@

PBforOperationalRequest，来自CCNV的ZVRD输出请求

$

PBforOperationalRequest stands for the ZVRD output order from CCNV.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[PBforOperationalRequest](#PBforOperationalRequest)(k)

= not [NonVitalRequest](#NonVitalRequest).VitalParkingBrakingNotRequested(k)

else:

[PBforOperationalRequest](#PBforOperationalRequest) = **True**

#

@

TrainParkingBrakeRequested，判断本周期是否需要施加停车制动。条件如下：由于moral time 导致需要输出停车制动；或者，由于超能导致需要输出

停车制动；或者，由于在PSD区域内车门未关闭而导致需要输出停车制动；或者，由于NUDE导致需要输出停车制动；或者，CCNV请求需要输出停车制动；或者，由于PSD未关闭而导致需要输出停车制动。或者，本周期已请求EB输出。

$

[TrainParkingBrakeRequested](#TrainParkingBrakeRequested), determine whether to apply parking brake. This variable shall be **True** when one of the following conditions met:Train is in front of a possibly non-exclusive route,Synthesis of speed constraints on the train implies that it is not allowed to move anymore. Any movement may lead to an hazardous situation,Train is located on a passenger exchange area with PSD and train doors are not proven closed and locked,Train is located on a passenger exchange area with PSD and PSD are not proven closed and locked,There is a possibility of undetectable dangers,An operational parking brake is requested,The PSD are opened and are under the supervision of ATP,The EB has been requested in this cycle.

%

[TrainParkingBrakeRequested](#TrainParkingBrakeRequested) = ([PBonNonExclusiveRoute](#PBonNonExclusiveRoute)(k)

or [PBforOverEnergy](#PBforOverEnergy)(k)

or [PBforTrainDoorsNotClosedAndLocked](#PBforTrainDoorsNotClosedAndLocked)(k)

or [PBforPSDnotClosedAndLocked](#PBforPSDnotClosedAndLocked)(k)

or [PBforUndetectableDangerRisk](#PBforUndetectableDangerRisk)(k)

or [PBforOperationalRequest](#PBforOperationalRequest)(k)

or [PBforPSDopenedAndSupervisedByATP](#PBforPSDopenedAndSupervisedByATP)(k)

or not [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k))

#

@

InhibitParkingBrake，当前不施加停车制动。

$

[InhibitParkingBrake](#InhibitParkingBrake)，ATP software do not apply the parking brake.

%

[InhibitParkingBrake](#InhibitParkingBrake) = not [TrainParkingBrakeRequested](#TrainParkingBrakeRequested)(k)

#

@

IncompatibleDistantATP，判断本ATP与冗余ATP之间的[Coreld](#Coreld)和[SubSystemID](#SubSystemID)是否相匹配。当初始化，冗余ATP信息不可用，或者冗余ATP读取的Dataplug中的SSID与本ATP相一致而[Coreld](#Coreld)不一致时，认为两端ATP相互匹配；否则，ATP将触发紧急制动。

$

The [Coreld](#Coreld) and [SubSystemID](#SubSystemID) of the ATP and redundant ATP need to compare for the consistency, which records in [IncompatibleDistantATP](#IncompatibleDistantATP). In initialization, the message from redundant ATP cannot be used. On the other hand, when SubSystemID in the Dataplug read by redundant ATP is the same, but the Coreld is different, both ATP regards as consistency. Otherwise, ATP would trigger emergency brake.

%

def [IncompatibleDistantATP](#IncompatibleDistantATP)(k):

if (Initialization

or not [OtherATPmessageValid](#OtherATPmessageValid)(k)):

return **False**

elif ([OtherCoreId](#OtherCoreId)(k) != [OtherATP](#OtherATP)(k).[CoreId](#CoreId)

or [SubSystemId](#SubSystemId)(k) != [OtherATP](#OtherATP)(k).CC\_SSID):

return **True**

else:

return [IncompatibleDistantATP](#IncompatibleDistantATP)(k-1)

#

@

EBforOperationalRequest，来自CCNV的EB输出请求

$

ATP shall trigger emergency brake according to CCNV‘s operational emergency brake request.

%

if ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

[EBforOperationalRequest](#EBforOperationalRequest)(k) = not [NonVitalRequest](#NonVitalRequest).EmergencyBrakingNotRequested(k)

else:

[EBforOperationalRequest](#EBforOperationalRequest) = **True**

#

@

TrainEmergencyBrakeRequested，判断本周期是否需要施加EB。

$

ATP shall control emergency brake output according following emergency braking requests from control functions:moral-time control function has detected an hazardous situation (route exclusivity violation);train speed is no longer compliant with respect of whole speed restriction of guide way;an approachable speed limit has been over-run (RM speed limit or memorized location speed limit);an over-speed in reverse direction of travel has been detected;an emergency evacuation is required for passengers;train departure with not all doors closed and locked has been detected;the train starts to move on a PSD zone which status is not "all PSD proven closed and locked";train has moved although there are potential undetectable dangers;an operational emergency braking has been requested by CC-Non Vital;train end doors are not closed and locked; not all doors closed and locked has been

detected on a PSD zone and parking brake is not applied, not all PSD closed and locked has been detected on a PSD zone and parking brake is not applied;the approachable signal is overrun;the VLE-2 safe timer failed;the information of Dataplug in both ends of cab is inconsistent.

%

[TrainEmergencyBrakeRequested](#TrainEmergencyBrakeRequested)(k)

= [EBonNonExclusiveRoute](#EBonNonExclusiveRoute)(k)

or [EBforOverEnergy](#EBforOverEnergy)(k)

or [EBforRMoverSpeed](#EBforRMoverSpeed)(k)

or [EBforMemorizedLocationOverSpeed](#EBforMemorizedLocationOverSpeed)(k)

or [EBforRollbackOverSpeed](#EBforRollbackOverSpeed)(k)

or [EBforReverseOverSpeed](#EBforReverseOverSpeed)(k)

or [EBforEvacuationWhileTrainLeavingStation](#EBforEvacuationWhileTrainLeavingStation)(k)

or [EBforEvacuationWithTrainStopped](#EBforEvacuationWithTrainStopped)(k)

or [[EBforDepartureWithoutTDCL](#EBforDepartureWithoutTDCL)](#DepartureWithNoTDCL)(k)

or [EBforMovingWithoutTDCL](#EBforMovingWithoutTDCL)(k)

or [EBforUnexpectedPSDopening](#EBforUnexpectedPSDopening)(k)

or [EBforUndetectableDangerRisk](#EBforUndetectableDangerRisk)(k)

or [EBforOperationalRequest](#EBforOperationalRequest)(k)

or [EBforNotAllTrainEndHoldDoorsClosed](#EBforNotAllTrainEndHoldDoorsClosed)(k)

or [EBforPBnotAppliedDueToTrainDoors](#EBforPBnotAppliedDueToTrainDoors)(k)

or [EBforPBnotAppliedDueToPSD](#EBforPBnotAppliedDueToPSD)(k)

or [ApproachableSignalOverrun](#ApproachableSignalOverrun)(k)

or [SafeTimerFailed](#SafeTimerFailed)(k)

or [IncompatibleDistantATP](#IncompatibleDistantATP)(k)

#

@

InhibitEmergencyBrake，输出和缓解EB的条件

$

If an emergency braking request ordered by a control function, ATP shall not inhibit emergency brake until train filtered-stop reached. ATP shall inhibit emergency brake if and only if train detected at filtered stop and there is no emergency braking request from control functions.

%

if ([InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1) == **True**)

[InhibitEmergencyBrake](#InhibitEmergencyBrake) = not [TrainEmergencyBrakeRequested](#TrainEmergencyBrakeRequested)(k)

elif (([InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1) == **False**)

and ([TrainFilteredStopped](#TrainFilteredStopped)(k) == **True**))

[InhibitEmergencyBrake](#InhibitEmergencyBrake) = not [TrainEmergencyBrakeRequested](#TrainEmergencyBrakeRequested)(k)

else:

[InhibitEmergencyBrake](#InhibitEmergencyBrake) = [InhibitEmergencyBrake](#InhibitEmergencyBrake)(k-1)

#

@

CCworkOvertime，监控CC是否连续工作超过**MAX\_RESET\_TIME**时间(该时间小于**MAX\_ATP\_LOOP\_HOUR**)。如果CC运行超过**MAX\_RESET\_TIME**时间，则ATP需将所有对VIOM输出的端口置为限制状态。

$

ATP shall monitor the CC continuous work time. If the CC is running more than **MAX\_RESET\_TIME** (the value is far less than **MAX\_ATP\_LOOP\_HOUR**), the ATP shall set all output to VIOM as restricted status.

%

def [CCworkOvertime](#CCworkOvertime)(k):

return (([CoreId](#CoreId)(k) is **END\_1**

and (([ATPtime](#ATPtime)(k) - **CC1\_INIT\_TIME**) > MAX\_RESET\_TIME))

or ([CoreId](#CoreId)(k) is **END\_2**

and (([ATPtime](#ATPtime)(k) - **CC2\_INIT\_TIME**) > MAX\_RESET\_TIME)))

#

@

VIOM1OutNotDisabled，VIOM2OutNotDisabled，CCNV请求“非禁止安全输出”。当来自CCNV的消息无效时，应设置CCNV请求的“非禁止安全输出”为限制状态；否则，根据CCNV发送的状态字进行设置。

$

Whether CCNV request the channel of VIOM shall be disabled or not.

%

def [VIOM1OutNotDisabled](#VIOM1OutNotDisabled)(port, k):

return ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest)(k).Viom1[port])

def [VIOM2OutNotDisabled](#VIOM2OutNotDisabled)(port, k):

return ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest)(k).Viom2[port])

#

@

VIOM1VitalOut，VIOM2VitalOut，ATP输出给VIOM的车辆安全控制命令。

对于每一个端口的具体含义，是由项目配置的。ATP支持的可配置端口如Table 5‑14所示。只有当CC未工作超时且CCNV未禁止该端口输出时，才能根据ATP计算结果输出该端口；否则，ATP默认该端口为限制状态。

%

def [VIOM1VitalOut](#VIOM1VitalOut)(k):

for port in [range](#range)(0, MAX\_VITAL\_OUTPUT\_NB):

if (not [CCworkOvertime](#CCworkOvertime)(k)

and not [MatchRebootCondition](#MatchRebootCondition)(k)

and [VIOM1OutNotDisabled](#VIOM1OutNotDisabled)(k)[port]):

[VIOM1VitalOut](#VIOM1VitalOut)[port] = [Offline](#Offline).[GetVIOM1VitalOut](#GetVIOM1VitalOut)(port)

else:

[VIOM1VitalOut](#VIOM1VitalOut)[port] = **False**

return [VIOM1VitalOut](#VIOM1VitalOut)

def [VIOM2VitalOut](#VIOM2VitalOut)(k):

for port in [range](#range)(0, MAX\_VITAL\_OUTPUT\_NB):

if (not [CCworkOvertime](#CCworkOvertime)(k)

and not [MatchRebootCondition](#MatchRebootCondition)(k)

and [VIOM2OutNotDisabled](#VIOM2OutNotDisabled)(k)[port]):

[VIOM2VitalOut](#VIOM2VitalOut)[port] = [Offline](#Offline).[GetVIOM2VitalOut](#GetVIOM2VitalOut)(port)

else:

[VIOM2VitalOut](#VIOM2VitalOut)[port] = **False**

return [VIOM2VitalOut](#VIOM2VitalOut)

#

@

TrainHeadMinLocation，车头最小定位位置。根据[REF5]，在[LocReport](#LocReport)中的坐标单位为0.5米，因此需进行单位转换。转换时应当向列车的“后方”即上游方向取整。如果列车失位，则设置相关定位信息为无效值；否则，如果列车向**UP**方向运行，则：车头最小定位所在BLOCK号不变；车头最小定位所在坐标以0.5米为单位向下取整；车头方向为**LOCREPORT\_DIRECTION\_UP**。

否则，如果车头最小定位坐标加0.5米小于该BLOCK长度，则：车头最小定位所在BLOCK号不变；车头最小定位所在坐标以0.5米为单位向上取整；车头方向为**LOCREPORT\_DIRECTION\_DOWN**。否则，如果车头最小定位所在BLOCK，与该BLOCK的**UP**方向下个BLOCK之间存在灯泡线极点，则：车头最小定位所在BLOCK需改为其**UP**方向的下个BLOCK；车头最小定位所在坐标为下个BLOCK长度以0.5米为单位向下取整；车头方向为**LOCREPORT\_DIRECTION\_UP**。否则：车头最小定位所在BLOCK需改为其**UP**方向的下个BLOCK；车头最小定位所在坐标为0；车头方向为**LOCREPORT\_DIRECTION\_DOWN**。

$

ATP shall send the minimum head location of the active cab to the [ZC](#ZC), including the block id and its abscissa. According to [REF5], the unit of the abscissa in Location Report is 0.5 meter, so the ATP needs to convert its internal unit to match that. The conversion shall be safety-oriented, which means the envelope of the train location tend to be "stretched" to the both ends. The rules of conversion are as following ARDL:

%

def [TrainHeadMinLocation](#TrainHeadMinLocation)(k):

if (not [TrainLocalized](#TrainLocalized)(k)):

[TrainHeadMinLocation](#TrainHeadMinLocation).Block = 0

[TrainHeadMinLocation](#TrainHeadMinLocation).Abscissa= 0

[TrainHeadOrientation](#TrainHeadOrientation) = **LOCREPORT\_DIRECTION\_UNKNOWN**

elif ([TrainFrontLocation](#TrainFrontLocation)(k).Min.Ort is **UP**):

[TrainHeadMinLocation](#TrainHeadMinLocation).Block = [TrainFrontLocation](#TrainFrontLocation)(k).Min.Block(k)

[TrainHeadMinLocation](#TrainHeadMinLocation).Abscissa = ([round.floor](#roundfloor)([TrainFrontLocation](#TrainFrontLocation)(k).Min.Abscissa(k)

/ **ABSCISSA\_TO\_HALF\_METER**))

[TrainHeadOrientation](#TrainHeadOrientation) = **LOCREPORT\_DIRECTION\_UP**

elif ([TrainFrontLocation](#TrainFrontLocation)(k).Min.Abscissa(k) + **ABSCISSA\_TO\_HALF\_METER**

<= [TrackMap](#TrackMap).[Blocks](#Blocks)[[TrainFrontLocation](#TrainFrontLocation)(k).Min.Block].Length):

[TrainHeadMinLocation](#TrainHeadMinLocation).Block = [TrainFrontLocation](#TrainFrontLocation)(k).Min.Block(k)

[TrainHeadMinLocation](#TrainHeadMinLocation).Abscissa = [round.ceil](#roundceil)([TrainFrontLocation](#TrainFrontLocation)(k).Min.Abscissa(k)

/ **ABSCISSA\_TO\_HALF\_METER**)

[TrainHeadOrientation](#TrainHeadOrientation) = **LOCREPORT\_DIRECTION\_DOWN**

else:

NextBlock = [TrackMap](#TrackMap).[NextBlock](#NextBlock)([TrackMap](#TrackMap).[Blocks](#Blocks)[[TrainFrontLocation](#TrainFrontLocation)(k).Min.Block], **UP**)

if [TrackMap.ExistThePole](#ExistThePole)([TrainFrontLocation](#TrainFrontLocation)(k).Min.Block, NextBlock.Id):

[TrainHeadMinLocation](#TrainHeadMinLocation).Block = NextBlock.Id

[TrainHeadMinLocation](#TrainHeadMinLocation).Abscissa = [round.floor](#roundfloor)(NextBlock.Length

/ **ABSCISSA\_TO\_HALF\_METER**)

[TrainHeadOrientation](#TrainHeadOrientation) = **LOCREPORT\_DIRECTION\_UP**

else:

[TrainHeadMinLocation](#TrainHeadMinLocation).Block = NextBlock.Id

[TrainHeadMinLocation](#TrainHeadMinLocation).Abscissa = 0

[TrainHeadOrientation](#TrainHeadOrientation) = **LOCREPORT\_DIRECTION\_DOWN**

return [TrainHeadMinLocation](#TrainHeadMinLocation)

#

@

TrainHeadCoupledStatus，车头连挂状态。

$

ATP shall send the coupled status of the active train cab to the [ZC](#ZC).

%

def [TrainHeadCoupledStatus](#TrainHeadCoupledStatus)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

return (([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_NOT\_COUPLED**)

or ([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_COUPLED\_END2**))

else:

return (([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_NOT\_COUPLED**)

or ([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_COUPLED\_END1**))

#

@

TrainTailCabId，车尾ID号。

$

ATP shall send the current inactive cab id to the [ZC](#ZC).

%

def [TrainTailCabId](#TrainTailCabId)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

return **END\_2**

else:

return **END\_1**

#

@

TrainTailMinLocation，车尾最小定位所在位置。根据[REF5]，在[LocReport](#LocReport)中的坐标单位为0.5米，因此需进行单位转换。转换时应当导向ZC处理的安全侧，偏向将列车包络定位“拉长”，即向列车的上游方向取整。如果列车失位，设置上述定位信息为0；否则，如果列车车尾朝**DOWN**方向，则：车尾最小定位所在BLOCK号不变；车尾最小定位所在坐标以0.5米为单位向下取整；车尾方向为**LOCREPORT\_DIRECTION\_DOWN**。否则，如果车尾最小定位坐标加0.5米小于该BLOCK长度，则：车尾最小定位所在BLOCK号不变；车尾最小定位所在坐标以0.5米为单位向上取整；车尾方向为**LOCREPORT\_DIRECTION\_UP**。否则，如果车尾最小定位所在BLOCK找不到**UP**方向的下个BLOCK，则：车尾最小定位所在BLOCK号不变，车尾最小定位所在坐标以0.5米为单位向上取整（此时由于发送的坐标超过了Block长度，ZC会将本LocReport丢弃，不会影响安全）。车尾方向为**LOCREPORT\_DIRECTION\_UP**。否则，如果车尾最小定位所在BLOCK，与该BLOCK的**UP**方向下个BLOCK之间存在灯泡线极点，则：车尾最小定位所在BLOCK需改为其上行方向的下个BLOCK；车尾最小定位所在坐标为下游BLOCK长度以0.5米为单位向下取整；车尾方向为**LOCREPORT\_DIRECTION\_DOWN**。否则，车尾最小定位所在BLOCK需改为其上行方向的下个BLOCK；车尾最小定位所在坐标为0；车尾方向为**LOCREPORT\_DIRECTION\_UP**。

$

ATP shall send the minimum tail location of the active cab to the [ZC](#ZC), including the block id and its abscissa. According to [REF5], the unit of the abscissa in Location Report is 0.5 meter, so the ATP needs to convert its internal unit to match that. The convertion shall be safety-oriented, which means the envelope of the train location tend to be "stretched" to the both ends. The rules of convertion are as following ARDL:

%

def [TrainTailMinLocation](#TrainTailMinLocation)(k):

if (not [TrainLocalized](#TrainLocalized)(k)):

[TrainTailMinLocation](#TrainTailMinLocation).Block = 0

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = 0

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_UNKOWN**

elif ([TrainRearLocation](#TrainRearLocation)(k).Min.Ort is **DOWN**):

[TrainTailMinLocation](#TrainTailMinLocation).Block = [TrainRearLocation](#TrainRearLocation)(k).Min.Block

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = ([round.floor](#roundfloor)([TrainRearLocation](#TrainRearLocation)(k).Min.Abscissa

/ **ABSCISSA\_TO\_HALF\_METER**))

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_DOWN**

elif ([TrainRearLocation](#TrainRearLocation)(k).Min.Abscissa + **ABSCISSA\_TO\_HALF\_METER**

<= [TrackMap](#TrackMap).Block[[TrainRearLocation](#TrainRearLocation)(k).Min.Block].Length):

[TrainTailMinLocation](#TrainTailMinLocation).Block = [TrainRearLocation](#TrainRearLocation)(k).Min.Block

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = [round.ceil](#roundceil)([TrainRearLocation](#TrainRearLocation)(k).Min.Abscissa

/ **ABSCISSA\_TO\_HALF\_METER**)

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_UP**

else:

NextBlock = [TrackMap](#TrackMap).[NextBlock](#NextBlock)([TrackMap](#TrackMap).[Blocks](#Blocks)[[TrainRearLocation](#TrainRearLocation)(k).Min.Block], **UP**)

if (NextBlock is **None**):

[TrainTailMinLocation](#TrainTailMinLocation).Block = [TrainFrontLocation](#TrainFrontLocation)(k).Min.Block

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = [round.ceil](#roundfloor)([TrainRearLocation](#TrainRearLocation)(k).Min.Abscissa

/ **ABSCISSA\_TO\_HALF\_METER**)

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_UP**

elif ([TrackMap.ExistThePole](#ExistThePole)([TrainRearLocation](#TrainRearLocation)(k).Min.Block, NextBlock.Id)):

[TrainTailMinLocation](#TrainTailMinLocation).Block = NextBlock.Id

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = [round.floor](#roundfloor)(NextBlock.Length

/ **ABSCISSA\_TO\_HALF\_METER**)

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_DOWN**

else:

[TrainTailMinLocation](#TrainTailMinLocation).Block = NextBlock.Id

[TrainTailMinLocation](#TrainTailMinLocation).Abscissa = 0

[TrainTailOrientation](#TrainTailOrientation) = **LOCREPORT\_DIRECTION\_UP**

return [TrainTailMinLocation](#TrainTailMinLocation)

#

@

TrainTailCoupledStatus，车尾连挂状态。

$

ATP shall send the coupled status of the inactive train cab to the [ZC](#ZC).

%

def [TrainTailCoupledStatus](#TrainTailCoupledStatus)(k):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

return (([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_NOT\_COUPLED**)

or ([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_COUPLED\_END1**))

else:

return (([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_NOT\_COUPLED**)

or ([TrainCoupledType](#TrainCoupledType)(k) is **TRAIN\_COUPLED\_END2**))

#

@

LocationError，最大最小定位误差. 根据[REF5]，在LocReport中的长度单位是0.5米，因此需进行转换，转换时应当导向安全侧。

$

ATP shall send the location error to the [ZC](#ZC). According to [REF5], the unit of the location error in Location Report is 0.5 meter, so the ATP needs to convert its internal unit to match that. The convertion shall be safety-oriented, which means the location error tend to be "overestimated".

%

def LocationError(k):

if [TrainLocalized](#TrainLocalized)(k):

return [round.ceil](#roundceil)(([TrainLocation](#TrainLocation)(k).Uncertainty + **ABSCISSA\_TO\_HALF\_METER**)

/ **ABSCISSA\_TO\_HALF\_METER**)

else:

return 0

#

@

RouteSetNotNeededSendable，是否可以发送RSNN信息。其状态来自于项目可配置的列车输入采集。

$

According to the status of [RouteSetNotNeededSendable](#RouteSetNotNeededSendable), ATP can judge whether it is necessary to send RSNN information.

%

def [RouteSetNotNeededSendable](#RouteSetNotNeededSendable)(k):

return [Offline.GetRouteSetNotNeededSendable](#GetRouteSetNotNeededSendable)()

#

@

NonVitalRouteSetNotNeeded，RSNN状态

$

Whether the CCNV request route set note needed.

%

def [NonVitalRouteSetNotNeeded](#NonVitalRouteSetNotNeeded)(k):

return ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest).RouteSetNotNeeded(k))

#

@

TrainRouteSetNotNeeded，是否发送RSNN信息。

$

ATP shall send the route set not needed information to [ZC](#ZC).

%

def [TrainRouteSetNotNeeded](#TrainRouteSetNotNeeded)(k):

return ([TrainFilteredStopped](#TrainFilteredStopped)(k)

and [NonVitalRouteSetNotNeeded](#NonVitalRouteSetNotNeeded)(k)

and [RouteSetNotNeededSendable](#RouteSetNotNeededSendable)(k))

#

@

TrainCorrectDocking，列车是否正确停靠车站。

$

ATP shall send the docking correction information to the [ZC](#ZC).

%

def [TrainCorrectDocking](#TrainCorrectDocking)(k):

return ([EnableDoorOpening\_A](#EnableDoorOpening_A)(k) or [EnableDoorOpening\_B](#EnableDoorOpening_B)(k))

#

@

LocReportSpeed，列车最大速度，需转换为CC-ZC接口协议中的单位，并向上取整。

$

ATP shall send the maximum train speed to the [ZC](#ZC). According to [REF5], the unit of the speed in Location Report is KPH, so the ATP needs to convert its internal unit to match that. The convertion shall be safety-oriented, which means the speed tend to be "overestimated".

%

def [LocReportSpeed](#LocReportSpeed)(k):

return [round.ceil](#roundceil)([TrainMaxSpeed](#TrainMaxSpeed)(k) / **KMPH\_TO\_MMPS**)

#

@

TrainMonitoringMode，监控模式.

$

ATP shall send the current monitoring mode to the [ZC](#ZC).

%

def [TrainMonitoringMode](#TrainMonitoringMode)(k):

if ([MotionProtectionInhibition](#MotionProtectionInhibition)(k)

and [RMRselectedDrivingMode](#RMRselectedDrivingMode)(k)):

return **RMR**

elif ([MotionProtectionInhibition](#MotionProtectionInhibition)(k)):

return **RMF**

else:

return **OTHERS**

#

@

SignalOverrideSendable，发给ZC的关信号机命令。

%

def [SignalOverrideSendable](#SignalOverrideSendable)(k):

return [Offline.GetSignalOverrideSendable](#GetSignalOverrideSendable)(k)

#

@

SignalsOverride，是否CBTC下取消信号。

$

ATP shall send the signal override information to the [ZC](#ZC).

%

def [SignalsOverride](#SignalsOverride)(k):

return ([SignalOverrideSendable](#SignalOverrideSendable)(k)

and not [MotionProtectionInhibition](#MotionProtectionInhibition)(k)

and ([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k)

and [NonVitalRequest](#NonVitalRequest).CancelSignal))

#

@

ATCcontrolledTrain，ATP未被切除。

%

def [ATCcontrolledTrain](#ATCcontrolledTrain)(k):

return [Offline.GetATCcontrolledTrain](#GetATCcontrolledTrain)(k)

#

@

DateSynchronizationReceived，收到时钟同步消息

%

def [DateSynchronizationReceived](#DateSynchronizationReceived)(lcId, k):

return [Message.Received](#Received)([DateSynchronizationReport](#DateSynchronizationReport)(lcId), k)

#

@

DateSynchronisationReportAvailable用于判断当前周期收到的LC安全时间消息是否可用。当满足以下条件时，设置DateSynchronisationReportAvailable为**True**。否则，设置DateSynchronisationReportAvailable为**False**。本周期收到来自LC的[DateSynchronizationReport](#DateSynchronizationReport)消息，并且校核字正确；如果收到LC消息是LC应答本端CC发出的消息，且满足以下条件时:当前ATP时间应大于消息中所带的ccLoopHour;并且当前ATP时间与消息中所带的ccLoopHour的差值应当小于之前ATP使用的LC消息时间。如果该消息是LC应答远端CC发出的消息，则应满足以下条件:当前ATP维护的远端ATP最小时间应大于消息中所带的ccLoopHour;并且当前ATP维护的远端ATP最大时间时间与消息中所带的ccLoopHour的差值，应当小于之前ATP使用的LC消息时间。

$

[DateSynchronisationReportAvailable](#DateSynchronisationReportAvailable) used to judge whether the LC synchronization message could use or not. When the below condition fulfilled, [DateSynchronisationReportAvailable](#DateSynchronisationReportAvailable) shall set as **True**. Otherwise, it set as **False**. If ATP Receives the [DateSynchronizationReport](#DateSynchronizationReport) message from LC, and the checksum is correct. And If the received message is LC to respond the local CC, ATP shall qualify with below conditions:The current ATP time is more than ccLoopHour in the messageThe different value between the current ATP time and the ccLoopHour is less than the LC message time in the previous ATP.Or else:, If the received message is LC to respond the remote CC, and it should be qualified with below conditions:The minimum time in remote ATP maintained by current ATP is more than the ccLoopHour.The different value between maximum time in remote ATP maintained by current ATP and the ccLoopHour is less than the LC message time in the previous ATP.

%

def [DateSynchronisationReportAvailable](#DateSynchronisationReportAvailable)(lcId, k):

return Message.[Available](#Available)([DateSynchronizationReceived](#DateSynchronizationReceived)(lcId, k),

[DateSynchronizationReport](#DateSynchronizationReport)(lcId).CcLoopHour,

[ATPsetting](#ATPsetting).LCloophourValidityTime,

[LastSynchronisationReportAge](#LastSynchronisationReportAge)(lcId, k-1),

k)

#

@

LastSynchronisationReportAge，记录从上次收到LC消息到现在的时间。用于在新收到LC消息时，比较该“新”消息是否的确比之前的消息“新”。即防止在网络传输中发生消息逆序的情况。

$

[LastSynchronisationReportAge](#LastSynchronisationReportAge) records the age from previous LC message to current cycle. It used to compare whether the new received message is more updating than last recorded one, to prevent the inverse transition in the network communication.

%

def [LastSynchronisationReportAge](#LastSynchronisationReportAge)(lcId, k):

return Message.[LastAge](#LastAge)([DateSynchronisationReportAvailable](#DateSynchronisationReportAvailable)(lcId, k),

[DateSynchronizationReport](#DateSynchronizationReport)(lcId).CcLoopHour,

[LastSynchronisationReportAge](#LastSynchronisationReportAge)(lcId, k-1),

k)

#

@

LCsynchronisationLoopHourValid，每周期更新LC消息是否还在有效期内。

$

[LCsynchronisationLoopHourValid](#LCsynchronisationLoopHourValid) will check whether the LC message is still valid.

%

def [LCsynchronisationLoopHourValid](#LCsynchronisationLoopHourValid)(lcId, k):

return Message.[Valid](#Valid)([DateSynchronizationReport](#DateSynchronizationReport)(lcId).CcLoopHour,

[ATPsetting](#ATPsetting).LCloophourValidityTime,

k)

#

@

对于线路上的LC，ATP需要维护其时间信息GroundTimeReference，时间信息更新规则如下：LC消息无效时，设为默认值；本周期收到新的可用的LC消息时，更新为消息中的时间；否则根据车载和轨旁周期的比值进行累加更新

$

ATP shall estimate a time called [GroundTimeReference](#GroundTimeReference) for trackside equipment ([ZC](#ZC) /LC). From power-up, [GroundTimeReference](#GroundTimeReference) shall consider invalid. The rules for time information are as follows:When LC message invalid, it is set as default value. When ATP receives an available LC message, it updates as the time of message. Otherwise, it updates the value according to the on board and track side cycle.

%

if (Initialization

or [LCsynchronisationLoopHourValid](#LCsynchronisationLoopHourValid)(LcId, k) != **True**)

[GroundTimeReference](#GroundTimeReference)[LcId].Time = **INVALID\_LC\_DATE**

[GroundTimeReference](#GroundTimeReference)[LcId].Milliseconds = 0

elif ([DateSynchronisationReportAvailable](#DateSynchronisationReportAvailable)(LcId, k))

[GroundTimeReference](#GroundTimeReference)[LcId].Time(k)= [DateSynchronizationReport](#DateSynchronizationReport).Synchrodate

[GroundTimeReference](#GroundTimeReference)[LcId].Milliseconds = 0

else:

[GroundTimeReference](#GroundTimeReference)[LcId].Time(k)

= [GroundTimeReference](#GroundTimeReference)[LcId].Time(k-1)

+ (([GroundTimeReference](#GroundTimeReference)[LcId].Milliseconds(k-1)+ **ATP\_CYCLE\_TIME\_MS**)

/ **SYNCHRODATE\_TIME\_UNIT\_MS**)

[GroundTimeReference](#GroundTimeReference)[LcId].Milliseconds(k)

= (([GroundTimeReference](#GroundTimeReference)[LcId].Milliseconds(k-1)+ **ATP\_CYCLE\_TIME\_MS**)

% **SYNCHRODATE\_TIME\_UNIT\_MS**)

#

@

EnableSendLocReport，当所对应的LC消息在有效期内时，才允许给ZC发送位置报告。

$

Before received the first valid synchronization from a LC, the corresponding LC vital time shall consider not significant, and the ATP shall prevent to send Location Report message to the ZC(s) associated to the corresponding LC.

%

def [EnableSendLocReport](#EnableSendLocReport)(LcId, k):

return [LCsynchronisationLoopHourValid](#LCsynchronisationLoopHourValid)(LcId, k)

#

@

TimeElapseBetweenTwoLocReport，计算发送[LocReport](#LocReport)的时间控制：

$

ATP shall calculate when to send the Location Report as the following ARDL:

%

def [TimeElapseBetweenTwoLocReport](#TimeElapseBetweenTwoLocReport)(k):

if (Initialization

or ([TimeElapseBetweenTwoLocReport](#TimeElapseBetweenTwoLocReport)(k-1) == 1)):

return [round.floor](#floor)([ATPsetting](#ATPsetting).LocReportEmissionPeriod)

else:

return [TimeElapseBetweenTwoLocReport](#TimeElapseBetweenTwoLocReport)(k-1) - 1

#

@

ZCidUnderTrainTail，车尾定位所在的ZC标识.

$

ATP shall record the [ZC](#ZC) area id where the train tail located. If the train is not localized, ATP shall use the current receiving beacon where located, or the last known [ZC](#ZC) area id.

%

def [ZCidUnderTrainTail](#ZCidUnderTrainTail)(k):

if (Initialization):

return 0

elif ([TrainLocalized](#TrainLocalized)(k)):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([TrainTailMinLocation](#TrainTailMinLocation)(k).Block)

elif ([MemorizedLocationAvailable](#MemorizedLocationAvailable)(k)):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([MemLocation](#MemLocation).Ext2.Block)

else:

return [TrackMap](#TrackMap).[ZCId](#ZCId)([MemLocation](#MemLocation).Ext1.Block)

elif ([NewBeaconObtained](#NewBeaconObtained)(k)):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([BeaconLastObtained](#BeaconLastObtained).Block)

else:

return [ZCidUnderTrainTail](#ZCidUnderTrainTail)(k-1)

#

@

ZCidUnderTrainHead，车头定位的ZC标识

$

ATP shall record the [ZC](#ZC) area id where the train head located. If the train is not localized, ATP shall use the current receiving beacon where located, or the last known [ZC](#ZC) area id.

%

def [ZCidUnderTrainHead](#ZCidUnderTrainHead)(k):

if (Initialization)

return 0

elif ([TrainLocalized](#TrainLocalized)(k)):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([TrainHeadMinLocation](#TrainHeadMinLocation)(k).Block)

elif ([MemorizedLocationAvailable](#MemorizedLocationAvailable)(k)):

if ([TrainFrontEnd](#TrainFrontEnd)(k) is **END\_1**):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([MemLocation](#MemLocation).Ext1.Block)

else:

return [TrackMap](#TrackMap).[ZCId](#ZCId)([MemLocation](#MemLocation).Ext2.Block)

elif ([NewBeaconObtained](#NewBeaconObtained)(k)):

return [TrackMap](#TrackMap).[ZCId](#ZCId)([BeaconLastObtained](#BeaconLastObtained).Block)

else:

return [ZCidUnderTrainHead](#ZCidUnderTrainHead)(k-1)

#

@

SendLocReportOnZCunderTrainTail，当距离上次发送[LocReport](#LocReport)到达既定时间，且车头和车尾不是同一个ZC时，要给车尾所在的ZC发送消息

$

When the time elapsed from last report is equal to the Location Report sending cycle, and the train tail located [ZC](#ZC) is valid and not as same as the train head [ZC](#ZC), and the [ZC](#ZC) train tail located authorized by the LC, ATP shall send the Location Report to this [ZC](#ZC) where train tail located.

%

def [SendLocReportOnZCunderTrainTail](#SendLocReportOnZCunderTrainTail)(k):

return ((([TimeElapseBetweenTwoLocReport](#TimeElapseBetweenTwoLocReport)(k) == 1)

or (not [MasterCCcore](#MasterCCcore)(k-1) and [MasterCCcore](#MasterCCcore)(k)))

and ([ZCidUnderTrainTail](#ZCidUnderTrainTail)(k) is not **None**)

and ([ZCidUnderTrainTail](#ZCidUnderTrainTail)(k) != [ZCidUnderTrainHead](#ZCidUnderTrainHead)(k))

and [EnableSendLocReport](#EnableSendLocReport)([TrackMap.ZC](#ZC)[[ZCidUnderTrainTail](#ZCidUnderTrainTail)(k)].LcId, k)

and ([VersionAuthorizedByLC](#VersionAuthorizedByLC)([ZCidUnderTrainTail](#ZCidUnderTrainTail)(k), k)))

#

@

SendLocReportOnZCunderTrainHead，是否给车头的ZC区发[LocReport](#LocReport)。

$

When the time elapsed from last report is equal to the Location Report sending cycle, and the train head located [ZC](#ZC) is valid, and the [ZC](#ZC) train head located authorized by the LC, ATP shall send the Location Report to this [ZC](#ZC) where train head located.

%

def [SendLocReportOnZCunderTrainHead](#SendLocReportOnZCunderTrainHead)(k):

return ((([TimeElapseBetweenTwoLocReport](#TimeElapseBetweenTwoLocReport)(k) == 1)

or (not [MasterCCcore](#MasterCCcore)(k-1) and [MasterCCcore](#MasterCCcore)(k)))

and ([ZCidUnderTrainHead](#ZCidUnderTrainHead)(k) is not **None**)

and [EnableSendLocReport](#EnableSendLocReport)([TrackMap.ZC](#ZC)[[ZCidUnderTrainHead](#ZCidUnderTrainHead)(k)].LcId, k)

and ([VersionAuthorizedByLC](#VersionAuthorizedByLC)([ZCidUnderTrainHead](#ZCidUnderTrainHead)(k), k)))

#

@

CCvariantRequestMsgReceived，收到来自CI的CC变量请求并校核字正确。

%

def [CCvariantRequestMsgReceived](#CCvariantRequestMsgReceived)(cbiId, k):

if (Initialization):

return **False**

elif ([Message.Received](#Received)([CCvariantRequest](#CCvariantRequest), k)):

return **True**

else:

return [CCvariantRequestMsgReceived](#CCvariantRequestMsgReceived)(Cbi, k-1)

#

@

ReceivedCBIloopHour，记录CBI请求中的联锁的loop hour。

%

def [ReceivedCBIloopHour](#ReceivedCBIloopHour)(cbi, k):

if (Initialization):

return **INVALID\_LOOP\_HOUR**

elif ([CCvariantRequestMsgReceived](#CCvariantRequestMsgReceived)(Cbi, k)):

return [CCvariantRequest](#CCvariantRequest)(Cbi, k).CbiLoopHour

else:

return [ReceivedCBIloopHour](#ReceivedCBIloopHour)(Cbi, k-1)

#

@

CCvariants，ATP发送给联锁的Overlap解锁信息。

$

ATP shall check the following conditions when sending overlap release to CBI:

Train front location is in overlap release zone,and the other ATP's overlap timer has expired,and ATP received variant request from CBI in this zone.

%

def [CCvariants](#CCvariants)([CbiId](#CbiId), k):

Orz = [TrackMap](#TrackMap).[ExistZoneLocationIncluded](#ExistZoneLocationIncluded)(**SGL\_OVERLAP\_RELEASE\_ZONE**,

[TrainFrontLocation](#TrainFrontLocation)(k).Max):

if ([OverlapReleasable](#OverlapReleasable)(k)

and [OtherATP](#OtherATP)(k).OverlapExpired

and [CCvariantRequestMsgReceived](#CCvariantRequestMsgReceived)(CbiId, k)

and Orz is not **None**

and Orz.[CbiId](#CbiId) == [CbiId](#CbiId)

and [CbiId](#CbiId) == [NonVitalRequest](#NonVitalRequest)(k).VariantRequestCbiId):

for Index in [range](#range)(0, MAX\_CC\_VARIANTS\_NB):

if (Orz.RadioBlockModeVariantIndex == Index):

[CCvariants](#CCvariants)[[CbiId](#CbiId)].Variant[Index] = **True**

else:

[CCvariants](#CCvariants)[[CbiId](#CbiId)].Variant[Index] = **False**

else:

[CCvariants](#CCvariants)[[CbiId](#CbiId)] = **None**

return [CCvariants](#CCvariants)

#

@

定时中断被激活后，在每次中断中对ImmediateNb，CycleSynchronized，Trace，Dt和CycleBiasNb进行更新：如果当前是上电后第一个中断：ATP需设置ImmediateNb为0，并根据所在的CPU初始化Trace，VCP的时间标签Dt，以及中间变量m：否则，如果ImmediateNb = 0，而[VitalTime](#VitalTime)与上个中断相比仍然未发生变化，则将CycleBiasNb加1。如果CycleBiasNb > 1，设置[CycleSynchronized](#CycleSynchronized)为**False**；否则，如果ImmediateNb = 0，而[VitalTime](#VitalTime)与上个中断相比发生了变化，则令ImmediateNb = 1；令Trace = PDoperationDt([Trace](#Trace)，**Bi**[[ImmediateNb](#ImmediateNb)], [Dt](#Dt)),设置CycleBiasNb = 0，而且[CycleSynchronized](#CycleSynchronized)为**True**.使用LockedImmediateCounter锁存此时的[ImmediateCounter](#ImmediateCounter)值，作为新周期初始的中断号。否则，将ImmediateNb的值加1；令Trace = PDoperationDt([Trace](#Trace), **Bi**[[ImmediateNb](#ImmediateNb)], [Dt](#Dt))；如果ImmediateNb > (**ATP\_INTERRUPT\_NB**-1)，则设置ImmediateNb = 0；并令Trace = PDoperation([Trace](#Trace)，m)，令Dt = PDoperation([Dt](#Dt), 0)

$

When the fixed-time interrupt triggered, ATP shall update the [ImmediateNb](#ImmediateNb)，[[CycleSynchronized](#CycleSynchronized)](#CycleSynchroized), [Trace](#Trace), [Dt](#Dt) and [CycleBiasNb](#CycleBiasNb). If it is the first interrupt after powered up, ATP shall set [ImmediateNb](#ImmediateNb) as zero，and initialize the [Trace](#Trace), [Dt](#Dt) (the dynamic time of VCP), and the middle variables m based on CPU. Or else:, If the [ImmediateNb](#ImmediateNb) is zero, but the [VitalTime](#VitalTime) has not changed comparing to the previous interrupt, then: Or else, If [ImmediateNb](#ImmediateNb) is zero, and the [VitalTime](#VitalTime) has changed comparing to the previous interrupt, then: Otherwise, set: and if the [ImmediateNb](#ImmediateNb) is greater than (**ATP\_INTERRUPT\_NB**-1), then:

%

if ([DataPlugContent](#DataPlugContent).VLECpuId == **CPU1**)

[Dt](#Dt) = **CPU1\_DT\_INIT**

[Trace](#Trace) = **CPU1\_TRACE\_0** ^ [Dt](#Dt)

m = InversePDoperation(**CPU1\_TRACE\_0**, **CPU1\_TRACE\_N**)

else:

[Dt](#Dt) = **CPU2\_DT\_INIT**

[Trace](#Trace) = **CPU2\_TRACE\_0** ^ [Dt](#Dt)

m = InversePDoperation(**CPU2\_TRACE\_0**, **CPU2\_TRACE\_N**)

if ([DataPlugContent](#DataPlugContent).VLECpuId == **CPU1**)

[Dt](#Dt) = **CPU1\_DT\_INIT**

[Trace](#Trace) = **CPU1\_TRACE\_0** ^ [Dt](#Dt)

m = InversePDoperation(**CPU1\_TRACE\_0**, **CPU1\_TRACE\_N**)

else:

[Dt](#Dt) = **CPU2\_DT\_INIT**

[Trace](#Trace) = **CPU2\_TRACE\_0** ^ [Dt](#Dt)

m = InversePDoperation(**CPU2\_TRACE\_0**, **CPU2\_TRACE\_N**)

[CycleBiasNb](#CycleBiasNb) = [CycleBiasNb](#CycleBiasNb) + 1

if ([CycleBiasNb](#CycleBiasNb) > 1)

[[CycleSynchronized](#CycleSynchronized)](#CycleSynchroized) = **False**

[ImmediateNb](#ImmediateNb) = 1

[Trace](#Trace) = PDoperationDt([Trace](#Trace)，**Bi**[[ImmediateNb](#ImmediateNb)], [Dt](#Dt))

[CycleBiasNb](#CycleBiasNb) = 0

[[CycleSynchronized](#CycleSynchronized)](#CycleSynchroized) = **True**

[LockedImmediateCounter](#LockedImmediateCounter) **=** [ImmediateCounter](#ImmediateCounter)

[ImmediateNb](#ImmediateNb) = [ImmediateNb](#ImmediateNb) + 1

[Trace](#Trace) = PDoperationDt([Trace](#Trace), **Bi**[[ImmediateNb](#ImmediateNb)], [Dt](#Dt))

[ImmediateNb](#ImmediateNb) = 0

[Trace](#Trace) = PDoperation([Trace](#Trace)，m)

[Dt](#Dt) = PDoperation([Dt](#Dt), 0)

#

@

VitalTime，ATP主任务维护的当前周期序号。上电后从0开始，每周期递增加1。当主任务在执行完成本周期的所有工作后，监控中断任务是否执行完成，即[ImmediateCounter](#ImmediateCounter)和[LockedImmediateCounter](#LockedImmediateCounter)的差值是否大于等于（**ATP\_INTERRUPT\_NB** -1）：若是，则表明主周期执行完成：将VitalTime送给另一个CPU模块；将[Trace](#Trace)(k)和[Dt](#Dt)(k)作为校核字送给VIOM进行校验。在本周期最后，设置, 否则，继续等待。

$

The [VitalTime](#VitalTime) stand for the current cycle of ATP deferred task. After power up, it starts from zero and increase one each cycle. When all the work is executed in the main task, ATP detects whether the interrupt task is over, i.e. the difference between [ImmediateCounter](#ImmediateCounter) and [LockedImmediateCounter](#LockedImmediateCounter) is equal to or larger than (**ATP\_INTERRUPT\_NB** -1). If it is so, it shows that the main task in this cycle finishes. Then ATP shall:send the [VitalTime](#VitalTime) to the other CPU,and send [Trace](#Trace) and [Dt](#Dt) to VIOM to check,and at the end of this cycle, set. Otherwise, keep waiting.

%

[VitalTime](#VitalTime) = [VitalTime](#VitalTime)(k-1) + 1

#

@

RTCtime，ATP维护的非安全时钟：ATP软件在初始化时从VLE-2后板上获取RTC时钟信息；之后，ATP软件每秒钟将该RTC时钟加1；但如果RTC时钟与来自CCNV的NTP时间差超过**MAX\_NTP\_TIME\_ERROR**，则使用NTP时间更新RTC时间。其中Time.Update()意为ATP软件每秒钟将RTC时间加1。

$

ATP software shall maintain the RTC time for non-vital functions.In initialization, ATP software get RTC time from VLE-2 board;And then, ATP software updates the RTC time every second;And if the difference between RTC time ATP used and the NTP time CCNV sent is greater than **MAX\_NTP\_TIME\_ERROR**, ATP shall reset the RTC time as NTP time. The Time.Update() means ATP software shall update the RTC time every second.

%

if (Initialization)

[RTCtime](#RTCtime) = [VLE\_RTCtime](#VLE_RTCtime)

elif (([ATOcontrolTimeValid](#ATOcontrolTimeValid)(k) == **True**)

and ([NonVitalRequest](#NonVitalRequest).NtpTime != **None**)

and (|[NonVitalRequest](#NonVitalRequest).NtpTime - [RTCtime](#RTCtime)(k-1)| > **MAX\_NTP\_TIME\_ERROR**))

[RTCtime](#RTCtime) = [NonVitalRequest](#NonVitalRequest).NtpTime

else:

[RTCtime](#RTCtime) = Time.Update()

#

@

Message. ReplyLocalCC，判断该消息中的MessageLoopHour是否与本CC一致

%

def [ReplyLocalCC](#ReplyLocalCC)(MessageLoopHour):

if ((MessageLoopHour >= **CC1\_INIT\_TIME**

and MessageLoopHour <= **CC1\_MAX\_TIME**

and [CoreId](#CoreId) is **END\_1**)

or (MessageLoopHour >= **CC2\_INIT\_TIME**

and MessageLoopHour <= **CC2\_MAX\_TIME**

and [CoreId](#CoreId) is **END\_2**)):

return **True**

else:

return **False**

#

@

Message. ReplyDistantCC，判断该消息中的MessageLoopHour是否与远端CC一致

%

def [ReplyDistantCC](#ReplyDistantCC)(MessageLoopHour):

if ((MessageLoopHour >= **CC1\_INIT\_TIME**

and MessageLoopHour <= **CC1\_MAX\_TIME**

and [OtherCoreId](#OtherCoreId) is **END\_1**)

or (MessageLoopHour >= **CC2\_INIT\_TIME**

and MessageLoopHour <= **CC2\_MAX\_TIME**

and [OtherCoreId](#OtherCoreId) is **END\_2**)):

return **True**

else:

return **False**

#

@

Message. ModularSub，loop hour之间的模减,考虑到判断消息可用的算法，此处只需强制为严格大于才返回**True**因为在计算[LastAge](#LastAge)时是要+1的，若之前取等号,则加1后可能该消息已无效

%

def [ModularSub](#ModularSub)(lh\_1, lh\_2):

if (([ReplyLocalCC](#ReplyLocalCC)(lh\_1) and [ReplyLocalCC](#ReplyLocalCC)(lh\_2))

or ([ReplyDistantCC](#ReplyDistantCC)(lh\_1) and [ReplyDistantCC](#ReplyDistantCC)(lh\_2))):

if (lh\_1 - lh\_2 < 0):

return ((lh\_1 - lh\_2) + MAX\_ATP\_LOOP\_HOUR)

else:

return (lh\_1 - lh\_2)

else:

raise LoopHourException

#

@

Message.IsMoreRecent，若两端都对应CC1或CC2，且lh\_1 > lh\_2，则返回**True**；否则返回False

%

def IsMoreRecent(lh\_1, lh\_2):

if (([ReplyLocalCC](#ReplyLocalCC)(lh\_1) and [ReplyLocalCC](#ReplyLocalCC)(lh\_2))

or [ReplyDistantCC](#ReplyDistantCC)(lh\_1) and [ReplyDistantCC](#ReplyDistantCC)(lh\_2)):

return ([ModularSub](#ModularSub)(lh\_1, lh\_2) > 0

and [ModularSub](#ModularSub)(lh\_1, lh\_2) < **ATP\_MESSAGE\_MAX\_DELAY**)

else:

raise LoopHourException

#

@

Message.Received，收到消息并且校验正确

%

def Received(MessageContent, k):

return ([Exists](#Exists)(MessageContent, k)

and [VitalChecksumValid](#VitalChecksumValid)(MessageContent))

#

@

Message.Available，判断消息是否可用，即收到消息校验正确，且在有效期内，且比之前收到的更新;当收到一条[ATPtime](#ATPtime)-MessageLoopHour恰好等于MessageValidityTime的消息不应认为有效;否则计算本周期[LastAge](#LastAge)就将大于消息有效期

%

def [Available](#Available)(MessageReceived, MessageLoopHour, MessageValidityTime, LastMessageAge, k):

if (MessageReceived

and (([ReplyLocalCC](#ReplyLocalCC)(MessageLoopHour)

and IsMoreRecent([ATPtime](#ATPtime)(k), MessageLoopHour)

and IsMoreRecent(MessageLoopHour + MessageValidityTime, [ATPtime](#ATPtime)(k))

and ([ModularSub](#ModularSub)([ATPtime](#ATPtime)(k) - MessageLoopHour) < LastMessageAge))

or ([ReplyDistantCC](#ReplyDistantCC)(MessageLoopHour)

and IsMoreRecent([OtherATPminTime](#OtherATPminTime)(k), MessageLoopHour)

and IsMoreRecent(MessageLoopHour + MessageValidityTime, [OtherATPmaxTime](#OtherATPmaxTime)(k))

and ([ModularSub](#ModularSub)([OtherATPmaxTime](#OtherATPmaxTime)(k) - MessageLoopHour) < LastMessageAge)))):

return **True**

else:

return **False**

#

@

Message.LastAge，当前消息已存活的时间。Age的最大值为**REPORT\_AGE\_MAX**。

%

def [LastAge](#LastAge)(MessageAvailable, MessageLoopHour, PreviousLastAge, k):

if (Initialization

or PreviousLastAge >= **REPORT\_AGE\_MAX**):

return **REPORT\_AGE\_MAX**

elif (MessageAvailable,

and [ReplyLocalCC](#ReplyLocalCC)(MessageLoopHour)):

return (1 + [ModularSub](#ModularSub)([ATPtime](#ATPtime)(k), MessageLoopHour))

elif (MessageAvailable,

and [ReplyDistantCC](#ReplyDistantCC)(MessageLoopHour)):

return (1 + [ModularSub](#ModularSub)([OtherATPminTime](#OtherATPminTime)(k), MessageLoopHour))

else:

return (1 + PreviousLastAge)

#

@

Message. Valid，当前消息是否依然有效

%

def [Valid](#Valid)(MessageLoopHour, MessageValidityTime, k):

if ([ReplyLocalCC](#ReplyLocalCC)(MessageLoopHour)):

return (([ATPtime](#ATPtime)(k) - MessageLoopHour) <= MessageValidityTime)

elif ([ReplyDistantCC](#ReplyDistantCC)(MessageLoopHour)):

return (([OtherATPmaxTime](#OtherATPmaxTime)(k) - MessageLoopHour) <= MessageValidityTime)

else:

return **False**

#