# Manuel d'automatisation 2N

v.2.45

www.2n.com

# Contenu :

# 1. Termes et symboles

Les symboles et pictogrammes suivants sont utilisés dans le mode d'emploi.

**Risque d'accident**

- **Respectez toujours** ces consignes pour écarter un risque d'accident.

**Avertissement**

- **Respectez toujours** ces consignes pour éviter d'endommager l'appareil.

**Observation**

- **Observation importante.** Le non-respect des consignes peut entraîner un dysfonctionnement de l'appareil.

**Conseil**

- **Informations utiles** pour un fonctionnement ou un réglage plus facile et plus rapide.

**Note**

- Procédés et conseils pour profiter de manière efficace des caractéristiques de l'appareil.

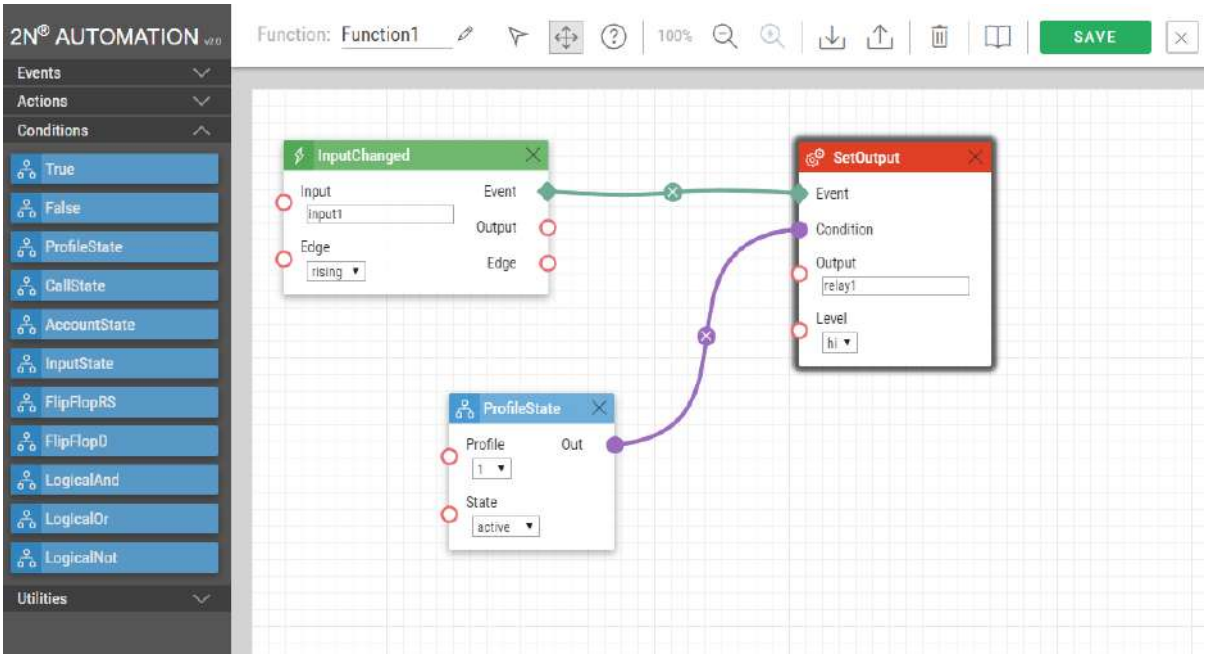
# 2. Configuration de l'Automatisation IP 2N

Les **appareils 2N** offrent des options de configuration flexibles en fonction des besoins de l'utilisateur. Si les options de paramétrage standards (paramètres interrupteurs / appels, par exemple) sont insuffisantes pour l'utilisation prévue, vous pouvez bénéficier d'une interface programmable spéciale appelée Interface d'**Automatisation**. En règle générale, **l'automatisation** est utile pour les applications qui nécessitent une interconnexion plus complexe avec des systèmes tiers.

> **Note**
>
> - **L'automatisation** fonctionne uniquement avec une licence Intégration ou Gold valide.

Certains modèles **d'appareil IP 2N** sont équipés d'un certain nombre d'entrées et de sorties logiques, dont la plupart peuvent être configurées comme des interrupteurs de l'appareils (voir la sous-section Interrupteurs). Vous pouvez utiliser toutes ces entrées et sorties **d'automatisation** selon des combinaisons variables.

**L'automatisation** vous permet de combiner des **Evénements** survenant dans le système (tels que l'utilisation des touches d'appel, l'utilisation de la carte RFID, le changement d'état d'une entrée logique, l'activation de l'autoprotection…etc.) avec des **Actions** spécifiques (telles que l'activation d'une sortie relais, la lecture d'un message audio, le déclenchement d'un appel, l'envoi d'un email ou d'une commande http…etc.), le cas échéant. De plus, l'exécution de ces actions peut être liée à des **Conditions** spécifiques présélectionnées (état du profil temporel ou état de l'entrée logique, par exemple)**.**



Le schéma ci-dessus montre une interconnexion typique des blocs Evénement, Action et Condition. Une action est toujours liée à un événement prédéfini et est exécutée lorsqu'une condition spécifique est remplie. La condition est facultative et si aucune condition n'est sélectionnée, alors l'action est exécutée à chaque fois que l'événement en question se produit. **L'automatisation** définit un certain nombre d'événements, d'actions et de conditions à définir en fonction du besoin. Reportez-vous aux sous-sections ci-dessous pour découvrir la liste complète.

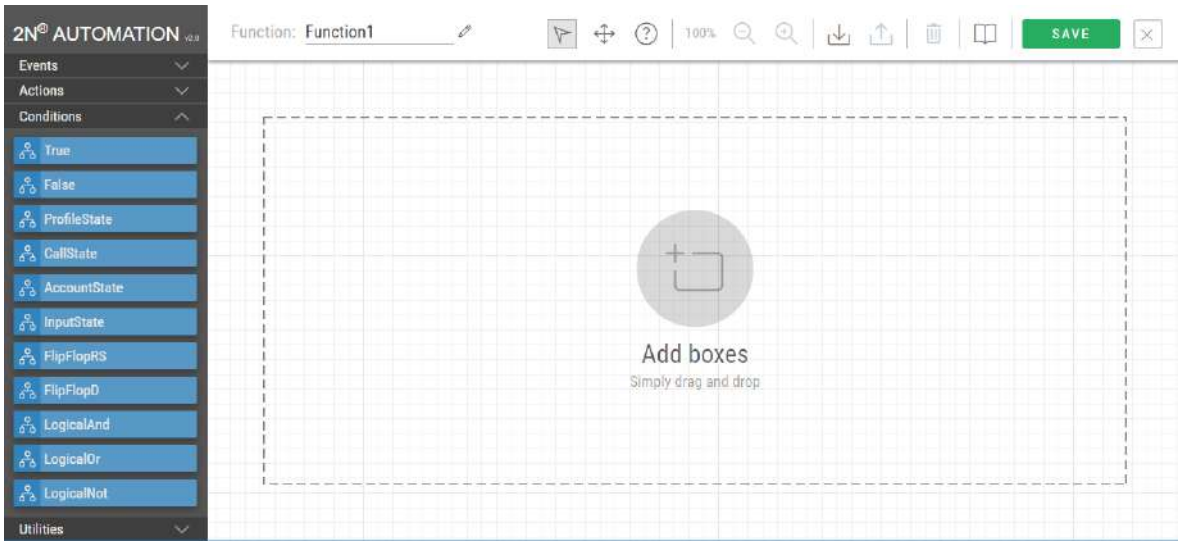
L'exemple présenté dans l'image ci-dessus peut être interprété comme ceci : L'action **« SetOutput »** (Activation de la sortie relais 1) est exécutée si l'événement **« InputChanged »** (entrée logique 1 passe de la position 0 à la position 1) et que la condition **« ProfileState »** (Profil temporel 1) est remplie.

## Fonctions de l'automatisation

- **Fonction Marque-page** – Les **appareils IP 2N** permettent de créer et d'interconnecter jusqu'à 30 blocs sur 5 pages indépendantes (quel que soit le type de bloc – événements, actions et conditions). Plusieurs actions peuvent être affectées à un seul événement ou à une seule condition. Ainsi, vous pouvez créer 15 actions et les affecter à 15 événements ou bien créer 29 actions et les affecter à 1 seul et même événement, par exemple.
  - **Activé** – activation de la fonction
  - **Nom** – nom de la fonction
  - **Statut** – Etat actuel de la fonction : Activé/arrêté/vide/erreur
  - **Action** – cliquez sur  pour définir une fonction et sur  pour la supprimer.


## Contrôle de l'Automatisation

L'image ci-dessous représente une Fonction d'Automatisation vide.




**Colonne de Bloc de la fonction** – elle comprend quatre groupes de blocs : Evénements, Actions, Conditions et Utilités. Il suffit de déplacer les blocs sur la fenêtre du milieu.

**Barre d'outils** – elle intègre les fonctions de contrôle de votre scénario d'automatisation.

| Pictogrammes | Description |
|---|---|
| Function: Function1 | Nom de la fonction |
| | Fonction – Mode édition |
| | Fonction – Mode déplacement des blocs |
| | Mode aide pour les blocs de la fonction. Cliquez sur un bloc pour afficher l'aide |
| 68% | Grossissement |
| | Zoom −/+ |
| | Importer/Exporter la Fonction |
| | Effacer |

| Pictogrammes | Description |
|---|---|
|  | Manuel d'Automatisation |
|  | Sauvegarder |
|  | Quitter la Fonction |

- Le **bureau** vous permet de déplacer et d'interconnecter les blocs dans la fonction.

## Paramètres des blocs

Sélectionnez l'événement (Event.xxx), l'action (Action.xxx) et la condition (Condition.xxx) requis dans la colonne **Type d'objet**. Définissez un ou plusieurs paramètres pour les blocs dans la ligne correspondante des **paramètres** – reportez-vous à la description des sous-sections ci-dessous pour connaître les paramètres pris en charge. Entrez la valeur du paramètre du bloc dans le champ approprié sous le nom du paramètre.

Les modifications ne seront exécutées que lorsque vous appuierez sur le bouton **Enregistrer** situé dans le coin supérieur droit de la page.

Cliquez sur Enregistrer pour enregistrer les modifications. Si le réglage de la fonction est correct, les informations sont affichées dans un champ vert. S'il est incorrect (nom / valeur non-valide ou paramètre obligatoire manquant), les informations d'erreur sont affichées dans un champ rouge. Les valeurs incorrectes sont marquées de l'icone  et le nom du paramètre sera en rouge. **L'automatisation** ne fonctionne que si tous les blocs sélectionnés sont configurés correctement. Sinon, elle passera sur le statut erreur.

La plupart des blocs incluent des paramètres (Event, Condition, StartEvent, par exemple) qui font référence à d'autres blocs. Pour interconnecter les blocs, cliquez sur la sortie d'un bloc et faites-la glisser vers l'entrée d'un autre bloc.

**Conseil**

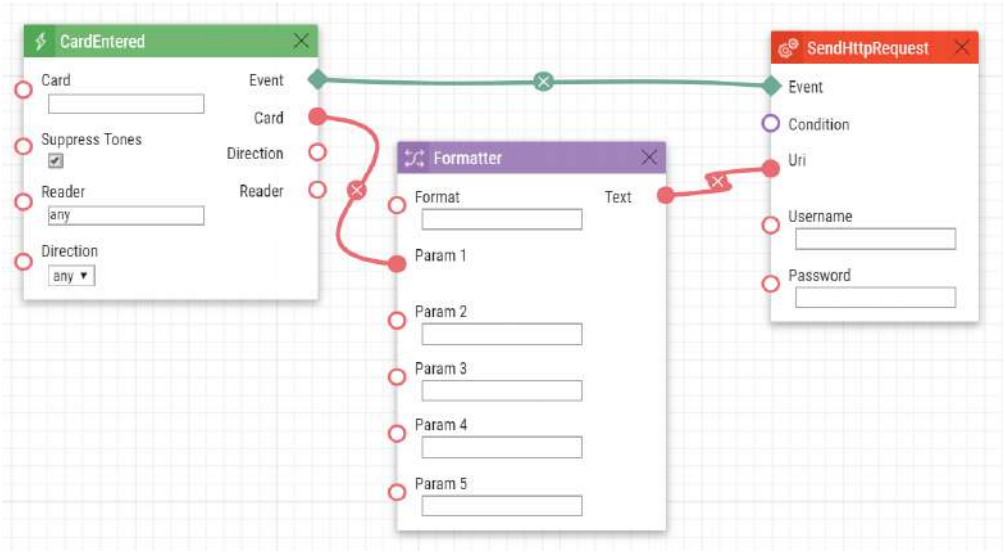
- Il n’est pas nécessaire de respecter les majuscules dans les noms de paramètre.
- Certains paramètres de bloc sont facultatifs. Si vous n'entrez pas ces paramètres facultatifs dans le bloc, la valeur par défaut sera appliquée.

## Utilisation des paramètres de sortie

Les paramètres de sortie du bloc événements permettent de transférer des informations supplémentaires entre les blocs. Par exemple, l’envoi de l'ID d’une carte détectée via HTTP vers

un autre appareil, l'utilisation des paramètres reçus via HTTP pour définir les paramètres d'une action liée…etc.

Vous pouvez définir ces informations via le bloc **« Formatter »**. Les valeurs des paramètres de sortie seront mises à jour chaque fois qu'un événement est généré. Une valeur peut également être utilisée pour d'autres blocs grâce à l'interconnexion.




Pour déplacer un paramètre de sortie d'Evénements vers le **« Formatter »**, connectez le paramètre de sortie à Param1. Ce paramètre n'est disponible à l'adressage que dans le format suivant : *{number.outputParameter}*.

- Exemple d'utilisation du Formatter pour le transfert de l'identifiant d'une carte :
  - Format : https://ip_address/card={B1.TimeStamp}
  - Param1 : connecté à la sortie du bloc de carte "CardEntered"
  - Text : connecté à l'entrée URI du bloc "SendHttpRequest"

Chaque événement définit les paramètres de sortie **TimeStamp** (horodatage) et **Count** (décompte).

**TimeStamp** contient la date et l'heure codées de la dernière fois que l'événement a été généré au format Unix Time (deuxième compte à partir de 00:00:00 1.1.1970).

**Count** contient le nombre de fois où l'événement a été généré après le démarrage du périphérique ou le dernier changement de configuration du bloc. Le paramètre de sortie augmente de 1 à chaque fois qu'un événement est généré.

Reportez-vous aux sous-sections suivantes pour découvrir davantage de paramètres de sortie avec des fonctions spécifiques.

**Conseil**

- Les majuscules / minuscules ne sont pas respectée dans les noms des paramètres de sortie.

**Observation**

- Vous ne pouvez pas utiliser les paramètres de sortie dans la relation de bloc définissant les paramètres, c.-à-d. Événement, condition…etc.

**Observation**

**OBSERVATION**
Afin d'assurer le bon fonctionnement et la garantie des résultats, nous recommandons fortement une vérification de la version du firmware du produit ou de l'installation au cours du processus d'installation. Le client prend en considération le fait que le produit ou l'installation peut atteindre les rendements garantis et être pleinement opérationnel conformément aux instructions du producteur en utilisant la version la plus récente du produit ou de l'installation, qui a été testée pour une interopérabilité totale. Les versions les plus récentes sont disponibles sur le site https://www.2n.com/cs_CZ/, ou des fonctionnalités spécifiques, en fonction de leur capacité technique, permettent une mise à jour dans l'interface de configuration. Si le client était amené à utiliser une autre version du produit ou de l'installation que la plus récente ou la version que le fabricant a jugée incompatible avec certaines versions des produits des installations d'autres fabricants ou le produit ou l'installation d'une manière incompatible avec les instructions du fabricant, les lignes directrices, le manuel ou la recommandation ou en conjonction avec des produits ou des installations inappropriés des autres producteurs, il est conscient de toutes les limitations potentielles de la fonctionnalité d'un tel produit ou d'une telle installation et de toutes les conséquences connexes. Si le client était amené à utiliser une version autre que la version la plus récente du produit ou de l'installation, ou la version qui a été déterminée par le fabricant comme étant incompatible avec certaines versions des produits des installations d'autres fabricants ou le produit ou l'installation dans un manière incompatible avec les instructions du fabricant, les directives, le manuel ou la recommandation ou en association avec des produits ou des installations inappropriés des autres fabricants, il accepte que la société 2N TELEKOMUNIKACE décline toute responsabilité quant à la limitation de la fonctionnalité d'un tel produit, ni à aucun dommage, perte ou dommage lié à une telle limitation potentielle de fonctionnalité.

# 3. Evènements

L'interface **d'automatisation** vous permet de définir certains types d'événement pouvant survenir sur votre appareil :

- **AudioLoopTest** – Test de la boucle audio effectué
- **CallStateChanged** – Changement de l'état d'appel
- **CardEntered** – Carte RFID entrée sur le Lecteur
- **CardHeld** – Carte RFID maintenue sur le Lecteur
- **CodeEntered** – Code composé sur le clavier
- **Counter** – comptage des déroulements d'événements
- **Delay** – Délai défini
- **DoorOpenTooLong** – Porte restée ouverte trop longtemps
- **DtmfEntered** – Code DTMF composé pendant un appel
- **DtmfPressed** – Code DTMF reçu pendant un appel
- **ErrorStateChanged** – changement de l'état d'erreur (uniquement pour 2N LiftIP 2.0)
- **FingerEntered** – Empreinte digitale scannée sur le Lecteur Biométrique
- **HttpTrigger** – Réception d'une commande HTTP/HTTPS
- **InputChanged** – Changement d'état de l'entrée logique
- **KeyPressed** – Bouton pressé
- **KeyReleased** – Bouton relâché
- **LicensePlateRecognized** – Plaque d'immatriculation enregistrée
- **LockdownStateChanged** – Changement d'état du mode Verrouillage d'urgence
- **MobKeyEntered** – Authentification via le Lecteur Bluetooth
- **MotionDetected** – Mouvement détecté par la caméra
- **MulticastTrigger** – Réception d'une commande multicast
- **OnvifVirtualOutputChanged** – Evènement reçu depuis un VMS
- **NoiseDetected** – Bruit détecté par le Microphone
- **RegistrationStateChanged** – Changement dans l'état d'enregistrement du compte SIP
- **Rebooted** – Détection du redémarrage de l'appareil
- **RescueTerminated** – fin du mode de libération (uniquement pour 2N LiftIP 2.0)
- **SilentAlarm** – Activation de l'alarme silencieuse
- **Time** – Heure spécifique
- **Timer** – Minuterie d'évènement périodique
- **UnauthorisedDoorOpen** – Ouverture de porte non-autorisée
- **UserAuthorized** – Authentification d'un utilisateur
- **OutputChanged** – Changement d'état d'un sortie
- **SwitchChanged** – événement sur la sortie

Voir ci-dessous pour plus de détails sur les événements, leurs paramètres d'entrée et leur utilisation.

## AudioLoopTest (Test de la Boucle Audio)

Le Bloc **AudioLoopTest** définit l'événement généré après le test du haut-parleur et du microphone (test de boucle audio). Les actions qui suivent seront exécutées en fonction du résultat du test.

### Paramètres entrants

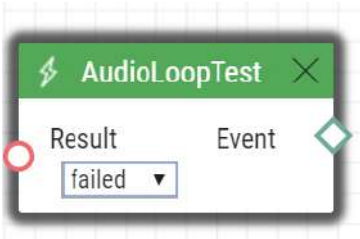
- **Result** – ce paramètre spécifie le résultat de test souhaité.
  - Valeurs valide :
    - **any** – l'événement est généré chaque fois que le test est effectué (quel que soit son résultat)
    - **passed** – l'évènement est généré lorsque le test est réussi
    - **failed** – l'évènement est généré lorsque le test a échoué
  - Ce paramètre est optionnel, la valeur par défaut est **failed**.

### Paramètres sortants

- **Event** – la sortie de l'Evènement permet de connecter une Action souhaitée.

### Exemple

Un événement est généré après le test de boucle audio si le résultat du test est négatif (c'est-à-dire que le microphone ou le haut-parleur est en panne) :




## CallStateChanged (Changement de l'état de l'appel)

Le bloc **CallStateChanged** définit l'événement généré par un changement d'état de l'appel (Sonnerie en cours, communication établie, appel terminé…etc.).

### Paramètres entrants

- **State** – indique le changement d'état de l'appel.
  - Valeurs valides :
    - **ringing** – sonnerie en cours
    - **connected** – communication établie
    - **terminated** – appel terminé
- **Direction** – définit la direction de l'appel.
  - Valeurs valides :
    - **incoming** – appel entrant
    - **outgoing** – appel sortant
    - **any** – les deux directions
  - Ce paramètre est optionnel, la valeur par défaut est **any**.
- **Number** – définit le numéro d'appel (numéro de téléphone, adresse IP) spécifique qui une fois appelé déclenchera l'événement. Saisissez plusieurs numéros séparés par des virgules si nécessaire. Une valeur non complétée sera équivalente à la valeur **any**.
  - Ce paramètre est optionnel, la valeur par défaut est **any**.

### Paramètres sortants

- **Event** – la sortie de l'Evènement permet de connecter une Action souhaitée.
- **State** – l'état d'appel détecté qui a généré cet événement. Les options correspondent au paramètre State.
- **Direction** – la destination d'appel qui a généré cet évènement. Appel entrant ou sortant.
- **Uri** – la sortie contient l'URI SIP complet de l'interlocuteur.
- **User ID** – l'identification de l'utilisateur qui a généré l'évènement.
- **User Name** – nom de l'utilisateur qui a généré l'évènement.

### Exemple

Evènement généré par la réponse à un appel entrant provenant du numéro 1234 :


## CardEntered (Carte RFID entrée sur le Lecteur)

Le bloc **CardEntered** définit l'événement généré en badgeant une carte RFID avec un ID défini (pour les modèles équipés d'un lecteur de carte RFID uniquement).

### Paramètres entrants

- **Card** – définit l'identifiant de la carte RFID ; reportez-vous à la sous-section Lecteur de carte du **Manuel de configuration**.
  - Valeurs valides :
    - **valid** – n'importe quelle carte valide (cartes dans le répertoire de l'interphone)
    - **invalid** – n'importe quelle carte invalide
    - **any** – n'importe quelle carte valide ou invalide
    - **< > (valeur nulle)** – l'évènement ne sera pas généré
    - Ou bien, compléter manuellement l'ID de la carte, **(plusieurs éléments séparés par une virgule peuvent être saisis)**.
- **Suppress Visual Signalin** – il permet de supprimer la signalisation visuelle (signalisation LED ou notifications sur l'écran) associée à la réception d'une carte invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimées (valeur par défaut)
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la réception d'une carte invalide.  Le paramètre est facultatif.

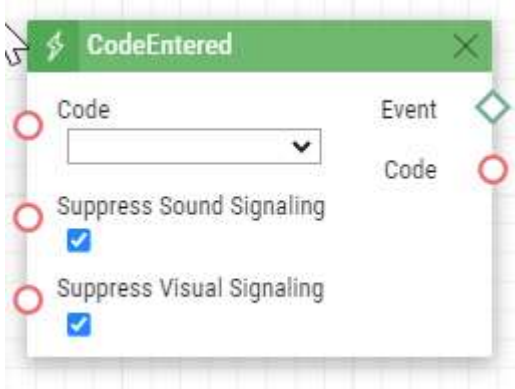
    - Valeurs valides :
        - **disabled** – les signalisations ne sont pas supprimées
        - **enabled** – les signalisations sont supprimées (valeur par défaut)
- **Reader** – permet de définir le lecteur ou module lecteur à utiliser.
    - Valeurs valides :
        - **internal_cardreader** – lecteur de carte interne (**2N IP Vario**, **Force**)
        - **external_cardreader** – lecteur de carte externe (**2N IP Vario**, **Force**)
        - **any** – n'importe quel lecteur / module
        - Ou bien, compléter manuellement le nom du module comme indiqué dans les paramètres de la section Hardware / Extendeurs / Modules / Menu des modules utilisés (**2N IP Verso**).
    - Ce paramètre est optionnel et la valeur par défaut est **any**.
- **Direction** – définit la direction de lecture.
    - Valeurs valides :
        - **in** – lecteur en entrée
        - **out** – lecteur en sortie
        - **any** – les deux directions
    - Ce paramètre est optionnel et la valeur par défaut est **any**.

## Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Card** – l'ID de la carte détectée qui a été la dernière à déclencher l'évènement ou l'action.
- **Direction** – configurez la direction du lecteur de carte (**in**, **out**, **any**).
- **Reader** – nom du module ou lecteur qui a été utilisé (**internal_cardreader**, **external_cardreader**, **<module_name>**).

## Exemple

Evènement généré après l'entrée de la carte ayant l'ID 0012456 :



## CardHeld

Le bloc **CardHeld** définit l'événement généré en maintenant une carte RFID avec un identifiant défini (pour les modèles équipés d'un lecteur RFID). L'évènement est généré en maintenant une carte RFID sur le lecteur pendant 4 secondes.

### Paramètres entrants

- **Card** – définit l'identifiant de la carte RFID ; reportez-vous à la sous-section Lecteur de carte du **Manuel de configuration**.
  - Valeurs valides :
    - **valid** – n'importe quelle carte valide (cartes dans le répertoire de l'interphone)
    - **invalid** – n'importe quelle carte invalide
    - **any** – n'importe quelle carte valide ou invalide
    - **< > (valeur nulle)** – l'évènement ne sera pas généré
    - Ou bien, compléter manuellement l'ID de la carte, **(plusieurs éléments séparés par une virgule peuvent être saisis)**.
- **Suppress Visual Signalin** – il permet de supprimer la signalisation visuelle (signalisation LED ou notifications sur l'écran) associée à la réception d'une carte invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - disabled – les signalisations ne sont pas supprimées
    - enabled – les signalisations sont supprimées (valeur par défaut)
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la réception d'une carte invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - disabled – les signalisations ne sont pas supprimées
    - enabled – les signalisations sont supprimées (valeur par défaut)
- **Reader** – permet de définir le lecteur ou module lecteur à utiliser.
  - Valeurs valides :
    - **internal_cardreader** – lecteur de carte interne (**2N® IP Vario**, **Force**)
    - **external_cardreader** – lecteur de carte externe (**2N® IP Vario**, **Force**)
    - **any** – n'importe quel lecteur / module.
    - Ou bien, compléter manuellement le nom du module comme indiqué dans les paramètres de la section Hardware / Extendeurs / Modules / Menu des modules utilisés (**2N® IP Verso**).
  - Ce paramètre est optionnel et la valeur par défaut est **any**.
- **Direction** – définit la direction de lecture.
  - Valeurs valides :
    - **in** – lecteur en entrée
    - **out** – lecteur en sortie
    - **any** – les deux directions

- Ce paramètre est optionnel et la valeur par défaut est **any**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Card** – l'ID de la carte détectée qui a été la dernière à déclencher l'évènement ou l'action.
- **Reader** – nom du module ou lecteur qui a été utilisé (**internal_cardreader**, **external_cardreader**, **<module_name>**).
- **Direction** – configurez la direction du lecteur de carte (**In**, **Out**, **Unspecified**).

### Exemple

Evènement généré après avoir maintenu la carte ayant l'ID 0012456 :


## CodeEntered

Le bloc **CodeEntered** définit l'événement généré par la saisie d'un code numérique et confirmation avec la touche * (pour les modèles à clavier numérique uniquement).

### Paramètres entrants

- **Code** – définit le code numérique.
  - Valeur valide :
    - **Code numérique** – 12345, e.g., **(plusieurs éléments séparés par une virgule peuvent être saisis)**.

    - **valid** – n'importe quel code valide
    - **invalid** – n'importe quel code invalide
    - **any** – n'importe quel code, valide ou invalide
    - **< > (Valeur nulle)** – l'évènement ne sera pas généré
- **Suppress Visual Signalin** – il permet de supprimer la signalisation visuelle (signalisation LED ou notifications sur l'écran) associée à la réception d'un code numérique invalide. Le paramètre est facultatif.
    - Valeurs valides :
        - **disabled** – les signalisations ne sont pas supprimées
        - **enabled** – les signalisations sont supprimées (valeur par défaut)
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la réception d'un code numérique invalide. Le paramètre est facultatif.
    - Valeurs valides :
        - **disabled** – les signalisations ne sont pas supprimées
        - **enabled** – les signalisations sont supprimées (valeur par défaut)

## Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Code** – le code numérique reçu qui a été le dernier à générer l'évènement.

## Exemple

Evènement généré en entrant le code 12345* sur le Clavier :

## Counter

Le bloc **Counter** suit le nombre de déroulements des événements définis précédents. Déroulement des événements entrant en tant qu'**Increase** augmentation du nombre. Déroulement des événements entrant en tant que **Decrease** diminution du nombre. Les événements diminuant le nombre ne doivent pas être configurés. Le bloc peut générer un événement sortant ou une action soit à chaque fois quand l'événement suivi survient, soit seulement lorsque le nombre obtient/dépasse la valeur définie dans **Max Value**.

### Paramètres entrants

> **Note**
>
> Si la configuration d'un des paramètres entrants de ce bloc se modifie lors du déroulement d'une fonction automatique, le nombre suivi revient automatiquement à la valeur **Min Value**.

- **Increase** – définit l'événement, dont un déroulement augmente le nombre suivi.
- **Decrease** – définit l'événement, dont un déroulement diminue le nombre suivi.
- **Reset** – définit l'événement après lequel le nombre suivi revient à la valeur **Min Value**.
- **Condition –** définit la condition qui devra être remplie pour que la modification du nombre suivi s'applique (Increase, Decrease, Reset).
- **Mode –** définit le mode d'initiation de l'événement.
  - Valeurs valides :
    - **always** (default) – le bloc entraîne l'événement à chaque modification du nombre suivi.
    - **condition** – le bloc entraîne l'événement dès que le nombre suivi atteint la valeur **Max Value** ou dépasse cette valeur.
- **Min Value –** définit la valeur initiale du nombre suivi. Il s'agit de la valeur possible la plus basse que ne diminue ni même un autre événement **Decrease**.
- **Max Value –** définit la valeur maximale du nombre suivi. Le comportement suivant est décrit dans **AutoReset**.
  - Valeurs valides : **Max value** doit être supérieur à **Min Value** !
- **Step** – définit le pas d'augmentation ou de diminution du nombre suivi.
  - Valeurs valides : chiffres positifs (>0)
- **Auto Reset –** définit si le nombre suivi revient automatiquement à la valeur initiale **Min Value** lors de l'obtention ou du dépassement de la valeur **Max Value**. Si n'est pas activé Auto Reset et que la valeur **Max Value** a été atteinte/dépassée, l'augmentation de la valeur du nombre de déroulements est interrompue (c'est-à-dire que les événements Increase sont ignorés).

### Paramètres sortants

- **Event** – sortie pour appeler l'Événement ou l'Action joint(e). La sortie s'active lors de l'obtention ou du dépassement de la valeur maximale Max Value (condition mode) ou à chaque fois que le bloc enregistre un des événements entrants (always mode).
- **Value** – contient le nombre actuel de déroulements suivis.

**Observation**

- **Min Value** doit être inférieur à **Max Value (Min Value < Max Value)**.
- Il convient de configurer les valeurs **Min Value**, **Max Value** et **Step** pour que la différence entre **Max Value** et **Min Value** soit un multiple entier de la valeur **Step.**

### Exemple

## Delay

Le bloc **Delay** définit le délai programmé après un autre évènement spécifique. Définissez cet événement pour retarder la réponse à un autre événement à l'aide d'un intervalle de temps défini (délai).

### Paramètres entrants

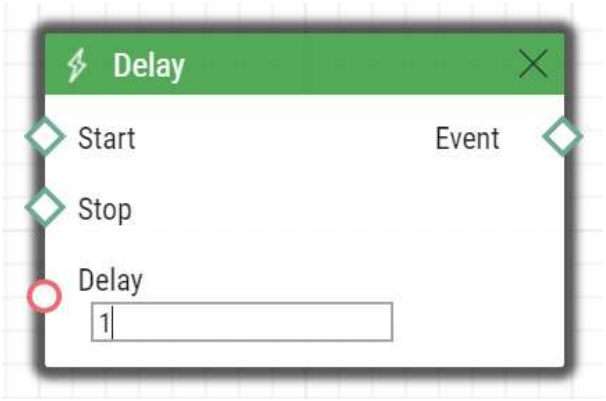
- **Start** – définit l'évènement qui déclenche le délai.
- **Stop** – définit l'évènement qui stoppe le délai. Ce paramètre est optionnel.
- **Delay** – définit le temps du délai. Il est seulement possible d'entrer une valeur numérique, il n'est pas possible d'entrer une valeur à partir d'un paramètre de sortie produit par d'autres événements.
- Exemple de valeurs valides :
  - **10** – 10 secondes (il n'est pas nécessaire d'entrer les unités)
  - **10s** – 10 secondes
  - **100ms** – 100 millisecondes

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Evènement généré 1s après qu'un premier évènement se soit produit :




## DoorOpenTooLong

Le bloc **DoorOpenTooLong** définit un évènement généré si une porte est restée ouverte plus longtemps que la durée préprogrammée dans l'interface de configuration.

### Paramètres entrants

- **State** – état du capteur de porte qui génère l'évènement.
  - **Start** – début de l'évènement
  - **End** – fin de l'évènement

- Valeur valide :

### Paramètres sortants

- **Event** – sortie permettant de générer l'événement ou l'action affectée.

### Exemple

La porte est restée ouverte trop longtemps par rapport à la programmation.

## DtmfEntered

Le bloc **DtmfEntered** définit un évènement généré en entrant un code DTMF sur un clavier suivie de la touche * lors d'un appel entrant ou sortant.

### Paramètres entrants

- **Code** – définit le code numérique.
  - Valeurs valides :
    - **Code numérique** – 12345, e.g., **(plusieurs éléments séparés par une virgule peuvent être saisis)**.
    - **valid** – n'importe quel code valide
    - **invalid** – n'importe quel code invalide
    - **any** – n'importe quel code, valide ou invalide
    - **< > (Valeur nulle)** – l'évènement ne sera pas généré
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la réception d'un code DTMF invalide. Ce paramètre est optionnel.
  - Valeurs valides :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimées (valeur par défaut)

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Code** – le code numérique reçu qui a été le dernier à générer l'évènement.

### Exemple

Evènement généré par la détection du code DTMF 123456*

## DtmfPressed

Le bloc **DtmfPressed** définit l'évènement généré par la détection d'un code DTMF prédéfini ou de n'importe quelles touche DTMF. La détection se fait pour les appels entrants et sortants.

### Paramètres entrants

- **Key** – définit le code DTMF (ou le groupe de code). Si ce paramètre n'est pas complété, l'évènement sera généré pour n'importe quel code DTMF détecté (valeur par défaut : Any).
    - Valeur valide :
- **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, *****, **#**, **A**, **B**, **C**, **D**
- **any** pour n'importe quel code (valeur par défaut).
- Séparez les valeurs par une virgule pour définir des groupes de code.
- **Direction** – définit la direction de l'appel.
    - Valeur valide :
        - **incoming** – appel entrant
        - **outgoing** – appel sortant
        - **any** – les deux directions
    - Ce paramètre est optionnel, la valeur par défaut est **any**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Pressed Key** – le code DTMF numérique reçu qui a été le dernier à générer l'évènement. Le DTMF est stocké dans le format de paramètre Key.

### Exemple

Evènement généré par la détection du code DTMF # :


## ErrorStateChanged (Changement de l'état d'erreur)

Le bloc **ErrorStateChanged** définit l'événement généré lorsque l'état d'erreur de 2N LiftIP 2.0 change.

### Paramètres entrants

- **Type** – définit le type de changement d'état de l'erreur.
  - Valeur valide :
    - **any** – n'importe lequel des types décrits ci-dessous
    - **button-error** – panne du bouton ALARM1

      > **Note**
      >
      > - Le dysfonctionnement d'un bouton est indiqué s'il est commuté plus longtemps que ce qui est configuré dans Hardware > Entrées numériques dans l'interface web de configuration de l'équipement.

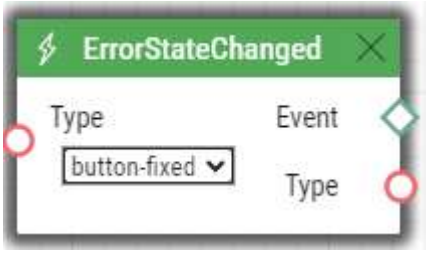
    - **button-fixed** – panne du bouton ALARM1 terminé
    - **audio-error** – le dernier test audio s'est terminé par une erreur audio
    - **audio-fixed** – l'erreur audio a été corrigée au cours du dernier test audio

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Type** – le type de changement d'état qui a déclenché l'événement.

### Exemple

Événement généré par la fin de la défaillance du bouton ALARM1.




## FingerEntered

Le bloc **FingerEntered** définit l'événement généré par la lecture d'une empreinte digitale connue sur le lecteur d'empreintes digitales (pour les appareils équipés d'un lecteur d'empreintes uniquement).

### Paramètres entrants

- **Fingerprint** – définit la validité d'une empreinte digitale scannée.
  - Valeur valide :
    - **valid** – empreinte digitale appartenant à un utilisateur
    - **invalid** – empreinte digitale n'appartenant à aucun utilisateur
    - **any** – n'importe quelle empreinte digitale
- **Finger** – définit une ou deux empreintes digitales pour un utilisateur.
  - Valeur valide :
    - **any** – n'importe quelle empreinte digitale d'un utilisateur
    - **F1** – l'empreinte digitale définie comme "F1" pour l'automatisation lors de l'enregistrement des empreintes
    - **F2** – l'empreinte digitale définie comme "F2" pour l'automatisation lors de l'enregistrement des empreintes
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la détection d'un utilisateur dont l'empreinte digitale est invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimées (valeur par défaut)

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **User ID** – l'identification de l'utilisateur qui a généré l'évènement.
- **User Name** – nom de l'utilisateur qui a généré l'évènement.

### Exemple

Evènement généré par l'entrée d'une empreinte digitale valide sur le lecteur biométrique.


## HttpTrigger

Le bloc **HttpTrigger** définit un évènement généré par la réception d'une commande HTTP/HTTPS depuis le serveur HTTP/HTTPS de l'interphone. Lorsque la commande http https://ip_addr/api/automation/trigger?triggerId=id est reçue, l'évènement sera généré pour l'ID correspondant à la valeur suivant "trigger" dans la commande HTTP/HTTPS. L'interphone envoie une réponse simple à cette requête (200 OK).

Pour envoyer une commande HTTP/HTTPS, vous devez autoriser le service Automation API et disposer d'un compte ayant accès à l'automatisation dans la section Services > HTTP API.

### Paramètres entrants

- **Name** – définit l'identifiant unique d'une commande HTTP/HTTPS intégrant des caractères alphabétiques et numériques.

### Paramètres sortant

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Params** – paramètres envoyés dans le bloc SendHttpRequest ou la commande arrivant sur l'Interphone IP 2N.

L'évènement HttpTrigger est toujours généré par la commande HTTP/HTTPS qui peut transporter une liste de paramètres d'entrée utilisateurs comme inclus dans la commande URI.

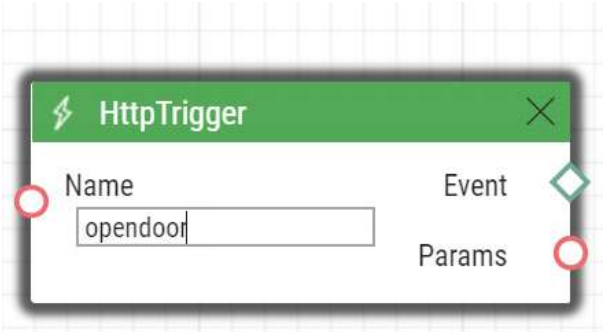
http://ip_address/api/automation/trigger?triggerId=id&param1=value1&param2=value2

La liste des paramètres entrants suit le caractère "?". Chaque paramètre doit inclure le nom et la valeur séparés par le caractère "=". Si la liste comprend plus d'un paramètre d'entrée, "&" est utilisé comme séparateur.

Les paramètres entrants de la commande HTTP/HTTPS reçue sont disponibles sous forme de bloc HttpTrigger Paramètres sortants. Le nom des paramètres sortants est égal au nom du paramètre transféré – $(line.param1) a $(line.param2).

### Exemple

Événement généré par la réception de la commande HTTPS suivante: https://ip_address/api/automation/trigger?triggerId=opendoor :




## InputChanged

Le bloc **InputChanged** définit un évènement généré par un changement d'état d'une entrée logique prédéfinie.

### Paramètres entrants

- **Input** – définit l'entrée logique.
  - Valeurs valides :
    - **tamper** – entrée du Commutateur d'autoprotection
    - **input1** – entrée logique 1
    - **input2** – entrée logique 2
    - **cr_input1** – entrée logique 1 du lecteur de carte
    - **cr_input2** – entrée logique 2 du lecteur de carte
  - Il peut y avoir différentes listes de valeurs valides en fonction des modèles d'**interphone IP 2N**; reportez-vous à la sous-section Entrées et sorties numériques disponibles dans le manuel de configuration.
- **Edge** – définit le changement détecté sur l'entrée logique.
  - Valeurs valides :
    - **falling** – front descendant, changement de. 1 à. 0
    - **rising** – front montant, changement de. 0 à. 1
  - Ce paramètre est optionnel, la valeur par défaut est **rising**.

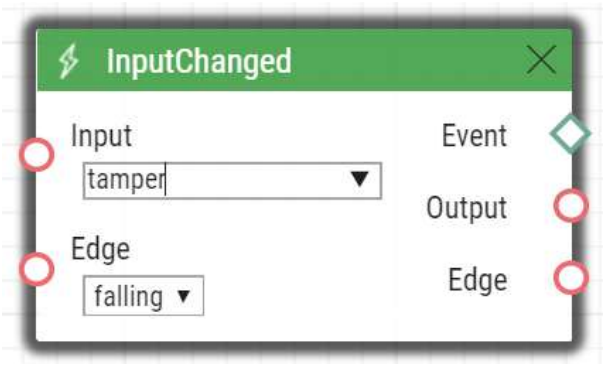
### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Output** – ID détecté de l'entrée dont le changement a été le dernier à générer cet événement. Les options correspondent aux valeurs des paramètres entrants.

- **Edge** – le changement de front détecté qui a été le dernier à générer cet évènement. Les options sont **falling** ou **rising**.

### Exemple

Evènement généré par la déconnection du Commutateur d'autoprotection (ouverture de l'appareil) :




## KeyPressed

Le bloc **KeyPressed** définit l'évènement généré par une pression sur un bouton ou un groupe de bouton prédéfini.

### Paramètres entrants

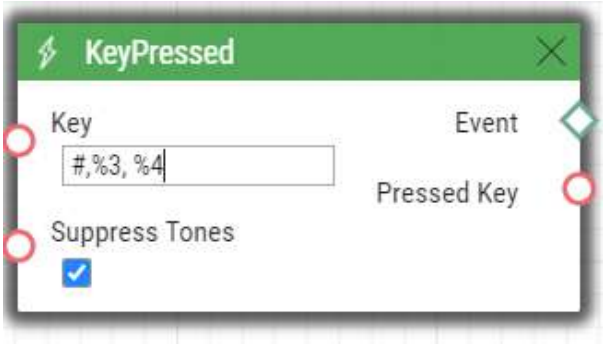
- **Key** – définit le bouton ou le groupe de boutons. Si ce paramètre n'est pas renseigné, l'événement est généré en appuyant sur un bouton quelconque (valeur par défaut : **Any**).
  - Valeurs valides :
    - **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, *****, **#** pour les touches du clavier numérique
    - **%1**, **%2**, .., **%999** pour les boutons d'appel
    - **T** – pour les touches de l'écran tactile
    - **B** – pour le contact initiant l'authentification Bluetooth
    - **alarm1** – pour le bouton ALARM1 sur l'appareil LiftIP 2.0
    - **alarm2** – pour le bouton ALARM2 sur l'appareil LiftIP 2.0
    - **vas_top** – pour le contact Voice Alarm Station sur le toit de la cabine
    - **vas_bottom** – pour Voice Alarm Station sous la cabine
    - **any** pour n'importe quel bouton (valeur par défaut)
  - Séparez les valeurs par une virgule pour définir plusieurs boutons.
- **SuppressTones** – permet de supprimer les signalisations audio et vidéo initiées par la pression sur un touche non programmée. Ce paramètre est optionnel.
  - Valeur valide :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimés (valeur par défaut)

## Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Pressed Key** – la touche enregistrée qui a été la dernière à générer cet événement. Le code de la touche est stocké dans le format de paramètre clé.

### Example

Evénement généré en appuyant sur la touche **#** et sur le bouton de numérotation rapide **3** ou **4** :




## KeyReleased

Le bloc **KeyReleased** définit l'événement généré en relâchant la touche définie ou les touches du groupe défini.

> **Note**
>
> - Pour le Modèle Vario : l'événement est généré chaque fois que le bouton est enfoncé, la fonctionnalité est identique à celle utilisée avec **KeyPressed**.

### Paramètres entrants

- **Key** – définit le bouton ou un groupe de boutons. Si ce paramètre n'est pas renseigné, l'événement est généré en appuyant sur un bouton quelconque (valeur par défaut : **any**).
  - Valeur valide :
    - **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, *****, **#** pour les boutons du clavier numérique
    - **%1**, **%2**, .., **%999** pour les boutons d'appel
    - **alarm1** – pour le bouton ALARM1 sur l'appareil LiftIP 2.0
    - **alarm2** – pour le bouton ALARM2 sur l'appareil LiftIP 2.0
    - **vas_top** – pour le contact Voice Alarm Station sur le toit de la cabine
    - **vas_bottom** – pour Voice Alarm Station sous la cabine
    - **any** pour n'importe quel bouton (valeur par défaut)
  - Séparez les valeurs par une virgule pour définir plusieurs boutons.

### Paramètres sortant

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

- **Released Key** – la touche enregistrée qui a été la dernière à générer cet événement. Le code de la touche est stocké dans le format de paramètre clé.

### Example

Evènement généré en relâchant la touche 1 et le bouton d'appel numéro 2 :


## LicensePlateRecognized

Le bloc **LicensePlateRecognized** définit l'événement généré par la réception d'une requête HTTP après la reconnaissance de la plaque d'immatriculation du véhicule.

### Paramètres entrants

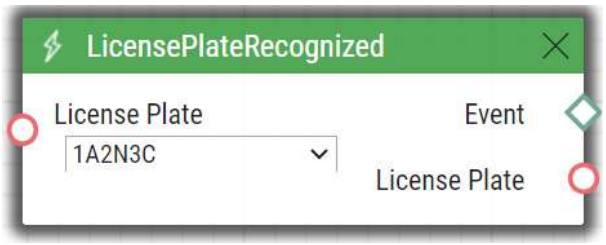
- **License Plate** – la plaque d'immatriculatio
  - Valeurs valides :
  - **license plate** – le texte pur de la plaque d’immatriculation
  - **valid** – toute plaque d'immatriculation valide (appartenant à l'utilisateur)
  - **invalid** – toute plaque d'immatriculation invalide (n'appartenant à aucun utilisateur)
  - **any** – toute plaque d'immatriculation

### Paramètres sortants

- **Event** – sortie générant l'Evènement/Action.
- **License Plate** – le texte pur de la plaque d’immatriculation

### Exemple

L’événement généré par la lecture de la plaque d'immatriculation 1A2N3C.


LicensePlateRecognized
License Plate
1A2N3C
Event
License Plate

## MobKeyEntered

Le bloc **MobkeyEntered** définit l'événement généré par la lecture d'une Clé Mobile connu sur un lecteur Bluetooth (pour les appareils équipés d'un lecteur Bluetooth uniquement).

### Paramètres entrant

- **MobKey** – définit la validé de la Clé Mobile.
  - Valeurs valides :
    - **valid** – la clé Mobile appartient à un utilisateur
    - **invalid** – la clé Mobile est inconnue
    - **any** – n'importe quelle clé Mobile
- **Suppress Visual Signalin** – il permet de supprimer la signalisation visuelle (signalisation LED ou notifications sur l'écran) associée à la réception d'une clé WaveKey Invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimées (valeur par défaut)
- **Suppress Sound Signaling** – il permet de supprimer la signalisation audio associée à la réception d'une clé WaveKey Invalide. Le paramètre est facultatif.
  - Valeurs valides :
    - **disabled** – les signalisations ne sont pas supprimées
    - **enabled** – les signalisations sont supprimées (valeur par défaut)

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **User ID** – l'identification de l'utilisateur qui a généré l'évènement.
- **User Name** – nom de l'utilisateur qui a généré l'évènement.

### Exemple

Evènement généré par l'utilisation d'un Clé Mobile Valide.


## MotionDetected

Le bloc **MotionDetected** définit un évènement généré par la détection d'un mouvement depuis la caméra de l'interphone. Ce mouvement ne peut être détecté que par la caméra interne au portier. Les paramètres entrant de la détection de mouvement peuvent être configurés dans la section Hardware / Camera interne / Paramètres de détection des mouvements.

### Paramètres entrants

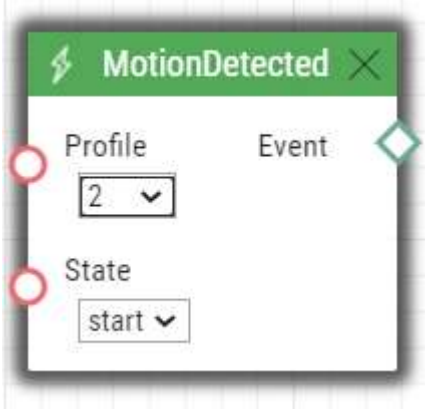
- **Profile** – définit le profil de détection de mouvement auquel cet événement répondra.
  - Valeurs valables:
    - **any** – mouvement détecté par l'un des profils de détection de mouvement
    - **1** – mouvement détecté par le profil de détection de mouvement 1
    - **2** – pohyb detekovaný profilem detekce pohybu 2
- **State** – définit si le début ou la fin d'un mouvement doit être détecté.
  - Valeurs valides :
    - **start** – début du mouvement
    - **end** – fin du mouvement
  - Ce paramètre est optionnel, la valeur par défaut est **start**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Événement généré lorsqu'un mouvement a commencé à être détecté par le profil de détection de mouvement 2.




## MulticastTrigger

Le bloc **MulticastTrigger** définit un évènement généré par la réception d'une commande envoyé par l'action du bloc **SendMulticastRequest**. La demande est un message envoyé par UDP à une adresse multicast (235.255.255.250:4433) et peut donc être reçue par plusieurs appareils en même temps. Le message comprend l'ID de commande (paramètre de commande) et des paramètres entrants facultatifs supplémentaires. Le message peut être sécurisé par un mot de passe (Paramètres Mot de passe).

### Paramètres entrants

- **Command** – définit l'ID de la commande pour distinguer les types de commande. Le bloc MulticastTrigger répond à l'action SendMulticastRequest uniquement si l'identifiant de commande est le même. Tout texte contenant les caractères A–Z, a–z et 0–9 peut être utilisé pour l'identification. Les majuscules / minuscules doivent être respectées dans le nom de la commande.
- **CheckTime** – activer / désactiver la vérification de l'heure de réception de la commande par rapport à la valeur de temps incluse dans le message de commande pour éliminer les attaques causées par la répétition d'un message déjà traité. L'heure synchronisée (via le serveur NTP) sur tous les appareils d'envoi et de réception de commandes est nécessaire pour permettre cette fonctionnalité.
  - Valeur valides :
    - **disabled** – l'heure de réception n'est pas vérifiée
    - **enabled** – l'heure de réception du message est vérifiée (sécurité accrue)
  - Ce paramètre est optionnel, la valeur par défaut est **0**.
- **Password** – définir un mot de passe pour sécuriser la commande contre tout accès non autorisé. Le mot de passe doit correspondre à la valeur définie dans l'action SendMulticastRequest à laquelle MulticastTrigger est censé répondre.

### Paramètres sortants

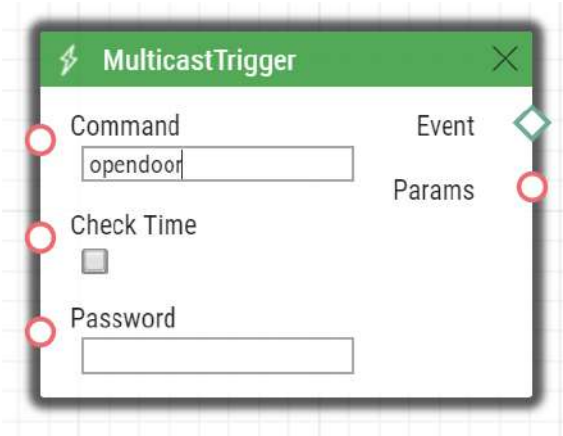
- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Params** – paramètres envoyés dans l'action SendMulticastRequest.

L'événement MulticastTrigger est généré chaque fois qu'une commande de masse comprenant la liste des paramètres entrants utilisateurs (paramètres Params, actions MulticastRequest) est reçue. Chacun des paramètres entrants a un nom unique défini par l'utilisateur et est disponible en tant que paramètre sortant du même nom dans le bloc MulticastTrigger.

Exemple : Supposons qu'une commande de masse générée par l'action MulticastRequest soit reçue, dans laquelle le Paramètre = "AAA = 123" est inclus. L'événement MulticastTrigger qui traite cette commande inclura automatiquement la valeur 123 pour le paramètre sortant AAA. Ce paramètre sortant peut être référencé dans les blocs interconnectés.

### Example

Événement généré par la réception d'une commande de masse opendoor :




## OnvifVirtualOutputChanged

Le bloc **OnvifVirtualOutputChanged** permet de transmettre un évènement depuis l'Interphone IP 2N vers un VMS (Logiciel de Gestion de Vidéosurveillance).

### Paramètres entrants

- **Port** – définit le port pour la VMS. Valeurs valides : 50–54.
- **Edge** – définit le changement détecté sur l'entrée virtuelle.
  - Valeurs valides :
    - **falling** – front descendant, changement de. 1 à. 0
    - **rising** – front montant, changement de. 0 à. 1

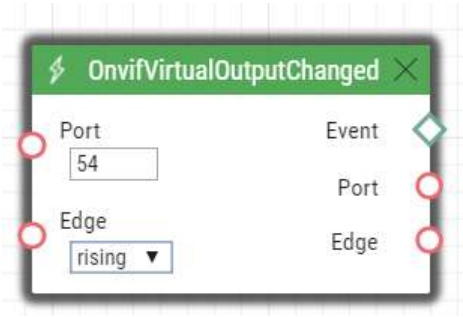
- Ce paramètre est optionnel, la valeur par défaut est **rising**.

### Paramètres sortants

- **Event** – evènement/Action activant la sortie
- **Port** – valeur du port changé depuis le VMS. Valeurs valides: 50–54.
- **Edge** – la dernière modification d'entrée virtuelle qui a généré cet événement. Valeurs valides: falling ou rising

### Exemple

L'événement généré lorsque la valeur du port virtuel change.




## NoiseDetected

Le bloc **NoiseDetected** définit un évènement généré en cas de détection d'un bruit. Un bruit peut être détecté par le microphone de l'interphone uniquement. Les paramètres de détection de bruit sont à configurer dans la section Hardware / Menu Audio, section paramètres de détection de bruit.

### Paramètres entrants

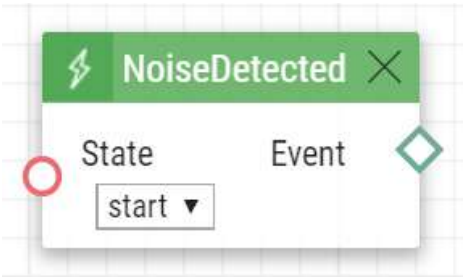
- **State** – définit si le début ou la fin d'un bruit doit être détecté.
  - Valeurs valides :
    - **start** – début du bruit
    - **end** – fin du bruit
  - Ce paramètre est optionnel, la valeur par défaut est **start**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Evènement généré par la détection d'un bruit (début du bruit).




## RegistrationStateChanged

Le bloc **RegistrationStateChanged** définit un évènement généré par le changement de statut de l'enregistrement à un compte SIP sur l'interphone. Vous pouvez définir les paramètres des comptes SIP dans la section Services / SIP 1 et SIP 2. L'enregistrement est modifié chaque fois que l'interphone est allumé, que la configuration est modifiée ou que la connexion au registrar SIP est perdue, par exemple.

### Paramètres entrants

- **Account Type** – sélectionner le compte souhaité
  - Valeurs valides:
    - **general** – comptes SIP à usage général
    - **msteams** – compte SIP sélectionné pour connexion avec MS Teams
    - **any** – n'importe quel compte SIP
- **Account** – sélectionner le compte SIP souhaité
  - Valeurs valides :
    - 1 – Compte 1
    - 2 – Compte 2
    - Any – n'importe quel compte
    - Ce paramètre est optionnel, la valeur par défaut est **Any**.
- **State** – définit le statut d'enregistrement qui génère l'évènement.
  - **Unregistered** – interphone non enregistré
  - **Registering** – enregistrement en cours
  - **Registered** – l'interphone est enregistré
  - **Unregistering** – désenregistrement en cours

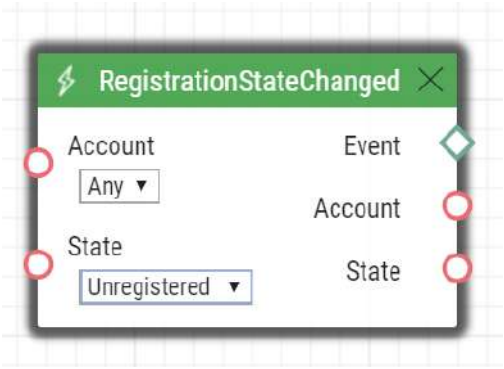
    - **Any** – n'importe quel statut
- Valeurs valides :
- Ce paramètre est optionnel, la valeur par défaut est **Any**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Account** – sélectionner le compte SIP pour lequel les évènements seront enregistrés.
- **State** – définir le statut d'enregistrement qui génère l'évènement.

### Exemple

Événement généré lorsque l'interphone a perdu la connexion avec le serveur SIP (chaque fois que le registraire n'a pas répondu à une demande d'inscription périodique) :




## Rebooted

Le bloc **Rebooted** définit un évènement généré par un redémarrage de l'appareil.

### Paramètres entrants

Ce bloc n'a aucun paramètre entrant.

### Paramètres sortants

- **Event** – sortie permettant de générer un évènement ou une action connectée
- **Time** – heure de redémarrage de l'appareil

- **Reason** – raison du redémarrage

### Exemple

Evènement généré par le redémarrage de l'appareil


## RescueTerminated

Le bloc **RescueTerminated** définit l'événement généré par la fin du mode de dégagement. Les méthodes de mode de dégagement sont définies dans la configuration de l'appareil **2N LiftIP2.0** dans la section Services > Mode de dégagement.

### Paramètres entrants

Ce bloc n'a pas de paramètre entrant.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Type** – méthode permettant de quitter le mode de dégagement.

### Exemple

Événement généré lorsque le bouton ALARM2 ou le mot de passe (en fonction de la configuration du dispositif) est utilisé pour quitter le mode de dégagement.


## SilentAlarm

Le bloc **SilentAlarm** définit l'évènement généré par l'activation de l'alarme silencieuse. L'alarme silencieuse peut être activée si quelqu'un a entré sur le clavier de l'interphone un code plus haut de 1 chiffre qu'un code valide. Par exemple, si le code est 123, l'alarme s'activera avec le code 124.

### Paramètres entrants

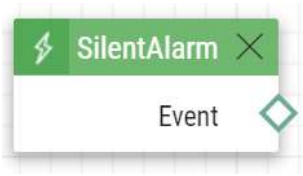
Ce bloc n'a pas de paramètre entrant.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Evènement généré en entrant le code 112 si l'un des codes valides est 111.




## Time

Le bloc **Time** définit un évènement généré tous les jours à une heure spécifique. Pour limiter la validité de cet événement à certains jours uniquement, utilisez la condition. ProfileState et assigné la à l'action générée. Il vous faudra spécifier les jours demandés dans le profil horaire utilisé.

### Paramètres entrants

- **Time** – définit l'heure à laquelle l'évènement est généré. Le format est hh:mm.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Evènement généré tous les jours à 17:30.


## Timer

Le bloc **Timer** définit un événement généré avec un délai préprogrammé après un autre événement spécifié et avec un nombre défini de répétitions. Définissez cet événement pour retarder la réponse à un autre événement d'un intervalle de temps défini ou pour exécuter la réponse à un évènement plusieurs fois.

### Paramètres entrants

- **Start** – définit l'événement de démarrage de la minuterie (c'est-à-dire le numéro de ligne dans l'onglet Automatisation sur lequel l'événement est défini). Ce paramètre est facultatif. Si aucune valeur n'est renseignée, la minuterie démarre automatiquement.
- **Stop** – définit l'évènement qui stoppe la minuterie (i.e. le numéro de ligne de l'onglet Automation sur lequel l'événement est défini). Lorsque StopEvent est opéré, la minuterie s'arrêtera et redémarrera par StartEvent uniquement. Ce paramètre est facultatif.
- **Period** – définit la durée du timer.
    - Exemple de valeurs valides :
        - **10** – 10 secondes (les unités ne sont pas nécessaires)
        - **10s** – 10 secondes

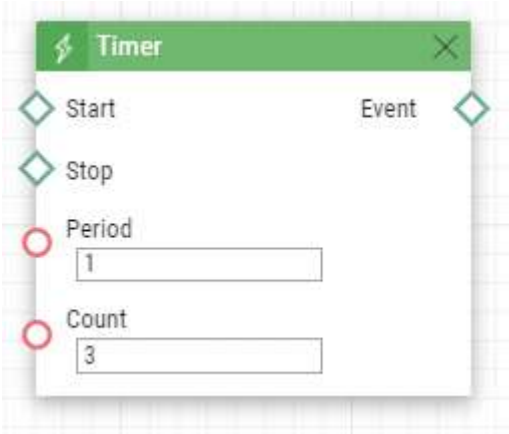
        - **100ms** – 100 millisecondes.
    - La durée minimum est **100ms**.
- **Count** – définit le nombre de répétitions. Le paramètre est facultatif et la valeur par défaut est 0, ce qui signifie que le nombre d'événements générés par le minuteur est illimité. La valeur 1 fait que la minuterie se comporte comme un retard.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Événement généré trois fois par intervalles de 1 s après la montée de l'événement sur la ligne 1 :




## UnauthorisedDoorOpen

Le bloc **UnauthorisedDoorOpen** définit un évènement généré lorsqu'une ouverture de porte non-autorisée est détectée.

### Paramètres entrants

- **State** – état du capteur de porte qui génère l'événement.
  - **Start** – début de l'évènement
  - **End** – fin de l'évènement

- Valeurs valides :

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.

### Exemple

Evénement généré lors d'une ouverture de porte non-autorisée :


## UserAuthorized

Le bloc **UserAuthorized** définit un évènement généré lors de l'authentification d'un utilisateur par n'importe quelle méthode d'accès (code, PIN, RFID, Bluetooth, empreinte digitale).

### Paramètres entrants

- **User** – définit l'utilisateur ou la liste d'utilisateurs. Si aucune valeur n'est complétée (valeur par défaut), l'utilisateur n'est pas important.

### Paramètres sortant

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **User ID** – l'identification de l'utilisateur qui a généré l'évènement.
- **User Name** – nom de l'utilisateur qui a généré l'évènement.

### Exemple

Evènement généré par l'authentification de l'utilisateur Victoria Black :

**Observation**

- Ce paramètre est limité à 10 utilisateurs.

## OutputChanged

Le bloc **OutputChanged** définit un évènement généré par le changement d'état d'une sortie.

### Paramètres entrant

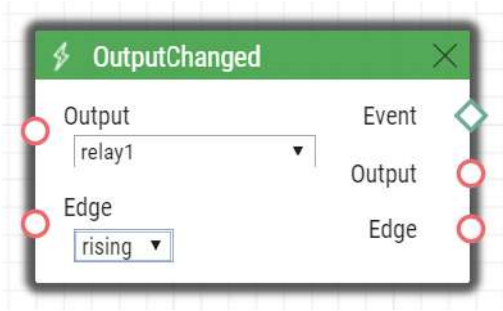
- **Output** – définit la sortie.
  - Valeurs valides :
    - **relay1** – relais 1 de l'unité principale
    - **relay2** – relais 2 de l'unité principale
    - **output1** – sortie 1 de l'unité principale
    - **output2** – sortie 2 de l'unité principale
  - La liste des valeurs valides peut être différente selon les modèles d'interphones, référez-vous à la sous-section Entrées et sorties disponibles.
- **Edge** – définit le changement d'état de la sortie.
  - Valeurs valides :
    - **falling** – front descendant, changement de. 1 à. 0
    - **rising** – front montant, changement de. 0 à. 1
  - Ce paramètre est optionnel, la valeur par défaut est **rising**.

### Paramètres sortants

- **Event** – le résultat de l'évènement sera de déclencher l'évènement ou l'action connectée.
- **Output** – ID détecté de l'entrée dont le changement a été le dernier à générer cet événement. Les valeurs disponibles correspondent aux valeurs d'entrée.
- **Edge** – changement détecté d'une sortie qui a été le dernier à générer un évènement. Les valeurs valides sont  falling ou rising.

### Exemple

Evénement généré par le changement de l'état de la sortie relais 1.




## SwitchChanged

Le bloc SwitchChanged génère un événement sur la sortie si l'événement programmé se produit.

### Paramètres d'entrée

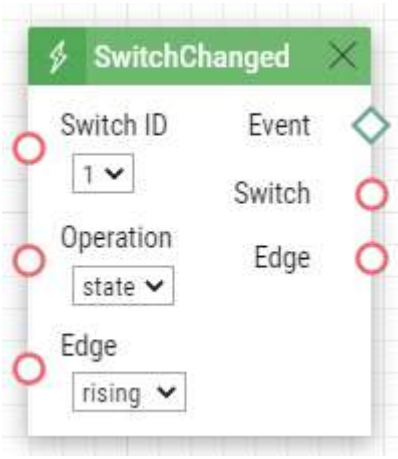
- ***Switch ID*** – définit quel commutateur est utilisé pour évaluer l'événement.
  - Valeurs valables :
    - où X est le numéro du commutateur correspondant (généralement 1... 4, différents modèles possèdent un nombre différent de commutateurs)
- ***Operation*** - définit le type d'opération du commutateur qui est utilisé pour évaluer l'événement.
  - Valeurs valables :
    - state – le commutateur peut être actif ou inactif (valeur par défaut)
    - lock – le commutateur peut être verrouillé (locked) ou déverrouillé (unlocked)
    - hold – le commutateur peut être maintenu (held) ou relâché (released)
- ***Edge***
  - Valeurs valables :
    - falling (descendante) – l'événement se produit si l'opération sélectionnée passe au niveau logique 0 (c'est-à-dire que le commutateur devient inactif, le commutateur est déverrouillé, le commutateur est relâché).
    - rising (croissante) – l'événement se produit si l'opération sélectionnée passe à la valeur logique 1 (c'est-à-dire que le commutateur devient actif, le commutateur devient verrouillé, le commutateur est maintenu) (valeur par défaut)

## Paramètres de sortie

- ***Event*** – un événement sur la sortie est généré lorsqu'un événement se produit sur l'entrée.
- **Switch** – identifiant détecté du commutateur qui a généré cet événement en dernier par son changement. Les options correspondent aux valeurs du paramètre Switch ID.
- **Edge** – changement de bord détecté qui a généré cet événement en dernier. Les options sont falling (baisse) ou rising (hausse).

## Exemple

Si le bloc est réglé sur Switch ID = Switch 1, Operation = hold, Edge = rising, un événement sera généré sur la sortie (la sortie ***Event*** déclenchera l'événement) si l'état du Switch 1 est « held » (maintenu).



# 4. Action

**Automatisation** – Permet de définir les types d'action suivants :

- **ActivateSwitch** – Activer l'interrupteur
- **ActiveWaveKey** – Activation de l'authentification Bluetooth
- **AnswerCall** – Répondre à un appel entrant
- **BeginCall** – Établir un appel sortant
- **ControlRtpStream** – Contrôler l'envoi du flux RTSP
- **EndCall** – Terminer un appel
- **LogAutomationEvent** – Enregistrement de l'évènement sur le serveur Syslog
- **OpenDoor** – Activation de la lumière et du son signalant l'accès via un lecteur de carte
- **PlayUserSound** – Jouer un message audio
- **SendDtmf** – Envoyer un code DTMF
- **SendEmail** – Envoyer un email
- **SendHttpRequest** – Envoyer une commande HTTP
- **SendMulticastRequest** – Envoyer une commande vers plusieurs appareils
- **SendWiegandCode** – Envoyer un code Wiegand
- **SetCameraInput** – Sélectionner l'entrée de la caméra
- **SetLed** – définit la couleur de la LED
- **SetOnvifVirtualInput** – Activer une entrée virtuelle ONVIF
- **SetOutput** – Paramétrer l'état de la sortie
- **SetSecuredState** – définit le niveau d'état de la condition SecuredState
- **StartAutoUpdate** – Mise à jour automatique du firmware et de la configuration
- **StartLiftCall** – Établir un appel (uniquement pour 2N LiftIP 2.0)
- **StartMulticastRecv** – Démarrer la réception d'un flux audio multicast
- **StartMulticastSend** – Démarrer l'envoi d'un flux audio multicast
- **StopMulticastRecv** – Interrompre la réception d'un flux audio multicast
- **StopMulticastSend** – Interrompre l'envoi d'un flux audio multicast
- **UploadSnapshotToFtp** – Capture d'une image chargée vers le serveur FTP

## ActivateSwitch

Le bloc **ActivateSwitch** définit l'action permettant l'activation de l'interrupteur de l'Interphone comme défini dans les onglets Interrupteurs 1–4. L'activité à effectuer dépend entièrement des paramètres spécifiques de l'interrupteur (activation de la sortie logique, envoi d'une commande HTTP, etc.). La désactivation des interrupteurs est également contrôlée par les réglages de ces derniers.

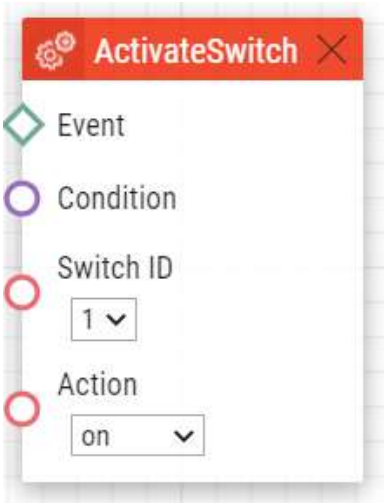
### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Switch ID** – définit l'interrupteur à activer (1–4).

- **State** – définit l'état de l'interrupteur en mode bistable. Ce paramètre ne s'applique pas lorsque l'interrupteur est en mode monostable.
    - Valeurs valides :
        - **on** – l'interrupteur est activé
        - **off** – l'interrupteur est désactivé
        - **toggle** – l'interrupteur est basculé
        - **lock** – l'interrupteur est verrouillé.
        - **unlock** – l'interrupteur est déverrouillé.
        - **hold** – l'interrupteur est maintenu.
        - **release** – l'interrupteur est relâché.
    - Ce paramètre est optionnel, la valeur par défaut est **on**.

### Exemple

L'interrupteur 1 s'activera si l'évènement définit à la deuxième ligne se produit et si la condition définit à la troisième ligne est respectée :




## ActivateWaveKey

Le bloc **ActionWaveKey** définit l'action requise pour activer le processus d'authentification WaveKey.

### Paramètres

- **Event** – définit l'événement qui déclenche l'action.
- **Condition** – définit la condition qui doit être remplie pour que l'action soit exécutée. Ce paramètre est optionnel.


## SetOutput

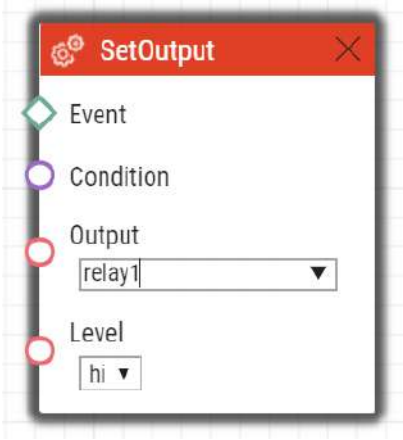
Le bloc **SetOutput** définit l'action permettant d'activer les sorties de l'interphone sur l'état demandé.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Output** – définit la sortie à activer.
  - Valeurs valides :
    - **relay1** – sortie relais 1 sur l'unité principale
    - **relay2** – sortie relais 2 sur l'unité principale
    - **output1** – sortie 1 sur l'unité principale
    - **output2** – sortie 2 sur l'unité principale
  - La liste des valeurs valides peut varier selon les différents modèles de portier 2N; référez-vous à la sous-section Sorties et entrées numériques disponibles.
- **Level** – définit l'état requis de la sortie. Ce paramètre est optionnel.
  - Valeurs valides :
    - **lo** – désactivation de la sortie
    - **hi** – activation de la sortie (valeur par défaut)

### Exemple

Activation de la sortie relais 1 si l'évènement défini à la ligne 2 survient :



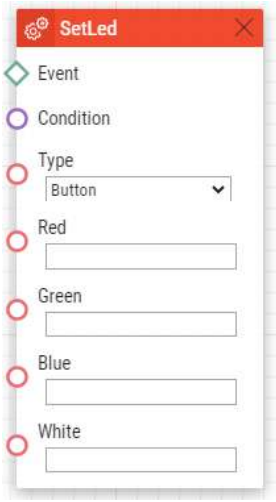

## SetLed

Le bloc **SetOutput** définit une action qui contrôle la modification du rétroéclairage LED ou du jeu de LED sur l'appareil. Après le redémarrage de l'appareil, le rétroéclairage défini dans l'interface de configuration Web de l'appareil sera rétabli.

### Paramètres

- **Event** – définissez l'événement qui déclenche l'action.

- **Condition** – définissez la condition qui doit être remplie pour que l'action soit lancée. Ce paramètre est facultatif.
- **Type** – définit la LED ou le jeu de LED à contrôler.
- **Red** – définit l'intensité de la couleur rouge dans une plage de 0 à 255. Une valeur vide est considérée comme 0 si au moins un autre paramètre de couleur est indiqué. Si tous les paramètres de couleur sont vides, le bloc désactive le paramètre de couleur (c'est-à-dire que la couleur revient à la couleur définie dans la configuration).
- **Green** – définit l'intensité de la couleur vert dans une plage de 0 à 255. Une valeur vide est considérée comme 0 si au moins un autre paramètre de couleur est indiqué. Si tous les paramètres de couleur sont vides, le bloc désactive le paramètre de couleur (c'est-à-dire que la couleur revient à la couleur définie dans la configuration).
- **Blue** – définit l'intensité de la couleur bleu dans une plage de 0 à 255. Une valeur vide est considérée comme 0 si au moins un autre paramètre de couleur est indiqué. Si tous les paramètres de couleur sont vides, le bloc désactive le paramètre de couleur (c'est-à-dire que la couleur revient à la couleur définie dans la configuration).
- **White** – définit l'intensité de la couleur blanc dans une plage de 0 à 255. Une valeur vide est considérée comme 0 si au moins un autre paramètre de couleur est indiqué. Si tous les paramètres de couleur sont vides, le bloc désactive le paramètre de couleur (c'est-à-dire que la couleur revient à la couleur définie dans la configuration).




## SetSecuredState

Le bloc **SetSecuredState** définit l'action nécessaire pour régler l'état Secured State au niveau souhaité. Le bloc ne fonctionne que lorsque l'entrée d'état sécurisée attribuée est réglée sur "Aucune" (c'est-à-dire qu'aucune entrée numérique n'est attribuée au contrôle d'état).

### Paramètres

- **Event** – définissez l'événement qui déclenche l'action.

- **Condition** – définissez la condition qui doit être remplie pour que l'action soit lancée. Ce paramètre est facultatif.
- **Level** – définissez le niveau de sortie souhaité.
  - Valeurs valables :
    - **lo** – état de désactivation
    - **hi** – activation de l'état (valeur par défaut).


## BeginCall

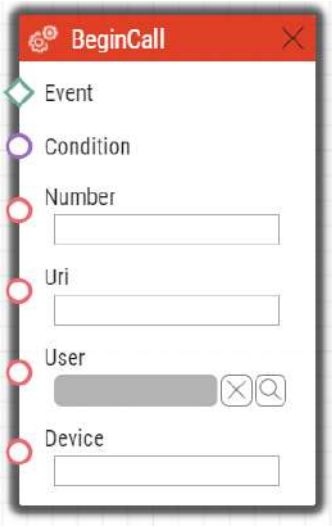
Le bloc **BeginCall** définit l'action permettant d'établir un appel sortant vers une destination d'appel, un numéro de téléphone, un SIP URI ou le numéro d'un utilisateur présent dans le répertoire téléphonique du portier.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Number** – définit le numéro à appeler (si le portier 2N est connecté à un IP PBX).
- **Uri** – définit le SIP URI à appeler : sip:utilisateur@domaine. Un URI particulier est utilisé pour les appels sur Axis Camera Station sous la forme **vms: ***.
- **User** – définit l'utilisateur à appeler.
- **Device** – définit l'application **2N® IP Mobile** à appeler : device_name.
- Il vous faut uniquement définir l'un de ces paramètres (**Number**, **Uri**, **User** ou **Device**).

### Exemple

Un appel sortant s'établit si l'évènement défini à la ligne 2 survient :




## AnswerCall

Le bloc **AnswerCall** définit l'action permettant de répondre à un appel entrant. Si aucun appel n'est généré ou que l'appel ne sonne pas, cette action ne se surviendra pas.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.

### Exemple

Réponse à un appel si l'évènement défini à la ligne 2 survient :


## EndCall

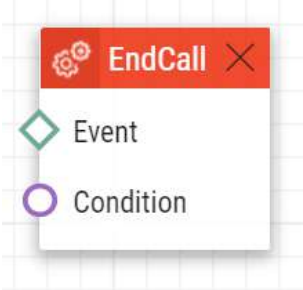
Le bloc **EndCall** définit l'action permettant de terminer un appel en cours. Si aucun appel est en cours lors de l'activation de cette action par un évènement, rien ne se produira.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.

### Exemple

Un appel en cours s'interrompra si l'évènement défini à la ligne 2 survient :




## SendHttpRequest

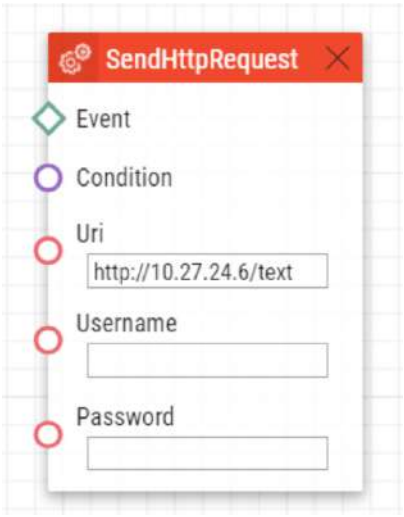
Le bloc **SendHttpRequest** définit l'action permettant d'envoyer une commande HTTP vers un autre dispositif réseau. Les commandes HTTP vous permettent de contrôler d'autres appareils sur le réseau (Web Relais, système d'enrgistrement, autres portiers...Etc.)

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Uri** – définit l'HTTP URI standard en incluant l'adresse de destination, optionnellement, le chemin à suivre ainsi que d'autres paramètres. La longueur maximum est de 2048 bytes.
- **Username** – définit le nom de l'utilisateur si une autorisation est requise par le serveur HTTP. Ce paramètre est optionnel, la valeur par défaut est "intercom".
- **Password** – définit le mot de passe si une autorisation est requise par le serveur HTTP. Ce paramètre est optionnel.
- **Method** – définit la méthode de la requête HTTP : **GET**, **POST**, **PUT**, **DELETE**.
- **Type** – sélectionnez le type de contenu du corps de la requête HTTP : "application/json" ou "text/plain". Appliquez le seulement aux méthodes valides **POST** et **PUT**.
- **Text** – sélectionnez le contenu du texte de la requête. Appliquez le seulement aux méthodes valides **POST** et **PUT**.

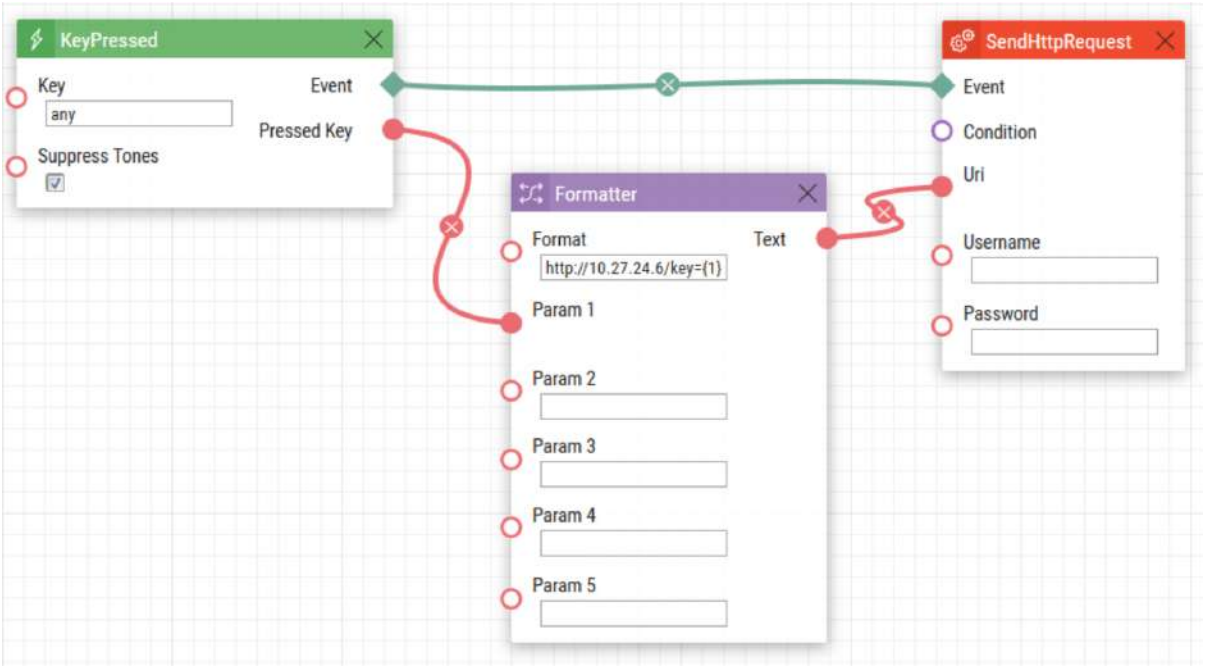
**Exemple 1)**

Lorsqu'un évènement connecté à l'action est généré, une commande HTTP est envoyé à l'adresse : 10.27.24.6 :




**Exemple 2)**

Lorsqu'un bouton du portier est pressé, son identification exacte est envoyée à l'adresse suivante : 10.27.24.6 :



**Observation**

- Les authentifications Basic et Digest sont prises en charge, nous recommandons Digest pour plus de sécurité.

**Observation**

Les commandes HTTP utilisent le codage URL. Dans l'exemple d'automatisation suivant :

1. Event.KeyPressed: Key=Any
2. Action.SendHttpRequest: Uri= <Command>; Event=1 enverra le message http://192.168.1.1/message=%251 ("%" est codé comme "% 25") après avoir pressé le bouton numéro 1.

| Bouton pressé | Format dans le formateur | Requête envoyée |
|---|---|---|
| Bouton 1 | http://10.27.1.6/message={1} | http://10.27.1.6/message=%251 ("%" est codé comme "%25") |
| Touche clavier 1 | http://10.27.1.6/message={1} | http://10.27.1.6/message=1 |
| Bouton 1 | http://10.27.1.6/mess?age={1} | http://10.27.1.6/mess?age=%251 |
| Touche clavier 1 | http://10.27.1.6/mess?age={1} | http://10.27.1.6/mess?age=1 |

**Observation**

- **Analyse de la valeur des paramètres séparés par des virgules**
  Un paramètre peut être délimité par des virgules en valeurs. Les valeurs peuvent être séparées par '' '(apostrophe arrière). Utilisez \ comme caractère d'échappement pour `.
  Exemples
  abc,def → abc et bcd
  `abc,def` → abc,bcd
  abc\`def → abc`def
  abc\def → abcdef
  abc\ \def → abc\def (barre oblique inversée deux fois de suite)
- La validité de la notification marquée % demeure.

## SendMulticastRequest

Le bloc **SendMulticastRequest** définit l'action permettant d'envoyer une commande vers plusieurs appareils. L'envoi de la commande peut-être déclenché par le bloc MulticastTrigger. La commande est un message envoyé par UDP vers l'adresse multicast (235.255.255.250:4433) et elle peut ainsi être reçue par de multiples appareils en même temps. Le message intègre l'ID de la commande (paramètre de la commande) et d'autres paramètres optionnels. Le message peut-être protégé par un mot de passe (Paramètres mot de passe). Il est recommandé d'envoyer ces commandes sur une fréquence de 1 commande par seconde maximum.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Command** – définit l'identifiant de la commande pour distinguer les types de commande. Le bloc MulticastTrigger block répond à l'action SendMulticastRequest seulement si l'identifiant de la commande est le même. N'importe quel texte contenant les caractères A–Z, a–z et 0–9 peut être utilisé pour l'identifiant.
- **Parameters** – définit un ou plusieurs paramètres de commande (séparés par des virgules) à inclure dans le message UDP. Conservez le format "nom_paramètre = valeur_paramètre".
  - Exemple :

    Params=“Address=192.168.1.1”, “Port=10000”

Les paramètres ainsi envoyés seront disponibles dans l'évènement HttpTrigger qui répond à cette commande ainsi que l'adresse et le port en paramètre de sortie et ils pourront être utilisés dans l'action rattachée à HttpTrigger, par exemple.

- **Password** – définit le mot de passe pour sécuriser la commande contre tout accès non-autorisé. Le paramètre est facultatif. Si aucun mot de passe n'est renseigné, la commande n'est pas sécurisée. Utilisez n'importe quel texte contenant les caractères A–Z, a–z et 0–9.

## Exemple

Envoi d'une commande d'ouverture de porte vers tous les appareils du réseau disposant du bloc Event.MulticastTrigger correctement paramétré si l'évènement défini à la ligne 2 survient :


**Observation**

- **Analyse de la valeur des paramètres séparés par des virgules**
  Un paramètre peut être délimité par des virgules en valeurs. Les valeurs peuvent être séparées par '' '(apostrophe arrière). Utilisez \ comme caractère d'échappement pour `.
  Exemples
  abc,def → abc et bcd
  `abc,def` → abc,bcd
  abc\`def → abc`def
  abc\def → abcdef
  abc\ \def → abc\def (barre oblique inversée deux fois de suite)
- La validité de la notification marquée % demeure.

## PlayUserSound

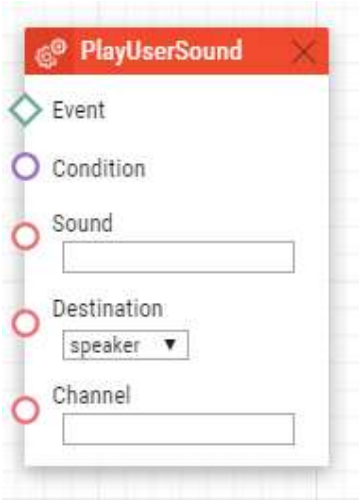
Le bloc **PlayUserSound** définit l'action permettant de déclencher un message audio.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Sound** – définit le message audio à jouer.
  - Valeurs valides pour les messages audio définit pas l'utilisateur :
    - **1–10** – numéro du message audio
  - Valeurs valides pour les messages audio prédéfinis (l'astérisque devant le numéro indique qu'il s'agit d'un message prédéfini) :
    - ***1** – Sonnerie moderne
    - ***2** – Gros gong
    - ***3** – Aboiement de chien
    - ***4** – Sirène
    - ***5** – Petit gong
- **Destination** – définit la destination du message à jouer.
  - Valeurs valides :
    - **speaker** – le son est joué par l'interphone
    - **call** – le son est joué pendant l'appel
    - **multicast** – le son est joué via une adresse multicast.
- **Channel** – définit le numéro de canal (0–3) à contrôler.
  - Ce paramètre est optionnel, la valeur par défaut est **speaker**.

### Exemple

Le message audio numéro 1 sera joué si l'évènement défini à la ligne 2 survient :




## StartLiftCall

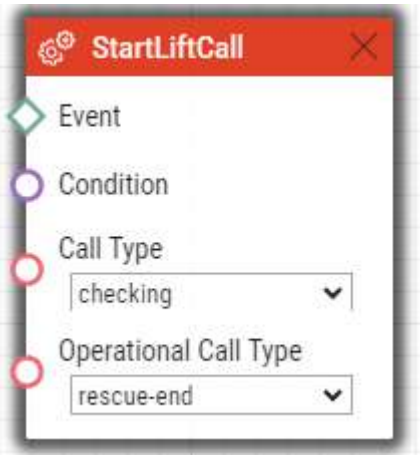
Le bloc **StartLiftCall** définit le lancement d'un appel à partir de l'appareil **2N LiftIP 2.0**. Si un appel de priorité supérieure est en cours, l'appel déclenché par cette action est placé en attente.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Call type** – définit le type d'appel qui sera lancé.
  - Valeurs valides :
    - **alarm** – Appel d'alarme (destination 1)
    - **alarm2** – Appel d'alarme (destination 2)
    - **operational** – Appel opérationnel
    - **checking** – Appel de contrôle
- **Operational Call Type** – définit le type d'appel de service. Ce paramètre n'est valable que si un appel de service est sélectionné dans le paramètre **Call Type**.
  - Valeurs valides :
    - **rescue-end** – appel de service à la fin du mode de dégagement.
    - **button-error** – appel de service en cas de panne du bouton ALARM1.
    - **button-fixed** – appel de service lorsque le bouton ALARM1 est réactivé après sa panne.
    - **audio-error** – appel de service lorsque le test audio échoue (erreur audio).
    - **audio-fixed** – appel de service lorsque l'erreur audio a été résolue lors du dernier test audio.

### Exemple

L'action lance un appel de contrôle lorsqu'un événement défini à la ligne 2 se produit et que la condition définie à la ligne 3 est valable :




## StartMulticastSend

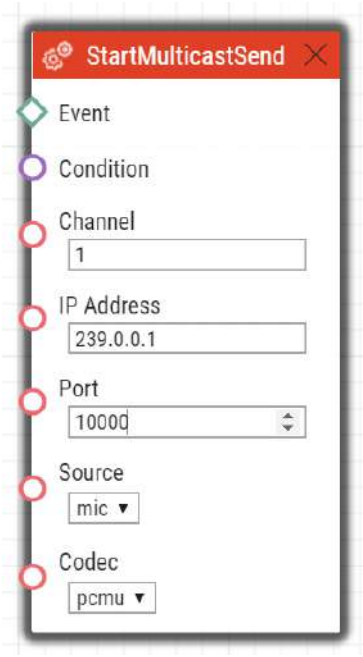
Le bloc **StartMulticastSend** définit l'action permettant de démarrer l'envoi du flux audio vers une adresse IP Multicast. Vous pouvez contrôler jusqu'à quatre canaux de transmission indépendants. Le protocole RTP / UDP est utilisé.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Channel** – définit le numéro de canal (0–3) à contrôler.
- **Address** – définit l'adresse IP multicast du flux audio.
- **Port** – définit le port UDP auquel le flux audio doit être envoyé.
- **Source** – définit la source audio.
  - Valeurs valides :
    - **mic** – la source audio est le microphone
    - **call** – la source audio est l'appel
  - Ce paramètre est optionnel, la valeur par défaut est **mic**.
- **Codec** – définit le codec audio à utiliser.
  - Valeurs valides :
    - **pcmu** – codec G.711 u-law
    - **pcma** – codec G.711 A-law
    - **g729** – codec G.729
    - **g722** – codec G.722
    - **l16** – codec L16, 16 kHz
  - Ce paramètre est optionnel, la valeur par défaut est **pcmu**.

### Exemple

Démarrage de l'envoi du flux audio via le canal numéro 1 vers l'adresse 239.0.0.1:10000 si l'évènement défini à la ligne 2 survient :




## StopMulticastSend

Le bloc **StopMulticastSend** définit l'action permettant de stopper l'envoi du flux audio vers une adresse IP Multicast.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Channel** – définit le numéro de canal (0–3) à contrôler.

### Exemple

Arrêt de l'envoi du flux audio via le canal 1 si l'évènement défini à la ligne 2 survient :


## StartMulticastRecv

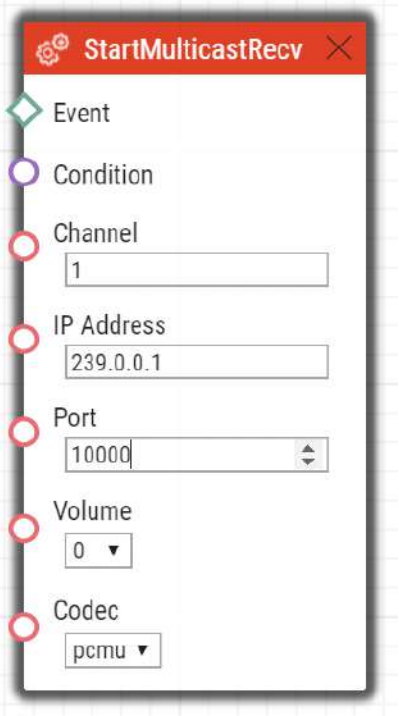
Le bloc **StartMulticastRecv** définit l'action permettant le démarrage de la réception et la lecteur d'un flux audio. Vous pouvez contrôler jusqu'à quatre canaux de transmission indépendants. Le protocole RTP / UDP est utilisé.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Channel** – définit le numéro de canal (0–3) à contrôler.
- **IP Address** – définit l'adresse IP multicast du flux audio.
- **Port** – définit le port UDP auquel le flux audio doit être reçu.
- **Volume** – définit le niveau de volume relatif du flux audio à lire (de −6 dB à +6 dB).
  - Valeurs valides :
    - **−6** – niveau minimum
    - **0** – niveau moyen (valeur par défaut)
    - **6** – niveau maximum
  - Ce paramètre est optionnel, la valeur par défaut est **0**.
- **Codec** – définit le codec audio à utiliser.
  - Valeurs valides :
    - **pcmu** – codec G.711 u-law
    - **pcma** – codec G.711 A-law
    - **g729** – codec G.729
    - **g722** – codec G.722
    - **l16** – codec L16, 16 kHz
  - Ce paramètre est optionnel, la valeur par défaut est **pcmu**.

### Exemple

Démarrage de la réception du flux audio via le canal 1 sur l'adresse IP 239.0.0.1:10000 si l'évènement défini à la ligne 2 survient :




## StopMulticastRecv

Le bloc **StopMulticastRecv** définit l'action permettant de stopper la réception d'un flux audio sur l'adresse IP multicast.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Channel** – définit le numéro de canal (0–3) à contrôler.

### Exemple

Arrêt de la réception du flux audio via le canal 1 si l'évènement défini à la ligne 2 survient :


## SetCameraInput

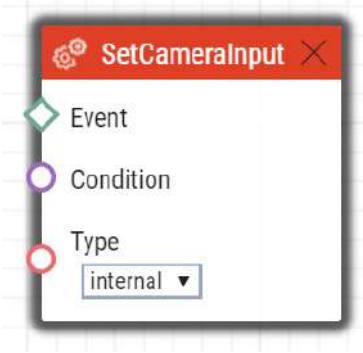
Le bloc **SetCameraInput** définit l'action permettant de changer la source du flux vidéo lors d'un appel : la caméra du portier, une caméra IP externe intégrée ou 2 caméra Analogue pour le **2N® IP Video Kit** si nécessaire. Cette action ne peut pas être utilisée pour la commutation de source vidéo pour les flux RTSP.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Type** – définit le type de signal vidéo. Un changement pendant un appel ne s'appliquera qu'à cet appel. D'autres récepteurs vidéo reçoivent la vidéo de la même source.
  - Valeurs valides :
    - **internal** – caméra interne au portier (ou caméra analogue externe directement connectée au portier)
    - **external** – caméra IP externe
  - Ce paramètre est optionnel, la valeur par défaut est **internal**.

### Exemple

Basculer la source du signal vidéo sur la première entrée de caméra interne si l'évènement défini à la ligne 2 survient :




## ControlRtpStream

Le bloc **ControlRtpStream** définit l'action permettant de contrôler le flux du stream RTP. Cette action contrôle uniquement les flux d'appels; les flux multicast ne sont pas affectés par cette action.

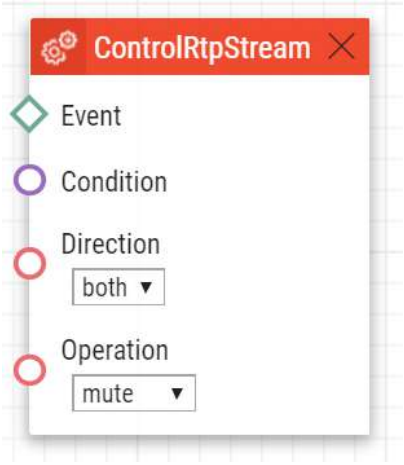
### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Direction** – définit la direction de lecture du flux RTP de l'appel.
  - Valeurs valides :
    - **in** – flux entrant sur l'interphone
    - **out** – flux sortant de l'interphone
    - **both** – flux entrant et flux sortant
  - Ce paramètre est optionnel, la valeur par défaut est **both**.

  **Operation** – définir l'opération de flux RTP.
  - Valeurs valides :
    - **mute** – mettre le flux en muet
    - **unmute** – réactiver le flux (le flux est lu).

### Exemple

Désactivation des flux d'appels dans les deux sens si l'évènement défini à la ligne 2 survient :




## LogAutomationEvent

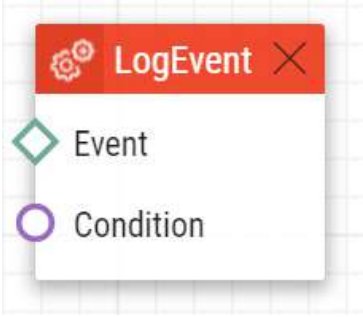
Le bloc **LogAutomationEvent** définit l'action permettant d'enregistrer un événement sur le serveur syslog. Ce bloc peut être utilisé pour vérifier les paramètres d'automatisation.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Log to Event Log** – détermine si l'événement doit être envoyé dans le journal des événements. La valeur par défaut est 0.
- **Name** – détermine le nom de l'événement. Ce paramètre est facultatif. La longueur maximale du nom est de 31 caractères dans le journal des événements.
- **Description** – détermine la description de l'événement. Ce paramètre est facultatif. La longueur maximale de la description est de 63 caractères.

### Exemple

Envoyer un message syslog avec l'événement capturé 2 (Event.CardEntered) si l'évènement défini à la ligne 2 survient :




## SendDTMF

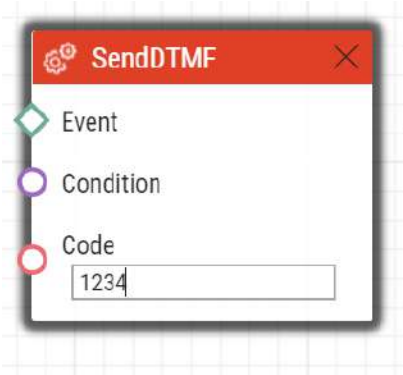
Le bloc **Action.SendDTMF** définit l'action permettant d'envoyer un code DTMF lors d'un appel actif.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Code** – définit les caractères DTMF envoyés.
  - Valeurs valides : 0–9, A–D, F

### Exemple

Envoi du code 1234 lors d'un appel :




## SendEmail

Le bloc **SendEmail** définit l'action permettant d'envoyer un e-mail.

Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Sender** – définit l'adresse de l'expéditeur pour les e-mails sortants.
- **Subject** – définit l'objet du message électronique à envoyer.
  - Des espaces réservés spéciaux peuvent être saisis pour la date, l'heure et l'identifiant de l'appareil. Ces espaces réservés sont remplacés par les valeurs actuelles avant l'envoi du message. Reportez-vous à la description du corps de l'email ci-dessous.
- **Body** – définit le corps du message à envoyer. Utilisez les caractères de formatage HTML. Vous pouvez entrer des espaces réservés spéciaux dans le texte pour la date, l'heure et l'identifiant de l'appareil à remplacer par les valeurs actuelles avant l'envoi du message. Voir la liste des symboles de substitution.

**Observation**

- Les symboles de substitution dépendant de cas particuliers fonctionnent si Action.SendEmail est déclenché par l'un des événements supportés. Voir Récapitulatif des symboles de substitution dans les événements.

- **Snapshots** – définit le nombre de photos prises par la caméra du portier à inclure dans l'e-mail [0, 5].
  - Ce paramètre est optionnel, la valeur par défaut est **1**.
- **TimeSpan** – définir la durée en secondes des photos joints à l'e-mail.
  - Ce paramètre est optionnel, la valeur par défaut est **1**.

**Conseil**

Il est conseillé de choisir des valeurs TimeSpan et Snapshots de manière à ce que le ratio TimeSpan / Snapshots soit un entier.
Exemple: Timespan = 8
Snapshots = 5
La dernière photo sera affichée suivi des précédentes dans un laps de temps de deux secondes.
Si la durée est inférieure au nombre de photo disponibles, certaines photos sont utilisées plusieurs fois.

- **Width** – définit la largeur de résolution de la photo de la caméra à inclure. Assurez-vous que la largeur de la photo est conforme à l'une des options de résolution prise en charge par l'interphone.
  - Ce paramètre est optionnel, la valeur par défaut est **640**.

- **Height** – définit la hauteur de résolution de la photo de la caméra à inclure. Assurez-vous que la largeur de la photo est conforme à l'une des options de résolution prise en charge par l'interphone.
  - Ce paramètre est optionnel, la valeur par défaut est **480**.
- **User** – définit l'utilisateur à qui cet e-mail sera envoyé.
- **E-mail** – définir l'adresse e-mail à laquelle l'e-mail sera envoyé. Entrez d'autres adresses e-mail si nécessaire, séparées par une virgule.
  - Valeurs valides :
    - **user@domain_name**
    - **user@ip_address**
    - **user@domain_name, user@ip_address**

**Conseil**

- Le paramètre **User** est préféré au paramètre **E-mail**.

### Exemple

Envoi d"un e-mail à l'adresse définie pour l'utilisateur Jana :


**Observation**

- Le paramètre **User** est limité à 10 utilisateurs.

## SetOnvifVirtualInput

Le bloc **SetOnvifVirtualInput** définit l'action permettant de changer l'état d'une entrée virtuelle via le protocole ONVIF. Le logiciel ONVIF Device Manager (version 2.2.250) peut être utilisé pour des tests.

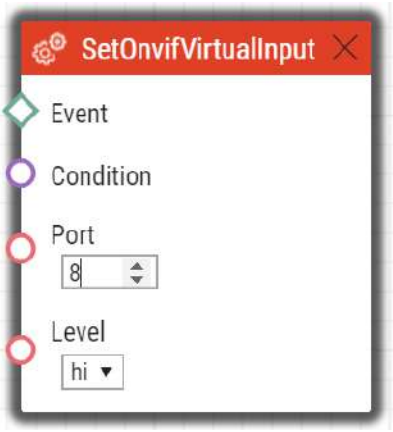
### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Port** – définit l'identifiant du port virtuel.
  - Valeurs valides :
    - **0–10** – ID du port

  **Level** – définit l'état de l'entrée.
  - Valeurs valides :
    - **hi** – définit la valeur logique sur Vrai
    - **lo** – définit la valeur logique sur Faux
  - Ce paramètre est optionnel, la valeur par défaut est hi (vrai).

### Exemple

Envoi d'informations indiquant que le port 8 a changé sa valeur pour 1 via le protocole ONVIF si l'évènement défini à la ligne 2 survient :




Les éléments suivants sont envoyés à ONVIF :

- InputToken: onvif_port_08
- LogicalState:true

## SendWiegandCode

Le bloc **SendWiegandCode** définit l'action permettant l'envoi d'un code saisi vers un autre appareil via l'interface Wiegand.

### Paramètres :

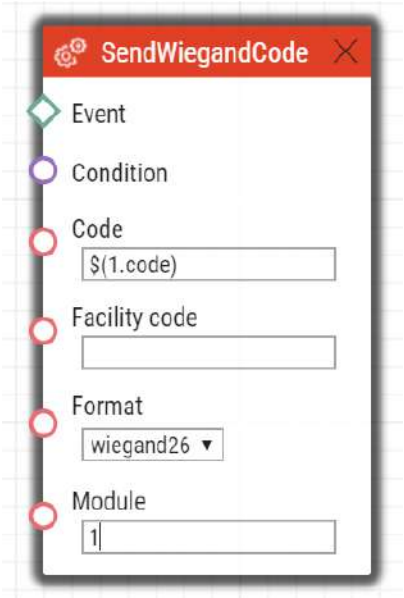
- **Event** – définit l'évènement permettant de déclencher l'action.

- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Code** – définit le code à envoyer sur le bus Wiegand. Si le code spécifié dépasse la capacité du message envoyé via Wiegand, les bits de poids fort sont réduits. Le code peut être saisi sous forme de nombre décimal ou hexadécimal. Il est également possible de saisir l'uuid de l'utilisateur depuis l'annuaire. Si l'uuid existe et que l'utilisateur dispose d'une carte virtuelle configurée, le bloc enverra cette carte virtuelle à l'interface Wiegand.
    - Valeurs valides :
        - Nombres décimaux
        - Nombres hexadécimaux
        - uuid
- **Facility code** – code de l'établissement. Ce paramètre s'applique uniquement pour "wiegand26".
    - Valeurs valides :
        - Nombres décimaux dans la tranche **0–255**
    - Ce paramètre est optionnel, s'il n'est pas définit, il n'est pas utilisé.
- **Format** – définit le format du message envoyé via Wiegand.
    - Valeurs valides :
        - **wiegand26** – 26 bits
        - **wiegand32** – 32 bits
        - **wiegand37** – 37 bits
    - Ce paramètre est optionnel, la valeur par défaut est **wiegand26**.
- **Module** – définit le module à travers lequel le code doit être envoyé.
    - Valeurs valides :
        - Le nom du module se configure dans l'interface à la section Hardware / Extendeurs / Modules / Menu des modules Wiegand.
    - Ce paramètre est obligatoire pour le modèle IP Verso mais ne s'applique pas aux autres modèles de portiers.

## Exemple

Envoi d'un code entré par l'évènement.CodeEntered via l'interface Wiegand :


Pour les modèles Verso et Access Unit la deuxième ligne doit ressembler à cela :



## UploadSnapshotToFtp

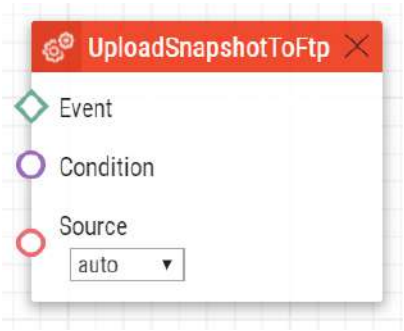
Le bloc **UploadSnapshotToFtp** définit l'action permettant d'envoyer une capture photo par la caméra vers un serveur FTP. Les paramètres du serveur FTP et des captures doivent être configurés dans la section Services / Streaming / Menu FTP.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Source** – définit la source vidéo de l'image à télécharger sur le serveur FTP.
  - Valeurs valides :
    - **auto** – la source vidéo est choisie en fonction des paramètres définis dans la section Hardware / Caméra / Paramètres communs / Source vidéo par défaut
    - **internal** – Caméra interne
    - **external** – Caméra externe
  - Ce paramètre est optionnel, la valeur par défaut est **auto**.

### Exemple

Chargement d'une photo de la caméra vers le serveur FTP si l'évènement défini à la ligne 2 survient :



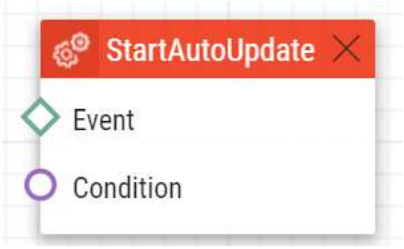

## StartAutoUpdate

Le bloc **StartAutoUpdate** définit l'action permettant la mise à jour automatique du FW et de la configuration. Les paramètres de provisionnement automatique sont configurés dans la section Système / Provisioning.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.

Envoyez un e-mail à l'adresse e-mail définie sur user2@domain_name si l'évènement 1 se produit :




## OpenDoor

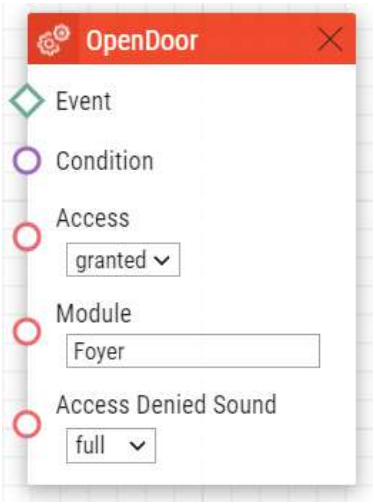
Le bloc **OpenDoor** définit l'action permettant le signalement sonore et visuel d'un accès Valide/Invalide depuis le lecteur de carte du **2N® IP Verso**.

### Paramètres :

- **Event** – définit l'évènement permettant de déclencher l'action.
- **Condition** – définit la condition à respecter pour exécuter l'action. Ce paramètre est optionnel.
- **Access** – signale la validité de l'accès
  - granted – valide
  - denied – invalide
- **Module** – définit le nom du module lecteur de carte
- **Sound** – definit le son de signalement
  - Valeurs valides :
    - **full** – son de signalement pour un accès valide/invalide
    - **none** – aucun son
- **Signaling Only** – l'appareil n'active pas l'interrupteur de la porte, l'appareil signale seulement l'ouverture de la porte

### Exemple

Signalement d'un accès valide au lecteur de carte du hall avec lumière et son :


OpenDoor
Event
Condition
Access
granted
Module
Foyer
Access Denied Sound
full

**Note**

**Other setting examples**

- Access granted / Access Denied Sound – None = Lumière verte, son paramétré comme pour l'interrupteur dans la section Hardware
- Access denied / Access Denied Sound – None = Lumière rouge, aucun son

**Observation**

- Les modules lecteur de carte connectés au bloc DoorOpen doivent être exclus du contrôle d'accès par les paramètres des modules Hardware / Extendeurs. Définissez Porte / Inutilisé pour le modèle donné.

# 5. Condition

**Automation** permet de définir les types de condition suivants :

- **True** – Condition toujours vrai
- **False** – Condition toujours faux
- **ProfileState** – Etat du profil horaire
- **CallState** – Statut actuel de l'appel
- **AccountState** – Statut d'enregistrement actuel du compte SIP
- **InputState** – Etat de l'entrée logique
- **FlipFlopRS** – Type de bascule RS
- **FlipFlopD** – Type de bascule D
- **LogicalAnd** – logique ET de la condition
- **LogicalOr** – logique OU de la condition
- **LogicalNot** – Négation de la condition
- **OnvifVirtualOutputState** – Etat du port ONVIF de la VMS
- **OutputState** – Etat de la sortie
- **SwitchState** – niveau logique du commutateur
- **SecureState** – compare le niveau logique au niveau défini dans le paramètre Level

Voir ci-dessous pour plus de détails sur les conditions, leurs paramètres et leurs utilisations.

## True

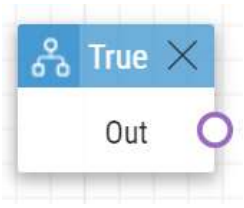
Le bloc **True** définit la condition à remplir à chaque fois.

### Paramètres sortants

- **Out**

### Exemple

La condition est toujours remplie :




## False

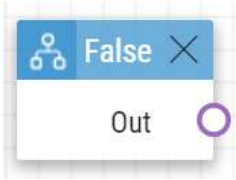
Le bloc **False** définit la condition à ne pas remplir, à chaque fois.

### Paramètres sortants

- **Out**

### Exemple

La condition n'est jamais remplie.




## ProfileState

Le bloc **ProfileState** définit la condition à remplir dans le cas d'un profil horaire actif/inactif.

### Paramètres entrants

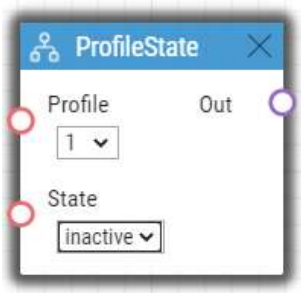
- **Profile** – définit le numéro du profil horaire (1–20 selon le modèle d'interphone).
- **State** – définit l'état souhaité du profil horaire. Ce paramètre est optionnel.
  - Valeurs valides :
  - **active** – profil actif (valeur par défaut)
  - **inactive** – profil inactif.

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie si le profil horaire 1 est inactif :



## CallState

Le bloc **CallState** définit la condition à remplir dans le cas d'un état défini de l'appel en cours.

### Paramètres entrants

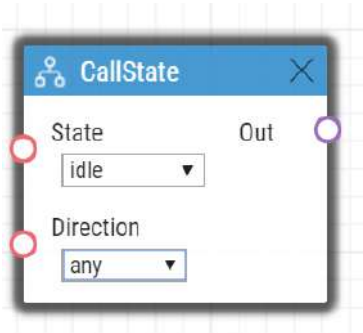
- **State** – définit le statut d'appel.
  - Valeurs valides :
    - **idle** – aucun appel n'est en cours
    - **connecting** – un appel sortant est en cours d'emission
    - **ringing** – sonnerie en cours
    - **connected** – appel en cours connecté
- **Direction** – définit la direction de l'appel.
  - Valeurs valides :
    - **incoming** – appels entrants
    - **outgoing** – appels sortants
    - **any** – les deux directions
  - Ce paramètre est optionnel, la valeur par défaut est **any**.

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie lorsqu'aucun appel n'est actif :




## AccountState

Le bloc **AccountState** définit la condition à remplir dans le cas de l'état d'enregistrement d'un compte SIP.

### Paramètres entrants

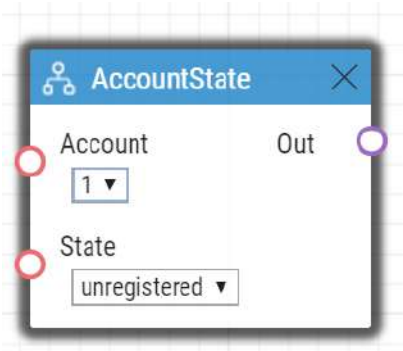
- **Account** – définit le compte SIP utilisé.
  - Valeurs valides :
    - **1** – Compte 1
    - **2** – Compte 2
  - Ce paramètre est optionnel, la valeur par défaut est **1**.
- **State** – définit l'état d'enregistrement.
  - Valeurs valides :
    - **registered** – le compte est enregistré
    - **unregistered** – le compte n'est pas enregistré
  - Ce paramètre est optionnel, la valeur par défaut est **registered**.

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie lorsque le compte SIP 1 n'est pas enregistré.




## InputState

Le bloc **InputState** définit la condition à remplir dans le cas où un état logique défini survient sur une entrée logique définie.

### Paramètres entrants

- **Input** – définit l'entrée logique.
  - Valeurs valides :
    - **tamper** – commutateur d'autoprotection
    - **input1** – entrée Logique 1
    - **input2** – entrée Logique 2

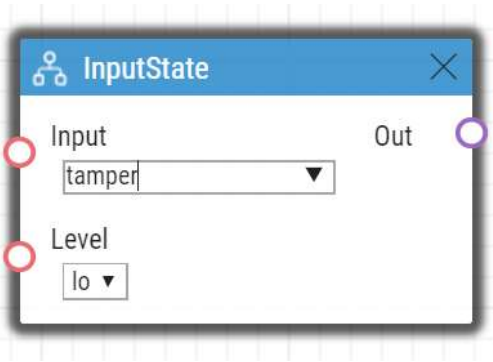
    - **cr_input1** – entrée logique 1 du lecteur de carte
    - **cr_input2** – entrée logique 2 du lecteur de carte
  - Il peut y avoir différentes listes de valeurs valides en fonction du modèle de l'interphone; référez-vous à la sous-section Entrée et Sortie logique disponibles.
- **Level** – définit l'état souhaité de l'entrée logique. Ce paramètre est optionnel.
  - Valeurs valides :
  - **lo** – logique 0
  - **hi** – logique 1 (valeur par défaut)

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie lorsque l'état du commutateur d'autoprotection est Lo (appareil non ouvert).




## FlipFlopRS

Le bloc **FlipFlopRS** est une cellule mémoire à un bit (paramètre de sortie), dont l'état passe à 1 ou 0 à la montée des événements définis. La sortie du bloc FlipFlopRS peut-être utilisée comme condition de contrôle d'action dans des scénarios complexe de l'interface **2N® Automation**. Il s'agit d'une simulation d'un circuit de bascule de type RS.

### Paramètres entrants

- **Set** – définit l'événement pour mettre la condition dans l'état «remplie» (1).
- **Reset** – définit l'événement pour mettre la condition dans l'état 'non remplie' (0).
- **ResetValue** – définit la valeur par défaut de la condition au redémarrage. Ce paramètre est optionnel.
  - Valeurs valides :

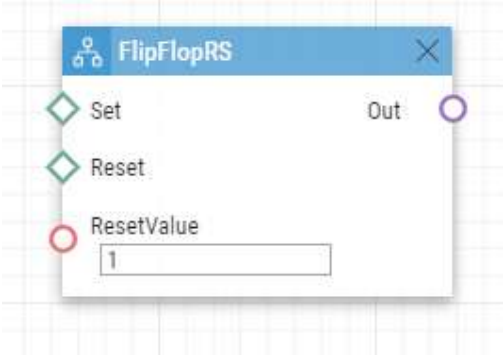
- **0** – la condition n'est pas remplie (valeur par défaut)
- **1** – la condition est remplie

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie à la montée de l'événement 1 et non remplie à la montée de l'événement 2 :




## FlipFlopD

Le bloc **FlipFlopD** est une cellule mémoire à un bit (paramètre de sortie), qui enregistre l'état d'une autre condition au moment de la montée de l'événement défini pour une utilisation ultérieure. La sortie du bloc FlipFlopD peut-être utilisée comme condition de contrôle d'action dans des scénarios complexe de l'interface **2N® Automation**. Il s'agit d'une simulation d'un circuit de bascule de type D.

### Paramètres entrants

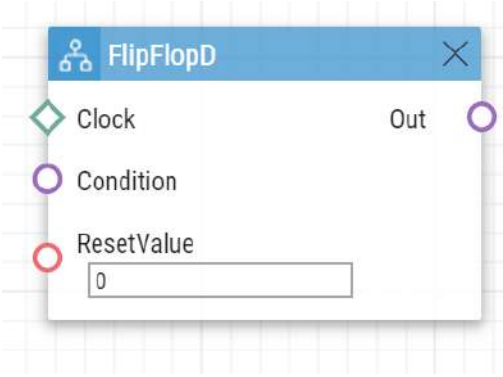
- **Clock** – définit l'événement au cours duquel l'état actuel de la condition doit être enregistré.
- **Condition** – définir la condition à enregistrer à la montée du bloc ClockEvent.
- **ResetValue** – définit la valeur par défaut de la condition au redémarrage. Ce paramètre est optionnel.
  - Valeurs valides :
  - **0** – la condition n'est pas remplie (valeur par défaut)
  - **1** – la condition est remplie

### Paramètres sortants

- **Out**

### Exemple

L'état de la condition est le même que l'état de la condition 2 à la montée de l'événement 1 :




## LogicalAnd

Le bloc **LogicalAnd** vous permet de créer des groupes de conditions. La condition est remplie si toutes les conditions du groupe défini sont remplies.

### Paramètres entrants

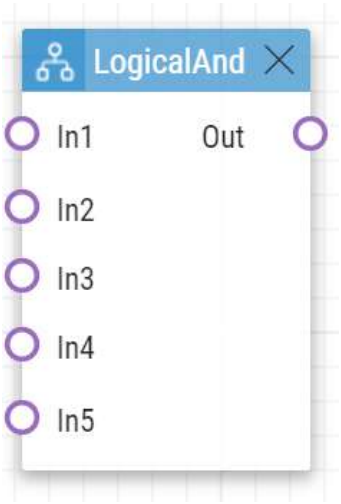
- **In1** – définit la condition à remplir.
- **In2** – définit la condition à remplir.
- **In3** – définit la condition à remplir.
- **In4** – définit la condition à remplir.
- **In5** – définit la condition à remplir.

**Paramètres sortants**

- **Out**

### Exemple

La condition est remplie si les conditions 1, 2 et 3 sont remplies au même moment.




## LogicalOr

Le bloc **LogicalOr** vous permet de créer des groupes de condition. La condition est remplie si au moins l'une des conditions définies dans le groupe est remplie.

### Paramètres entrants

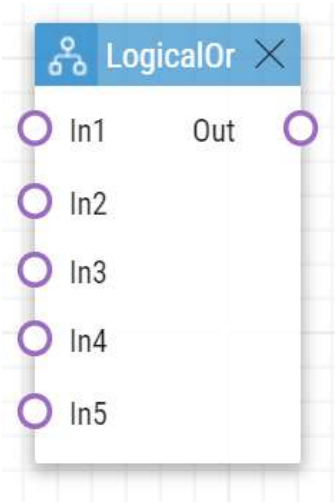
- **In1** – définit la condition à remplir.
- **In2** – définit la condition à remplir.
- **In3** – définit la condition à remplir.
- **In4** – définit la condition à remplir.
- **In5** – définit la condition à remplir.

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie si la conditions 1, 2 ou 3 est remplie :




## LogicalNot

Le bloc **LogicalNot** définit la condition à remplir si une autre condition n'est pas remplie.

### Paramètres entrants

- **In**

### Paramètres sortants

- **Out**

### Exemple

La condition est remplie si la condition 1 n'est pas remplie :


## OnvifVirtualOutputState

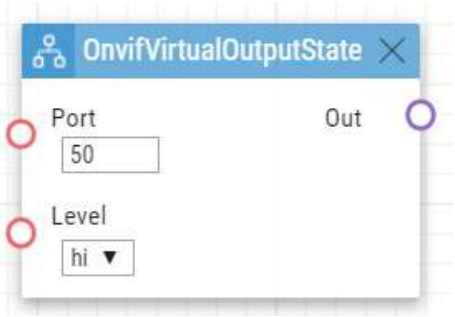
Le bloc **OnvifVirtualOutputState** définit la condition à remplir lorsque l'état du port virtuel défini est actif.

### Paramètres entrants

- **Port** – définit le port à observer. La valeur valide est compris dans la tranche 50–54.
- **Level** – définit la valeur logique du port à observer. Les valeurs valides sont high et low.

### Paramètres sortants

- **Out**




## OutputState

**OutputState** définit la condition qui correspond à l'état logique de la sortie.

### Paramètres entrants

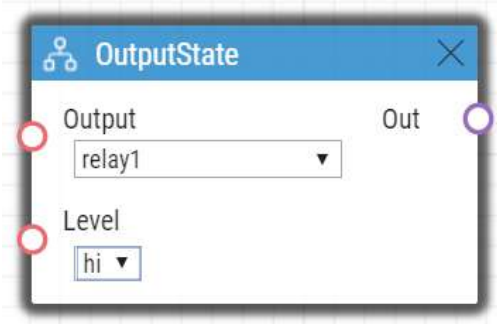
- **Output** – définit l'entrée logique.
  - Valeurs valides :
    - **relay1** – Relais 1 sur l'unité principale
    - **relay2** – Relais 2 sur l'unité principale
    - **output1** – Sortie 1 sur l'unité principale
    - **output2** – Sortie 2 sur l'unité principale
  - Les valeurs valides peuvent varier en fonction des modèles d'interphone; référez-vous à la sous-section Entrée et Sortie logique disponibles.
- **Level** – définit l'état souhaité de l'entrée logique. Ce paramètre est optionnel.
  - Valeurs valides :
    - **lo** – logique 0
    - **hi** – logique 1 (valeur par défaut)

### Paramètres sortants

- **Out**

**Exemple**

La condition est remplie si le relais 1 est actif :




## SwitchState

Le bloc SwitchState crée un niveau logique 0 ou 1 en fonction de la condition définie dans les paramètres ***Switch ID, Operation*** et ***Level***.

### Paramètres d'entrée

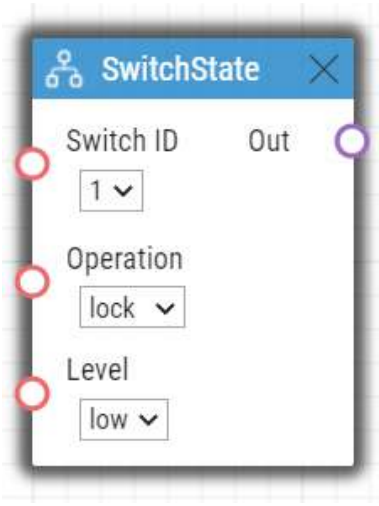
- ***Switch ID*** – définit quel commutateur est utilisé pour évaluer la condition.n
  - Valeurs valables :
    - X - où X est le numéro du commutateur correspondant (généralement 1... 4, différents modèles possèdent un nombre différent de commutateurs)
- ***Operation*** – définit le type d'opération du commutateur qui est utilisé pour évaluer la condition.
- Valeurs valables :
    - state – le commutateur peut être actif ou inactif (valeur par défaut)
    - lock – le commutateur peut être verrouillé (locked) ou déverrouillé (unlocked)
    - hold – le commutateur peut être maintenu (held) ou relâché (released)
- ***Level***
  - Valeurs valables :
    - lo – la condition est satisfaite si l'opération sélectionnée est en logique 0 (c'est-à-dire que le commutateur est inactif, le commutateur est déverrouillé, le commutateur est relâché).
    - hi – la condition est satisfaite si l'opération sélectionnée est en logique 1 (c'est-à-dire que le commutateur est actif, le commutateur est verrouillé, le commutateur est maintenu) (valeur par défaut)

### Paramètres de sortie

- ***Out*** – la valeur reflète si la condition est remplie.

### Exemple

Si le bloc est réglé sur Switch ID = Switch 1, Operation = lock, Level = low, la condition sera satisfaite (**Out** sera en logique 1) si le Switch 1 est déverrouillé.




## SecureState

Le bloc **SecureState** compare le niveau logique de l'état sécurisé au niveau défini dans le paramètre Level.

### Paramètres

- **Level** – définit la condition.
  - Valeurs valables :
    - lo – logique 0
    - hi – logique 1 (valeur par défaut)
- **Out** – la valeur indique si la condition est remplie.

# 6. Utilities

**L'Automatisation** permet de définir les types d'évènement suivants :

## Formatter

Le bloc **Formatter** vous permet d'utiliser les paramètres de sortie des événements dans les actions et de mettre en forme l'accès aux paramètres des actions. Le bloc Note peut être employé pour la documentation à l'intérieur de fonctions d'automatisation.

### Paramètres

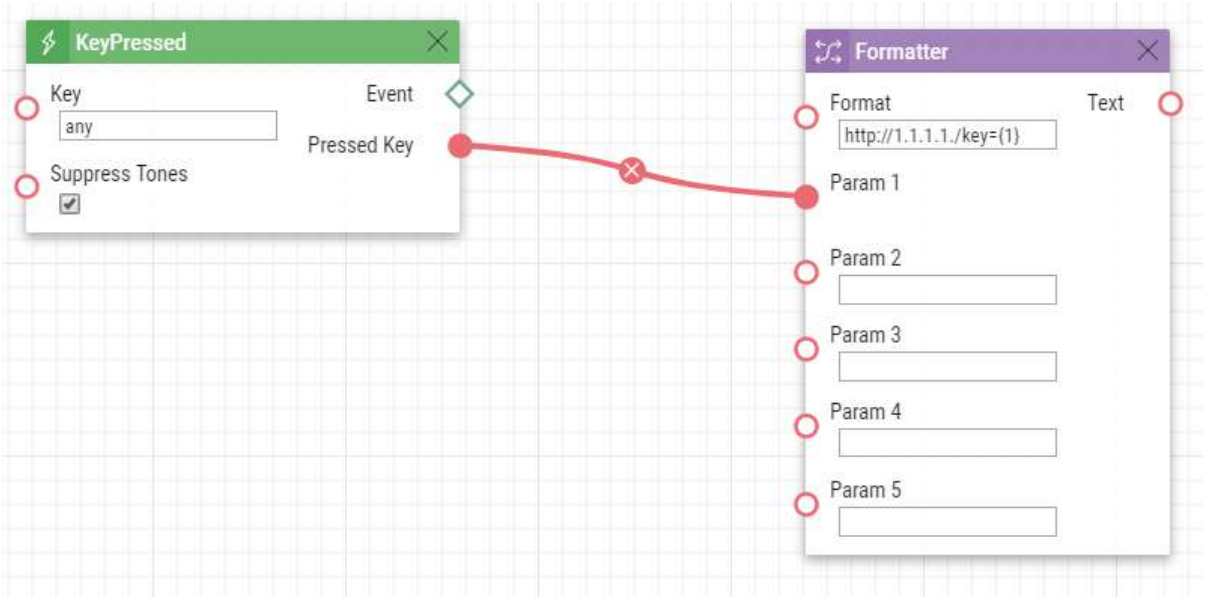
- **Format** – définit le texte à envoyer à la sortie de la variable Texte. Utilisez {} pour entrer les valeurs du Param1.
    - Exemple d'utilisation
        - *https:// ip_address/key={1.TimeStamp}* (enregistrer le texte avec la valeur du paramètre de sortie de Param 1).
- **Param 1** – définit un paramètre de sortie pour le paramètre Format. En général, il est connecté à la variable Events.
- **Param 2** – définit un paramètre de sortie pour le paramètre Format. En général, il est connecté à la variable Events.
- **Param 3** – définit un paramètre de sortie pour le paramètre Format. En général, il est connecté à la variable Events.
- **Param 4** – définit un paramètre de sortie pour le paramètre Format. En général, il est connecté à la variable Events.
- **Param 5** – définit un paramètre de sortie pour le paramètre Format. En général, il est connecté à la variable Events.

### Paramètres sortants

- **Text** – format final du texte avec les paramètres de sortie définis, le cas échéant. Dans Param 1.

## Exemple

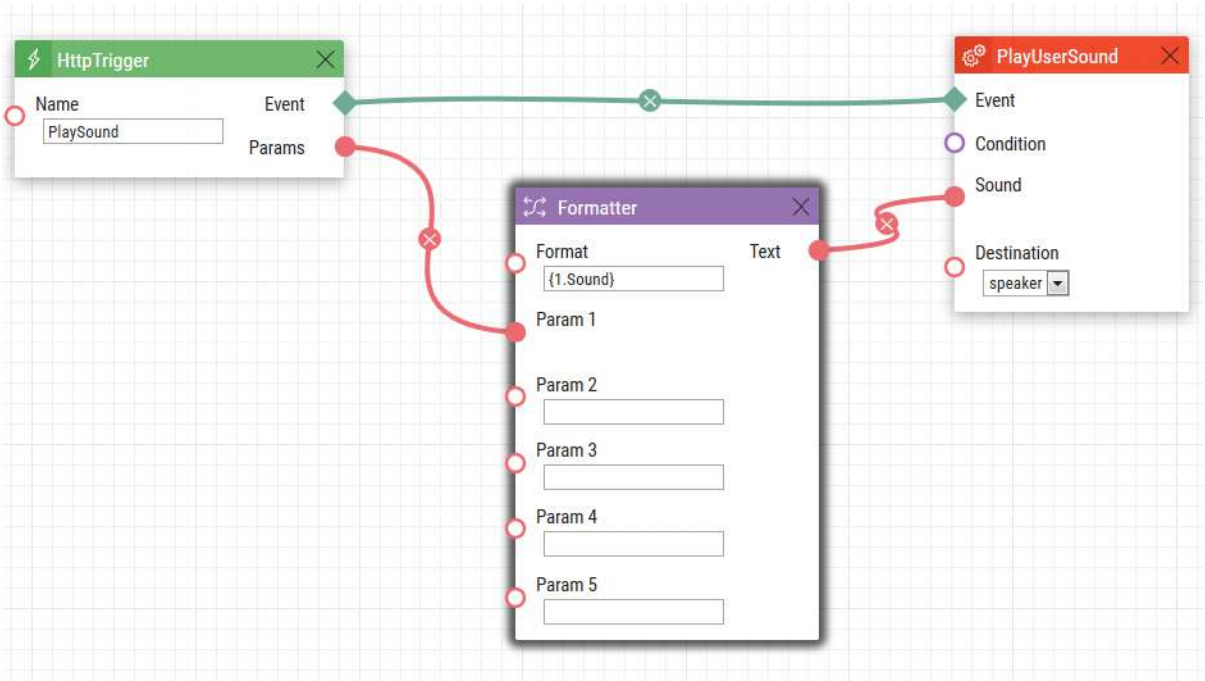
Préparez une commande HTTP incluant la valeur de la touche enfoncée KeyPressed.



## Exemple

Utilisez la valeur de paramètre reçue via HTTP comme variable d'action.

Lorsque la commande suivante est envoyée vers l'appareil : http://ip_address/api/automation/trigger?triggerId=id&param1=value1&param2=value2, les variables sont disponibles via la commande {param_input.param_name}.




La commande HTTP à envoyer est : *https://ip_address/api/automation/*

Format : {1.Sound}

Utilisez le son à jouer : 3

# 7. Entrées et sorties numériques disponibles

Dans cette section, les entrées et sorties numériques disponibles sur chaque appareil **2N** sont décrites.



## 2N IP Style

### Unité principale

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique 1 sur l'unité principale
- **input2** – entrée numérique 2 sur l'unité principale
- **input3** – entrée numérique 3 sur l'unité principale
- **tamper** – commutateur d'autoprotection

### Module I/0

Les entrées/sorties sont désignées de la sorte : **<module_name>.<input/output_name>**, e.g. module5.relay1.

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **relay1** – sortie relais 1
- **relay2** – sortie relais 2

**Entrées**

- **input1** – entrée numérique 1
- **input2** – entrée numérique 2

- **tamper** – commutateur d'autoprotection (s'il est installé)

### Module Wiegand

L'entrée est désignée de la manière suivante: **<module_name>.<input_name>**, e.g. module2.tamper

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **output1** – sortie LED OUT

**Entrées**

- **input1** – entrée LED IN
- **tamper** – commutateur d'autoprotection (s'il est installé)

## 2N IP One

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique 1 sur l'unité principale
- **tamper** – commutateur d'autoprotection

## 2N IP Vario

**Sorties**

- **relay1** – sortie relais sur l'unité principale
- **relay2** – sortie relais sur l'interrupteur additionnel (s'il est installé)
- **cr_relay1** – sortie relais 1 sur le lecteur de carte (s'il est installé)
- **cr_relay2** – sortie relais 2 sur le lecteur de carte (s'il est installé)

**Entrées**

- **cr_input1** – entrée numérique 1 sur le lecteur de carte (s'il est installé)
- **cr_input2** – entrée numérique 2 sur le lecteur de carte(s'il est installé)

## 2N IP Force/Safety

**Sorties**

- **relay1** – sortie relais sur l'unité principale

- **output1** – sortie numérique active sur l'unité principale (pour la version de carte 555v3 et supérieure, la sortie numérique active est connectée à la sortie relais 1 dans les cartes 555v2)
- **relay2** – sortie relais sur l'interrupteur additionnel (s'il est installé)
- **output2** – sortie numérique active sur l'interrupteur additionnel (s'il est installé)
- **cr_relay1** – sortie relais sur le lecteur de carte (s'il est installé)
- **cr_output1** – sortie numérique active sur le lecteur de carte (s'il est installé)

**Entrées**

- **tamper** – commutateur d'autoprotection (s'il est installé)
- **cr_input1** – entrée numérique 1 sur le lecteur de carte (s'il est installé)
- **cr_input2** – entrée numérique 2 sur le lecteur de carte (s'il est installé)
- **input2** – entrée numérique 2 sur l'interrupteur additionnel (s'il est installé)

## 2N IP Audio/Video Kit

**Sorties**

- **relay1** – sortie relais
- **output1** – sortie numérique 1
- **output2** – sortie numérique 2
- **led1** – LED 1 de contrôle de la sortie
- **led2** – LED 2 de contrôle de la sortie
- **led3** – LED 3 de contrôle de la sortie

**Entrées**

- **input1** – entrée numérique 1
- **input2** – entrée numérique 2

## 2N IP Verso/IP Verso 2.0/LTE Verso

### Unité principale

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique sur l'unité principale

### Module I/0 (Entrée/Sortie)

Les entrées/sorties sont désignées de la sorte : **<module_name>.<input/output_name>**, e.g. module5.relay1.

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **relay1** – sortie relais 1
- **relay2** – sortie relais 2

**Entrées**

- **input1** – entrée numérique 1
- **input2** – entrée numérique 1
- **tamper** – commutateur d'autoprotection (s'il est installé)

### Module Wiegand

L'entrée est désignée de la manière suivante: **<module_name>.<input_name>**, e.g. module2.tamper

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **output1** – sortie LED OUT

**Entrées**

- **input1** – entrée LED IN
- **tamper** – commutateur d'autoprotection (s'il est installé)

## 2N IP Solo

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique sur l'unité principale
- **tamper** – commutateur d'autoprotection

## 2N IP Base

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique sur l'unité principale
- **tamper** – commutateur d'autoprotection

## 2N SIP Audio Converter

**Sorties**

- **relay1** – sortie relais

**Entrées**

- Non-disponilbe.
- Il est possible d'utiliser l'évènement.KeyPressed: Key=%1 pour les événements générés sur l'entrée logique IN.

## 2N Access Unit

### Basic Unit

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique 1 sur l'unité principale
- **input2** – entrée numérique 2 sur l'unité principale
- **input3** – entrée numérique numéro 3 sur l'unité principale
- **tamper** – commutateur d'autoprotection

### Module I/0

Les entrées/sorties sont désignées de la sorte : **<module_name>.<input/output_name>**, e.g. module5.relay1.

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **relay1** – sortie relais 1
- **relay2** – sortie relais 2

**Entrées**

- **input1** – entrée numérique 1
- **input2** – entrée numérique 2
- **tamper** – commutateur d'autoprotection (s'il est installé)

### Wiegand Module

Les entrées sont désignées de la sorte: **<module_name>.<input_name>**, e.g. module2.tamper

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **output1** – sortie LED OUT

**Entrées**

- **input1** – entrée LED IN
- **tamper** – commutateur d'autoprotection (s'il est installé)

## 2N Access Unit 2.0

### Basic Unit

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique 1 sur l'unité principale
- **input2** – entrée numérique 2 sur l'unité principale
- **tamper** – commutateur d'autoprotection

### Module I/0

Les entrées/sorties sont désignées de la sorte : **<module_name>.<input/output_name>**, e.g. module5.relay1.

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **relay1** – sortie relais 1
- **relay2** – sortie relais 2

**Entrées**

- **input1** – entrée numérique 1
- **input2** – entrée numérique 2
- **tamper** – commutateur d'autoprotection (s'il est installé)

### Wiegand Module

Les entrées sont désignées de la sorte: **<module_name>.<input_name>**, e.g. module2.tamper

Le nom du module se configure dans la section de l'interface Hardware / Extendeurs

**Sorties**

- **output1** – sortie LED OUT

**Entrées**

- **input1** – entrée LED IN
- **tamper** – commutateur d'autoprotection (s'il est installé)

## 2N Access Unit M

### Basic Unit

**Sorties**

- **output1** – sortie numérique
- **relay1** – sortie relais

**Entrées**

- **input1** – entrée numérique 1 sur l'unité principale
- **input2** – entrée numérique 2 sur l'unité principale

# 8. Exemples d'utilisation

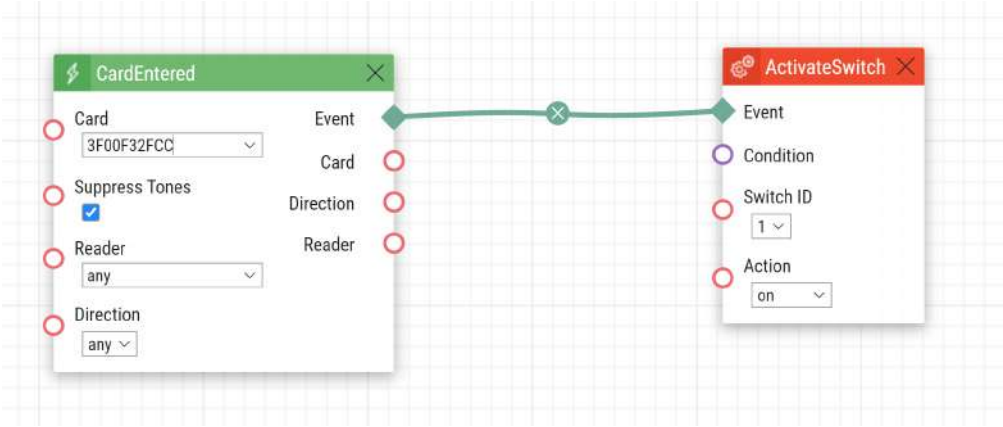
**Appel vers le PC sécurité en cas d'ouverture de porte non autorisée**

**Description**

Appelez le numéro de téléphone sélectionné chaque fois que le commutateur d'autoprotection est déconnecté (appareil ouvert).

**Diagramme de bloc :**

L'activation de l'entrée du Commutateur d'autoprotection (1: Evènement.InputChanged) déclenchera un appel vers le numéro défini 1111 (2: Action.BeginCall).




**Tip**

Fichier de configuration à télécharger

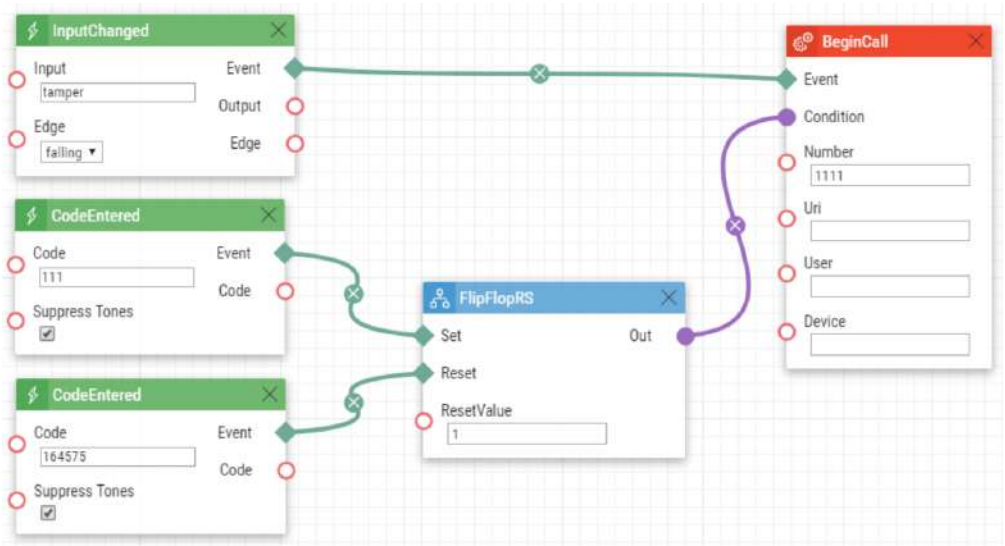
### Appel vers le PC sécurité en cas d'ouverture de porte non autorisée avec l'option blocage code de service

#### Description

Appelez le numéro de téléphone sélectionné chaque fois que le commutateur d'autoprotection est déconnecté (appareil ouvert). Activez le blocage et réactivez l'alarme de code numérique saisie à partir du clavier de l'interphone.

#### Diagramme de bloc :

L'activation de l'entrée du Commutateur d'autoprotection (1: Evènement.InputChanged) déclenchera un appel vers le numéro défini 1111 (5: Action.BeginCall) si la condition est remplie. La condition (4: Condition.FlipFlopRS) est validée par le redémarrage de l'interphone ou la saisie du code sélectionné (2: Condition.CodeEntered) sur le clavier numérique de l'interphone. Si un autre code est saisi (3: Condition.CodeEntered), la condition ne sera pas remplie.




**Tip**

Fichier de configuration à télécharger

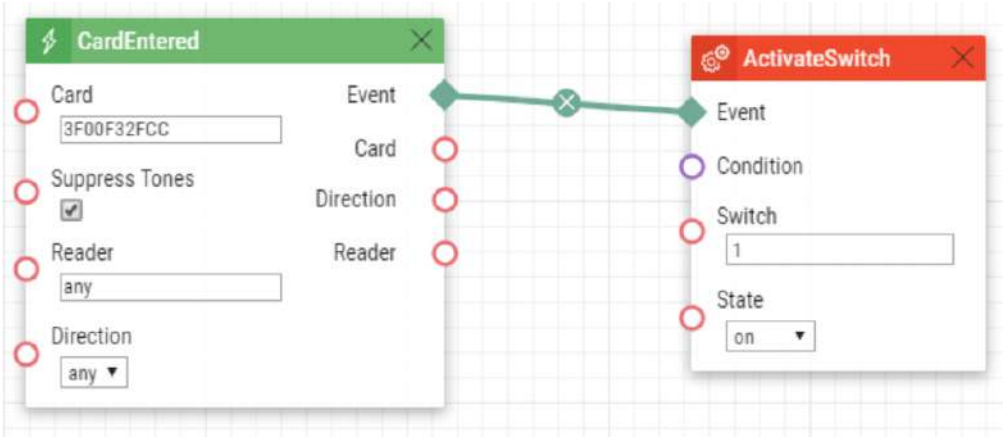
**Ouverture de porte avec une Carte RFID**

**Description**

Activez l'interrupteur de contact de porte en glissant la bonne carte RFID sur le lecteur.

**Diagramme de bloc :**

Entrer la carte RFID avec l'ID prédéfini suivant, (1: Evènement.CardEntered) activera l'interrupteur 1 (2: Action.ActivateSwitch).




**Tip**

Fichier de configuration à télécharger

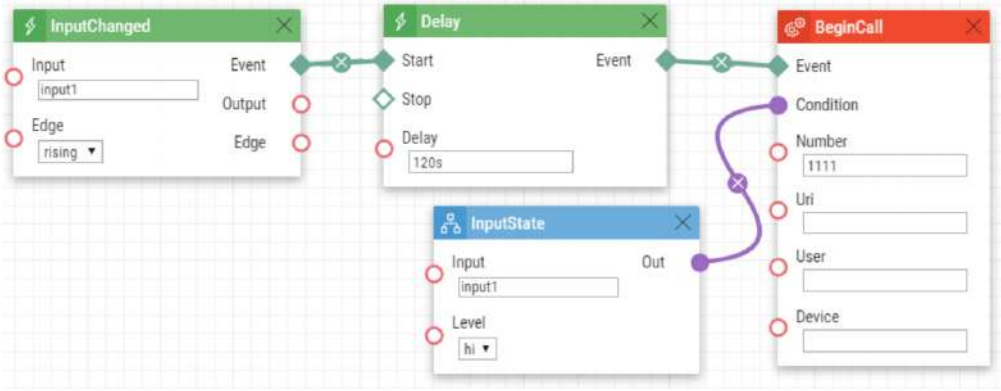
**Alarme (appel vers le PC Sécurité) causée par une ouverture de porte de plus de 2 minutes**

**Description**

Appelez le PC sécurité si la porte reste ouverte plus de 2 minutes. On suppose que le contact de signalisation d'ouverture de porte est connecté à l'entrée numérique Input1.

**Diagramme de bloc :**

Chaque fois que la porte s'ouvre, le changement d'état de l'entrée Input1 (1: Evènement.InputChanged) déclenche l'appel du numéro défini (4: Action.BeginCall) après un délai de 120 secondes (2: Event.Delay). L'appel ne s'effectue que si la porte reste ouverte plus de 2 minutes (3: Condition.InputState).




**Tip**

Fichier de configuration à télécharger

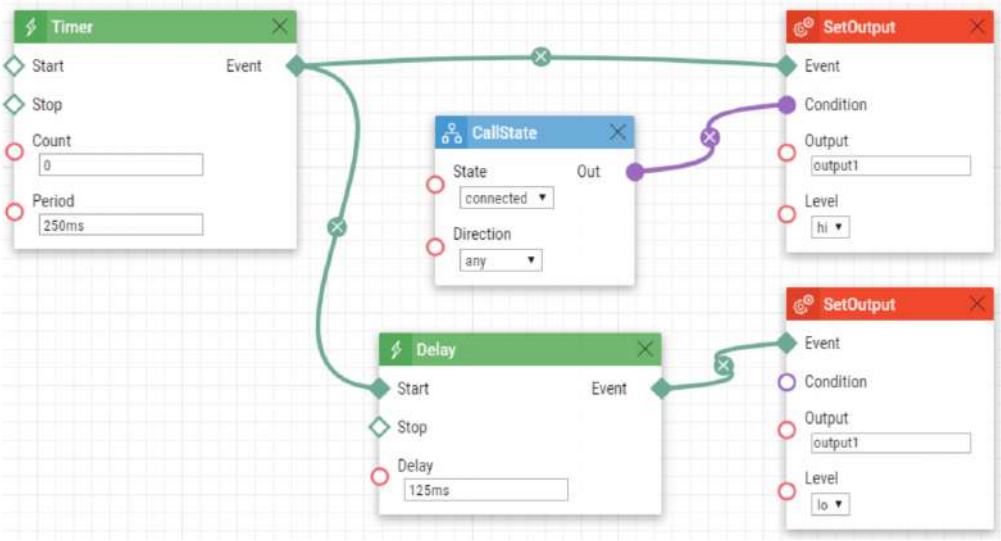
**LED clignotante pendant l'appel / ouverture de la gâche électrique de la porte**

**Description**

Activez le clignotement de la LED lors d'un appel en cours.

**Diagramme de bloc :**

Activer le clignotement de la LED par une combinaison du minuteur périodique (1: Evènement.Timer) et delai (2: Evènement.Delay). Ces deux blocs définissent la période (250 ms) et le rapport cyclique du signal ou de la période de clignotement de la LED (125 ms). Ces deux événements sont liés à l'action de mise en marche (4: Action.SetOutput) et de coupure (5: Action.SetOutput). L'action d'allumage de la LED est conditionnée par l'état d'appel actif (3: Condition.CallState).




**Tip**

Fichier de configuration à télécharger