Blocaïne

(La solution open source pour l'automatisme)

Table des matières

- 1 - Généralité

Blocaïne est une plateforme logicielle conçue pour l'automatisation des procédés industriels. Elle comprend deux composants principaux :

- Un **Éditeur de blocs**, qui permet de créer des programmes en assemblant des blocs fonctionnels.
- Un **Exécuteur ,** qui exécute les programmes créés avec l'**Éditeur de blocs,** pilotant ainsi l'équipement industriel via des entrées/sorties déportées.

L'**Éditeur de blocs** est installé sur un « PC de Développement » et communique par Ethernet avec l'**Exécuteur**, déployé sur un autre ordinateur appelé « Machine cible ».

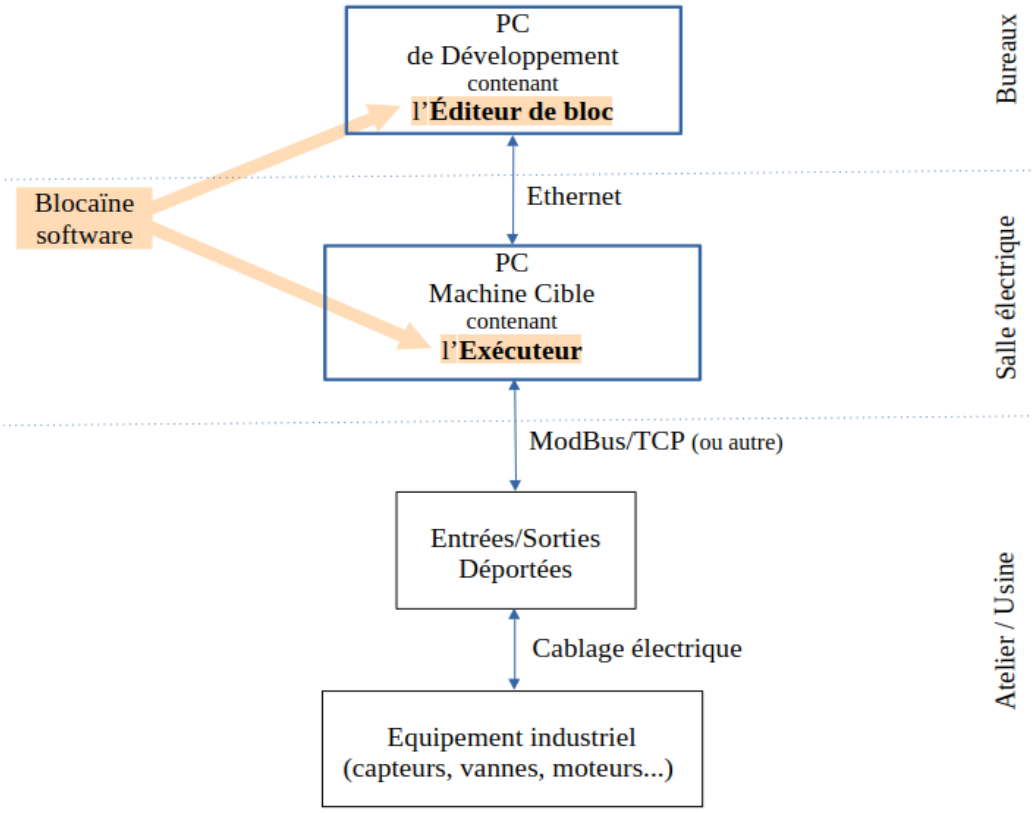


Figure 1 : Architecture générale

La connexion Ethernet permet de transférer les programmes créer avec l'**Éditeur de blocs** vers l'**Exécuteur**, elle autorise également la visualisation en temps réel au sein de l'**Éditeur de blocs** des variables des programmes en cours d'exécution par l'**Exécuteur**.

- 1.1 - Caractéristiques

- « Chargement à Chaud » (**HotSwap**) : Une nouvelle version d'un programme peut être chargée alors que l'ancienne version est en cours d'exécution.
- Programmation graphique sous forme de blocs fonctionnels chaînés
- Chaque variable possède nativement un bit de validité, qui se propage de bloc en bloc.
- Typage dynamique des variables.
- L'ordre d'exécution des blocs est géré automatiquement
- La mise en page des blocs est réalisée automatiquement
- Blocaïne est un logiciel libre et « open source »

- 1.2 - Avancement du projet

Le projet Blocaïne en est au stade de prototype Fonctionnel.

Il est téléchargeable pour évaluation sur « github.com », la procédure d'installation est décrite dans le document « evaluation.pdf »

Si vous souhaitez participer au projet, vous pouvez me contracter par Email : blocaine@barachet.com

- 1.3 - Précautions d'usages, responsabilités

L'utilisation de Blocaïne se fait sous la responsabilité de l'utilisateur.

- 2 - Principes

- 2.1 - Tout est bloc

Chaque bloc est constitué d'entré(s) de sortie(s) et du code permettant d'évaluer les sorties en fonction des entrées.

- Pour les blocs de la bibliothèque de base (appelés blocs **System**) c'est une fonction écrite en Python qui évalue les sorties en fonction des entrées
- Pour les blocs créés par l'utilisateur (blocs **user**) c'est un chaînage de blocs **system** ou **user**, qui permet d'évaluer les sorties en fonction des entrées

Un programme exécutable n'est autre qu'un bloc **user**, Un sous-programme se présente aussi sous la forme d'un bloc **user**, les entrées/sorties des blocs sont eux même des blocs (<inputs>, <output>), le bloc <input> possède une sortie définie par un bloc <output>, tandis que le bloc <output> possède une sortie définie par un bloc <input>. Bref tous est bloc.

- 2.2 - Chargement à Chaud (HotSwap)

l'**Exécuteur** dispose pour chaque programme (bloc **user**) de deux versions **Shift A** et **Shift B**. La nouvelle version d'un bloc est toujours téléchargée dans le **Shift** qui n'est pas en cours d'exécution, lorsqu'un **HotSwap** est initié, les valeurs courantes des variables mémorisées du **Shift** en cours d'exécution sont transférés à l'autre **Shift,** qui prend alors la relève.

- 2.3 - Méthode d'exécution

Pour chaque bloc **user** en cours d'exécution (Running), l'**Exécuteur** évalue périodiquement tous les blocs <output> associés à une tache périodique. Chaque entrée d'un bloc <output> est évalué par l'**Exécuteur** en exécutant d'abord le bloc précédent, et ainsi de suite.

Cette approche élimine la nécessité de définir explicitement l'ordre d'exécution des blocs.

- 2.4 - Typage des variables

Puisque Blocaïne s'appuie sur le langage Python, les variables (d'entrée et de sortie) utilisées par les blocs sont typées dynamiquement, leur type peut être quelconque y compris des structures complexes, dés lors qu'il s'agit d'une combinaison de types reconnue par Python (chaîne de caractères, entier, flotant, liste, dictionnaire, etc.).

- 2.5 - Bit de validité

Un bit de validité associé à chaque sortie de bloc est évalué automatiquement à chaque exécution en fonction de la validité des entrées et de la possibilité ou non d'évaluer de la sortie.

- 2.6 - Valeurs par défaut

Chaque entrée de bloc, qu'il s'agisse d'un bloc **system** ou **user**, possède une valeur par défaut modifiable à l'instanciation.

- 2.7 - Entrées/Sorties déportées

Les « remote I/O » sont gérées par des blocs **system** dédiés.

Actuellement, seul le protocole Modbus/TCP est pris en charge via ces blocs :

- modbus_conn : connection TCP avec le module d'entrées sorties déportées
- modbus_read : lecture dans un esclave ModBus
- modbus_write : écriture dans un esclave ModBus

- 3 - L’Éditeur de blocs

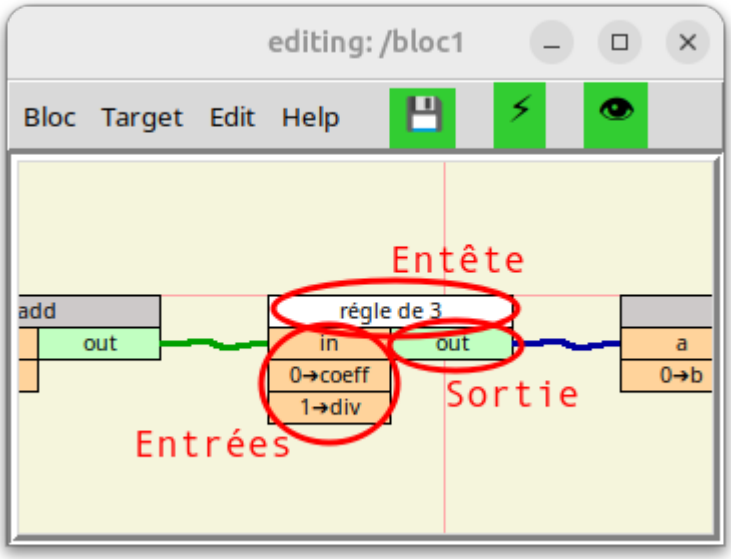
- 3.1 - Généralités

L’**Éditeur de blocs** permet :

- La création et la modification de programmes sous forme de blocs **user** contenant des blocs fonctionnels chaînés entre eux.
- La compilation , le téléchargement et le lancement de l’exécution du bloc **user** en cours d’édition dans la « Machine Cible ».
- la visualisation dynamique et le forçage des entrées/sorties des blocs **user** et les sous-blocs en cours d’exécution.

Chaque bloc est constitué de :

- Une entête : contenant le nom du bloc
- D’entrée(s) : située(s) à gauche sous l’entête, contenant le nom de l’entrée
- De sortie(s) : située(s) à droite sous l’entête à, contenant le nom de la sortie



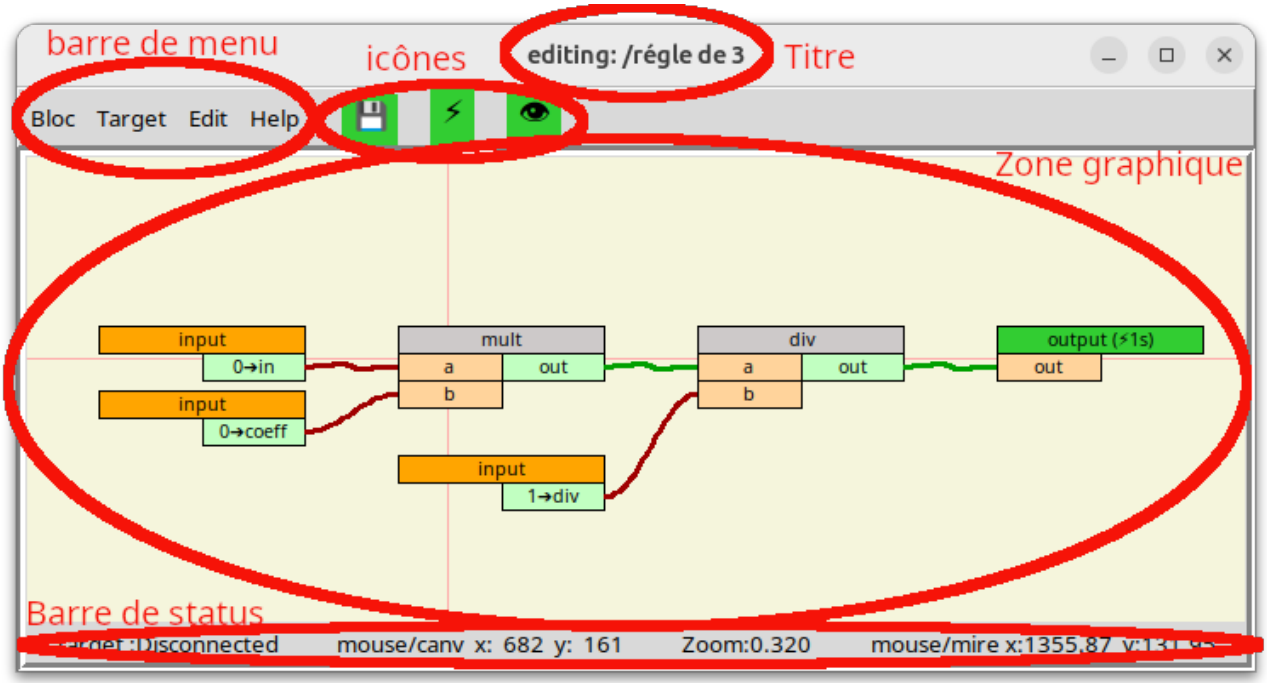
Les blocs peuvent êtres chaînés entre eux par des liens, notez qu’une sortie ne peut être liée qu’à une entrée et qu’une entrée ne peut être liée qu’à une seule sortie.

À chaque bloc correspond un fichier nommé : « *nom du bloc*.bloc ». Les fichiers des blocs **system** sont dans le répertoire /blocs/system, alors que les fichiers des blocs **user** sont dans /blocs/user.

- 3.2 - Avancement du projet Interface graphique

L'interface graphique de l'**Éditeur de blocs** « Blocaïne » est constituée d'une ou plusieurs fenêtres, chacune contient les éléments suivants :

1. Une barre de titre (de la fenêtre)
2. Une barre de menu
3. Trois icônes : 💾, ⚡, 👁️
4. Une zone graphique, ou vous déposez et reliez entre eux des blocs, afin de constitué un programme (blocs **user**).
5. Une barre de status :



Interface graphique de l'éditeur de bloc (Blocaïne)

- 3.2.1 - Barre de titre

En haut de chaque fenêtre se situe une barre de titre qui contient :

- le mode en cours : **Editing** ou **Monitoring**
- le nom du bloc en cours d'édition ou de débogage

Notez que si le bloc a été ouvert en temps que sous-bloc, sous-sous-bloc etc, le chemin ayant permis son ouverture apparaîtra devant le nom du bloc. Cela est utile en mode **Monitoring (**débogage) pour déterminer à quelle instance du bloc appartiennent les variables affichées dans cette fenêtre.

- 3.2.2 - Barre de menu

La barre de menu située en haut à gauche sous la barre de titre, contient les quatre menus suivants :

- **Bloc** : est équivalent au menu « Fichier » de la plupart des logiciels.
- **Target** : gère l'interface avec la « Machine Cible » (donc avec l'**Exécuteur).**
- **Edit** : gère la présentation.
- **Help** : affiche de la documentation.

Le menu « Bloc » contient le sous-menu suivant :

- **Open** : permet d'ouvrir un fichier contenant un bloc.
- **Save [F3]** : permet de sauvegarder le bloc en cours d'édition dans un fichier dont le nom correspond au nom du bloc courant.
- **Save as [F4]** : permet de sauvegarder le bloc en cours d'édition dans un fichier dont le nom sera saisie par l'utilisateur.
- **Update [F5]** : permet de mette à jour toutes les interface externe des blocs **user** utilisés comme sous-programme dans le bloc **user** en cour d'édition.
- **Change Properties** : cela permet de modifier les champs associés à l'entête du bloc **user** en cours d'édition comme « l'auteur du bloc », « les mots clef »...mais aussi son « nom ».
- **Open new windows** : ouvre une nouvelle fenêtre, ce qui permet ouvrir un autre bloc **user**.
- **Quit [Escape]** : quitte la fenêtre courante, sans sauvegarder le bloc **user** en cours d'édition.

Le menu « Target » contient le sous-menu suivant :

- **Disconnect from Target** ou **Connect to Target :** pour se connecter ou se déconnecter de la « Machine Cible » (donc à l'**Exécuteur**).
- **Check <*nom_du_bloc*>** : pour vérifier si le bloc en cours d'édition est valide et compilable.

- **Check & Transfert <*nom_du_bloc*>** : vérifie que le bloc en cours d'édition est valide le compile, puis transfert le résultat dans la « machine Cible » vers le **Shift** qui n'est pas « running ».
- **Check, Transfert & Execute <*nom_du_bloc*>** : vérifie que le bloc en cours d'édition est valide, le compile, transfert le résultat dans la « Machine Cible » vers un **Shift** qui n'est pas « running », puis l'**Exécuteur** lance son exécution: dans le cas ou le bloc est déjà en cours d'exécution sur l'autre **Shift** un chargement à chaud (HotSwap) sera initié, sinon un simple démarrage (Run) sera initié.
- **Monitoring (Start/Stop) [Space]** : permet de basculer du mode édition au mode débogage.

Le menu « Edit » contient le sous-menu suivant :

- **Auto layout [F7] :** cette commande effectue une mis en page automatique.
- **Status bar (show/hide) :** cette commande permet de montrer ou cacher la « barre de status ».

Le menu « Help » contient le sous-menu suivant :

- **Versions** : donne la version de l'**Éditeur de blocs** et de la structure des blocs.
- **Mouse usage** : donne des instructions sur l'utilisation de la souris :
  - [Left click] to select, to use menu
  - [Left held down] to scroll within window, to move bloc, to link I/O bloc
  - [Left double-click] to open **user** bloc
  - [Right button] to open the context menu
  - [wheel button] to monitor the target variables (on/off)
  - [wheel rotation] to zoom (in/out)
- **Key usage** : donne des instructions sur l'utilisation du clavier :
  - [F1] Open the documentation for the bloc whose header is located under the cursor
  - [F3] to save curant bloc
  - [F4] to save curant bloc as
  - [F5] to update curant bloc
  - [F7] to layout curant bloc
  - [Delete] Delete the bloc whose header is located under the cursor
  - [Space] to monitor the target variables (on/off)
  - [Escape] to close window

- **General Documentaion** : ouvre le présent document (au format HTML) dans le navigateur web par défaut.

- 3.2.3 - Icônes

Ces icônes permettent un accès rapide au fonctionnalités les plus fréquemment utilisées :

- "💾" : sauvegarde du bloc courant.
- "⚡" : vérifie la cohérence du bloc courant, le convertit (compilation) dans un format compréhensible par l'**Exécuter**, le transfert dans la Machine Cible puis demande à l'**Exécuteur** :
  1. Son démarrage (Run), si le bloc courant n'est pas déjà en cours d'exécution dans la machine Cible.
  2. Son chargement à chaud (HotSwap), si le bloc courant est déjà en cours d'exécution dans la machine Cible.
- "👁️" : bascule d'un mode à l'autre entre Editing et Monitoring.

- 3.2.4 - Zone Graphique

Cette zone vous permet de construire un programme (bloc **user**) en y déposant des blocs **system** ou **user** et en les reliant entre eux.

- 3.2.4.a - Navigation

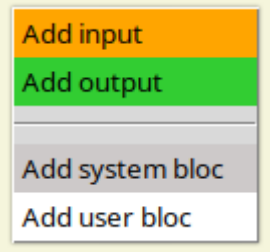
Pour naviguer dans cette zone, il faut utiliser une souris :

- **La rotation de la molette** : permet de zoomer/dézoomer.
- **Un déplacement de la souris bouton gauche enfoncée** : permet de déplacer la zone visible (scroller)
- **Clic droit** : permet de faire apparaître le menu contextuel.
- **Double clic gauche sur l'entête d'un bloc user** : ouvre le bloc **user** pointé dans une nouvelle fenêtre.
- **Double clic gauche sur une entrée ou une sortie d'un bloc user** : ouvre le bloc **user** de l'I/O pointée dans une nouvelle fenêtre et positionne l'entrée ou la sortie au centre de la zone graphique.
- **Clic gauche** : permet de sélectionner un champ dans un menu.
- **Déplacement clic gauche enfoncé** : permet de créer un lien (entre une entrée et une sortie) ou de déplacer un bloc.
- **Clic sur le bouton de la molette** : active/déactive la visualisation en temps réel des valeurs des variables situées dans la cible (Monitoring On/Off)

- 3.2.4.b - Menus contextuels

3.2.4.b.1 Menu « par défaut »

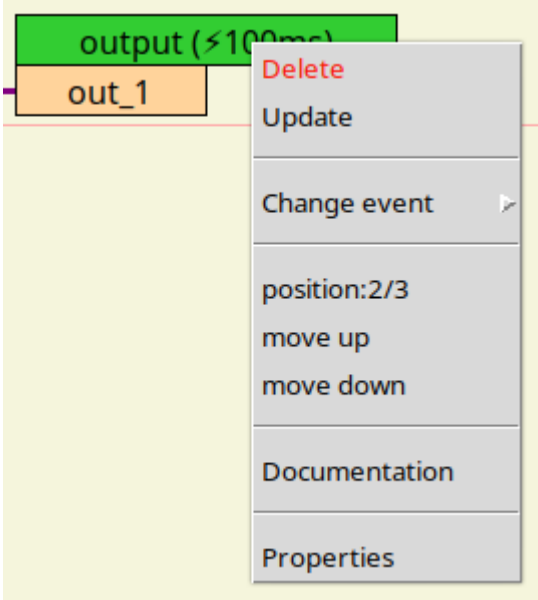
Un clic droit sur une zone vierge, fait apparaître le menu contextuel suivant :



ce menu permet d'ajouter un bloc.

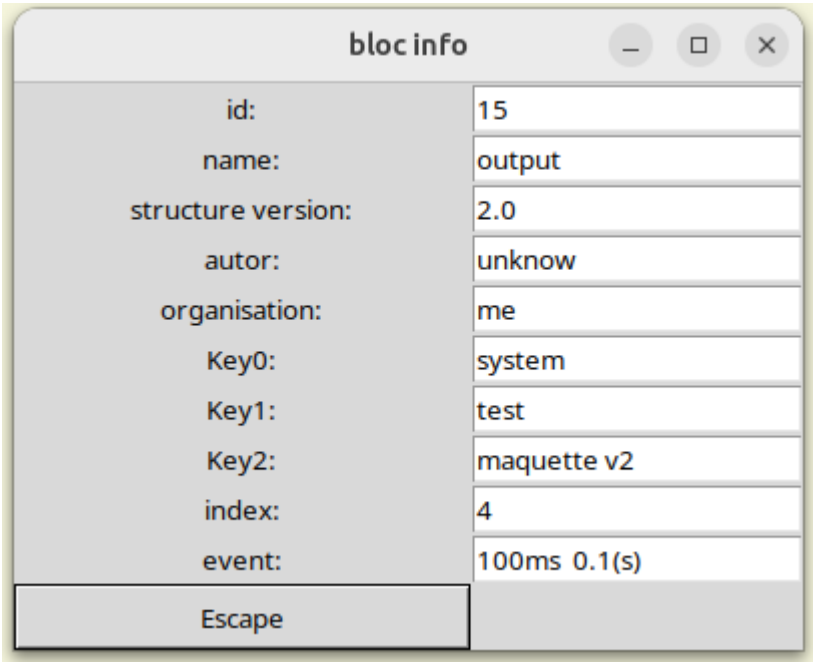
3.2.4.b.2 Menu « bloc »

Un clic droit sur l'entête d'un bloc, fait apparaître le menu contextuel suivant :



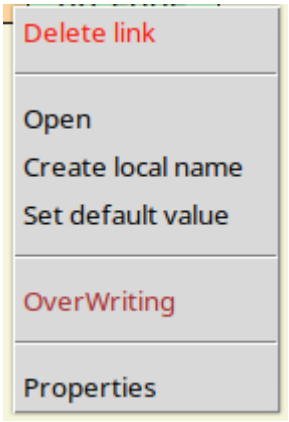
1. **Delete** : permet la suppression du bloc
2. **Update** : met à jour l'interface externe du bloc, à partir du fichier de ce bloc.
3. **Change event** : permet l'association le bloc à une tache périodique afin qui soi exécutable. Ce champ n'est disponible que pour les blocs <output>
4. **Position :2/3**, **move up**, **move down** : permet de modifier l'ordre des entrées ou des sorties dans l'interface externe du bloc **user** en cours d'édition. Ce champ n'est disponible que pour les blocs <input> ou <output>.
5. **Documentation** : ouvre le fichier d'aide du bloc dans le navigateur web par défaut. Ce champ n'est disponible que si un fichier d'aide nommé « bloc_ *type de bloc_nom du bloc*.html » est présent dans le répertoire « /Documentation ».
    1. *type de bloc* : peut être ; **user** ou **system**
    2. *nom du bloc* **:** correspondant au nom défini dans l'entête du bloc

6. Properties : afficher les propriétés du bloc comme ceci.

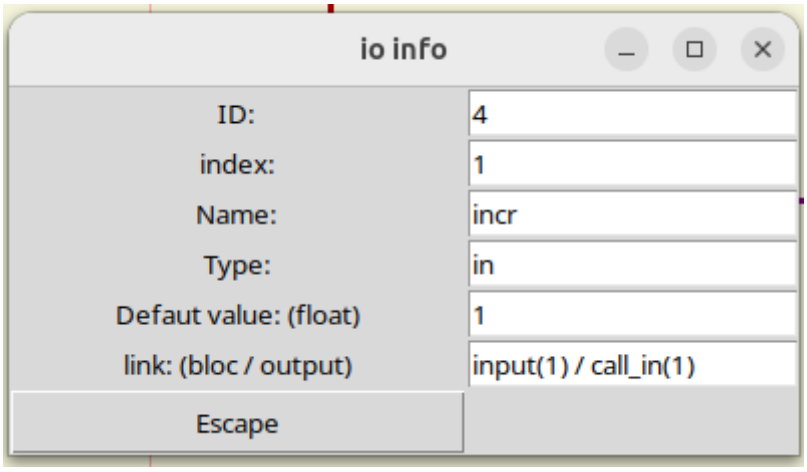


3.2.4.b.3 Menu « Entrée »

Un clic droit sur une entête d'un bloc, fait apparaître le menu contextuel suivant :

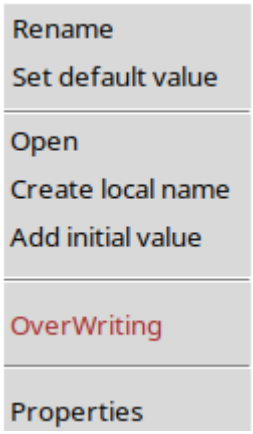


1. **Delete link** : permet la suppression du lien avec le bloc précédent. Ce champ n'est disponible que si il existe un lien entre l'entrée et la sortie d'un autre bloc.
2. **Open**: permet ouvrir une nouvelle fenêtre qui édite le bloc **user** de l'entrée, et positionne l'entrée au centre de la zone graphique.
3. **Create locale name** : permet de nommer localement l'entrée, ce nom est associé à l'interface externe de l'instance.
4. **Create locale comment** : permet d'associer un commentaire locale à l'entrée, ce commentaire est associé à l'interface externe de l'instance.
5. **Set default value** : permet de donner une valeur par défaut locale à l'entrée, cette valeur est associé à l'interface externe de l'instance.
6. **OverWritting** : permet de surcharger la valeur de l'entée.
7. **Properties** : afficher les propriétés de l'entrée comme ceci.

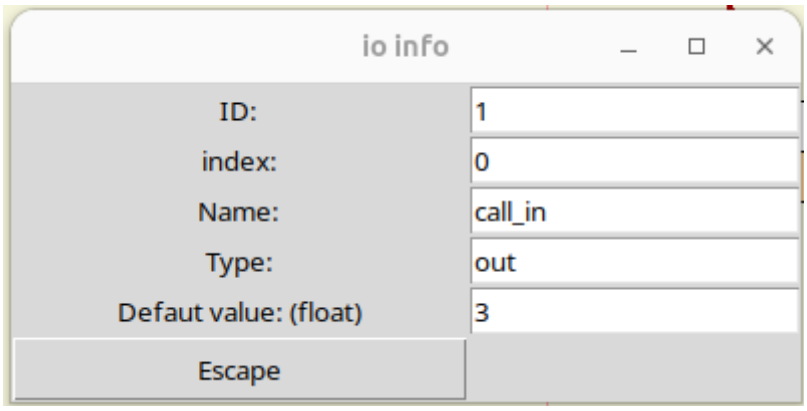


3.2.4.b.4 Menu « Sortie »

Un clic droit sur une sortie d'un bloc, fait apparaître le menu contextuel suivant :

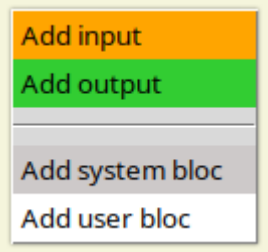


1. **Rename :** permet de renommer le nom de la sortie, se nouveau nom apparaîtra dans l'interface externe du bloc. Ce champ n'est disponible que pour les blocs <input>.
2. **Set defaut value :** permet de définir la valeur par défaut de l'entrée liée à l'instance du bloc. Ce champ n'est disponible que pour les blocs <input>.
3. **Open**: permet ouvrir une nouvelle fenêtre qui édite le bloc **user** de l'entrée, et positionne la sortie au centre de la zone graphique. Ce champ n'est disponible que pour les blocs **user**.
4. **Create locale name** : permet de nommer localement l'entrée, ce nom est associé à l'interface externe de l'instance.
5. **Create locale comment** : permet d'associer un commentaire locale à l'entrée, ce commentaire est associé à l'interface externe de l'instance.
6. **Add initial value** : permet de donner une valeur initiale à une sortie « mémorisée ». Ce champ n'est disponible que pour tous les blocs dont la sortie est « mémorisée » d'un cycle sur l'autre, ex : <memory>, <previous>, <edge>.
7. **OverWritting** : permet de surcharger la valeur de la sortie.
8. **Properties** : afficher les propriétés de la sortie comme ceci.



- 3.2.4.c - Insérer d'un bloc

Pour insérer un bloc, il faut faire un clic droit sur une zone vierge, afin de faire apparaître le menu contextuel suivant :

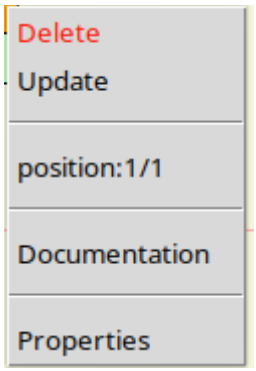


Il faut alors choisir le type de bloc que vous souhaitez :

- input : pour créer une nouvelle entrée dans le bloc **user** en cours d'édition.
- output : pour créer une nouvelle sortie dans le bloc **user** en cours d'édition.
- system : pour insérer un bloc faisant partie de la bibliothèque standard.
- user : pour insérer un bloc **user** précédemment créé.

- 3.2.4.d - Supprimer d'un bloc

Pour supprimer un bloc, vous pouvez faire un clic droit sur l'entête du bloc à supprimer, afin de faire apparaître le menu contextuel suivant :



puis sélectionner Detete.

Vous pouvez aussi positionner le curseur sur l'entête du bloc à supprimer, puis appuyer sur la touche « Suppr ».

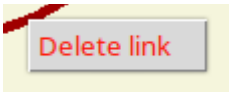
- 3.2.4.e - Créer un lien

Pour lier la sortie d'un bloc à l'entrée d'un autre bloc, enfoncez le bouton gauche de la sourie sur la sortie du premier bloc, puis glissez (bouton gauche toujours enfoncé) jusqu'à l'entrée du second bloc puis relâcher le bouton gauche. Les liens peuvent aussi être créés indifféremment d'entrée vers sortie ou de sortie vers entrée, mais une entrée ne peut avoir qu'une seule source (donc provenir d'une seule sortie).

- 3.2.4.f - Supprimer un lien

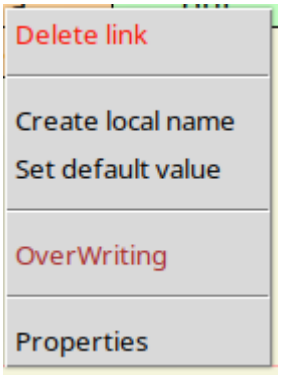
Il y deux possibilités pour supprimer un lien :

Soit faire un clic droit sur le lien, afin de faire apparaître le menu contextuel suivant :



puis sélectionner Delete link

Soit faire un clic droit sur l'entrée située à l'extrémité du lien, afin de faire apparaître le menu contextuel suivant :



puis sélectionner Delete link

- 3.2.5 - Barre de status

Elle indique :

- l'état de la connexion avec la Machine Cible.
- la position de la souris par rapport à la zone graphique (canvas).
- le coefficient de zoom courant.
- la position de la souris par rapport à la mire.

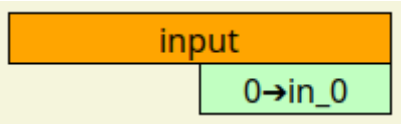
- 3.3 - Les entrées/sorties

L'interface d'un bloc **user** est constitué ; d'entrées et de sorties, celle-ci sont optionnelles mais indispensable pour l'interconnexion avec les autres blocs.

- Les entrées sont collectées à l'aide de blocs **system** <input>
- Les sorties sont délivrées à l'aide de blocs **system** <output>

- 3.3.1 - bloc <input>

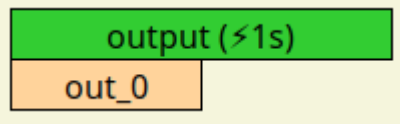
Voici l'interface graphique du bloc **system** <input>



Sa fonction est de définir une entrée d'un bloc **user**. Le bloc <input> n'a pas d'entrée, mais possède une sortie nommée « in_0 », c'est la variable (de type quelconque) qui rentrera dans du bloc **user,** sa valeur par défaut est 0 (int), elle peut être ajustée pour chaque instance d'un bloc <input>. Cette sortie peut être renommée, le nouveau nom apparaîtra dans l'interface externe du bloc **user** en tant qu'entrée.

- 3.3.2 - bloc <output>

Voici l'interface graphique du bloc **system** <output>



Sa fonction est de définir une sortie d'un bloc **user**. Le bloc <output> n'a pas de sortie, mais possède une entrée nommée « out_0 », c'est la variable (de type quelconque) qui sortira du bloc **user,** elle n'a pas de valeur par défaut, mais une valeur par défaut peut être ajouter pour chaque instance d'un bloc <output>. Cette entrée peut être renommée, le nouveau nom apparaîtra dans l'interface externe du bloc **user** en tant que sortie.

- 3.3.3 - Interface externe d'un bloc user

Voici un bloc **user** nommé « règle de 3 » qui contient trois blocs <input> et un bloc <output>.

- la sortie du première bloc <input> a été renommée en « in »
- la sortie du second bloc <input> a été renommée en « coeff »
- la sortie du troisième bloc <input> a été renommée en « div »
- l'entrée du bloc <output> a été renommée en « out »

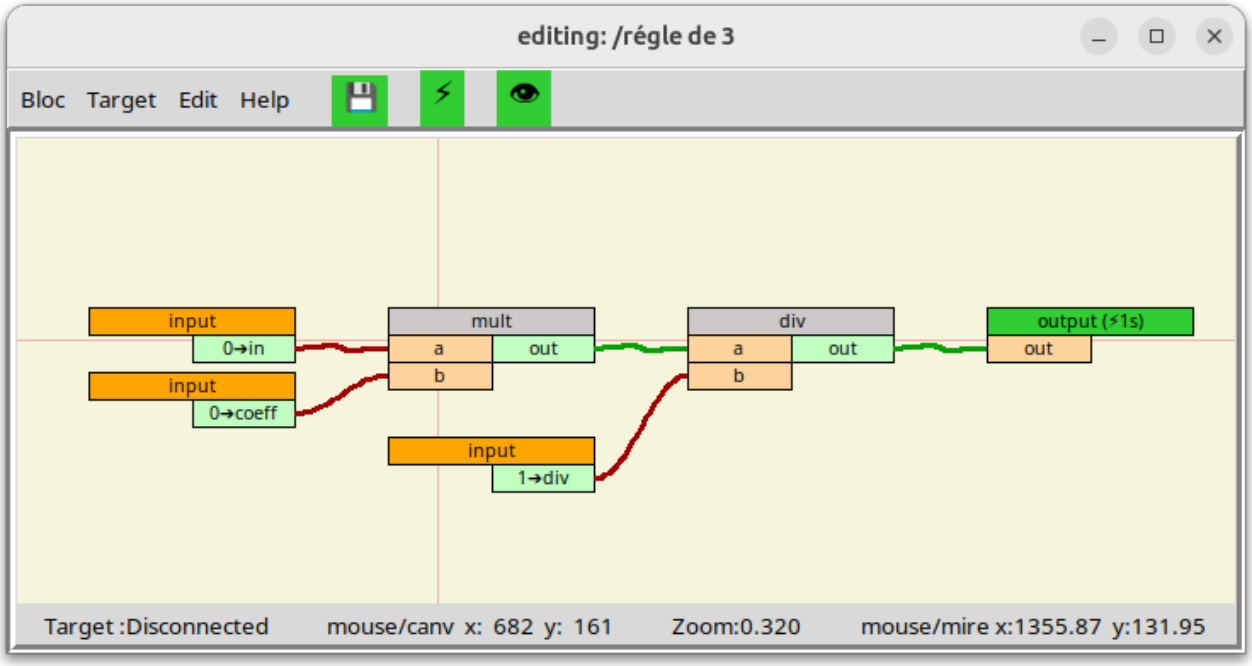


Figure : bloc **user** <règle de 3 >

Voici l'interface externe du bloc « règle de 3 », ici un bloc **user** nommé <bloc1> fait référence au bloc « régle de 3 », les 3 entrées et la sortie correspondent aux <input>/<output> définies dans le bloc « régle de 3 » sont bien présentes dans son interface externe.

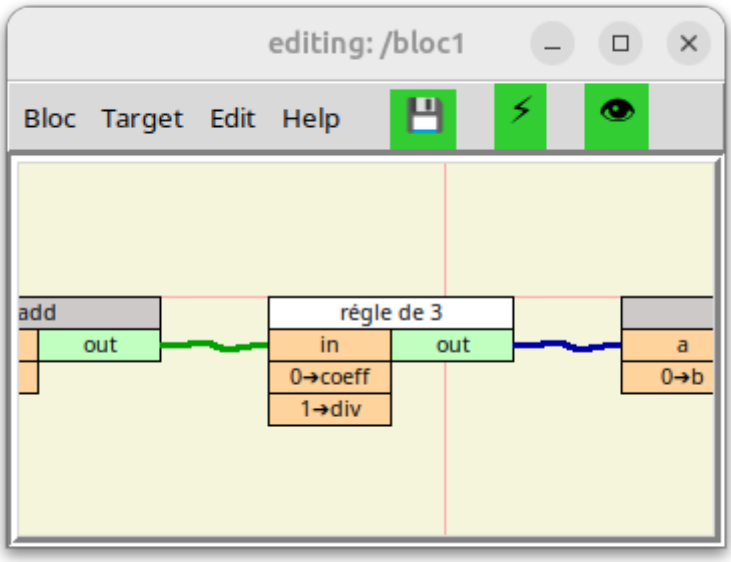
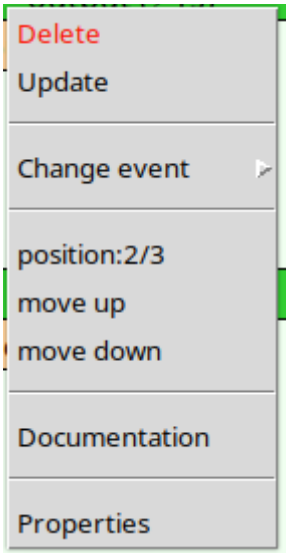


Figure ; interface externe du bloc **user** <règle de 3>

L'ordre d'apparition des entrées et des sorties dans l'interface externe d'un bloc **user** peut être modifier de la façon suivante :

1. Ouvrir le bloc **user** dont vous voulez modifier l'ordre des entrées et/ou des sorties de son interface externe, en double cliquant sur l'entête du bloc à modifier, si il est présent dans la zone graphique, ou en utilisant la barre de menu (bloc/open) ou (bloc/open new window).
2. Faites un clic droit sur l'entête du bloc <input> ou <output> selon que vous souhaitiez modifier l'ordre des entrées ou des sorties. Afin de faire apparaître le menu contextuel suivant :
3. **Position:2/3** indique la position dans l'interface externe, de l'entrée ou la sortie que vous avez sélectionnée , vous pouvez alors sélection **move up** ou **move down.** pour la décaler d'une position vers haut ou vers le bas.



- 3.4 - Mode : Editing / Monitoring

Le basculement du mode « édition » au mode « visualisation dynamique » et vice versa, s'effectue soit en cliquant sur l'icône « œil » (), soit en appuyant sur la touche [Espace], bien sûr le bloc édité doit être en cours d'exécution pour pouvoir basculé en mode « visu dynamique ».

- 3.4.1 - Mode : édition (Editing)

Ce mode est dédié à création et à la modification de blocs **user**.

Voici un exemple de bloc **user** qui calcule la moyenne de 2 nombres :

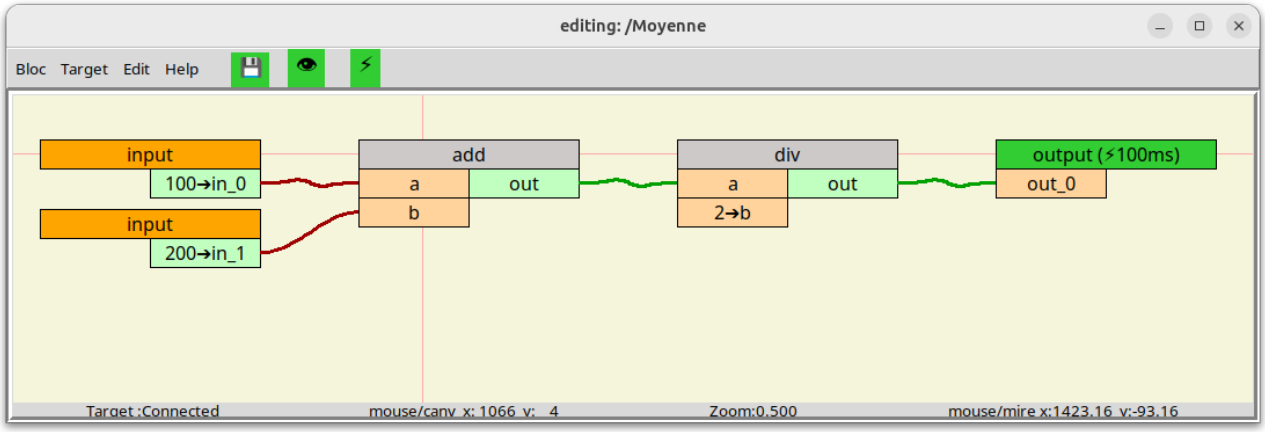


Figure 2 : bloc en mode édition

- 3.4.2 - Mode : visualisation dynamique (Monitoring)

Ce mode est dédié au débogage, il permet la visualisation en temps réel des valeurs des variables d'un bloc **user** ou d'un sous-bloc **user**

la validité de chaque variable est représentée par la couleur de fond :

- Vert = valide
- Rouge = invalide
- jaune = forcé et valide
- Noir = Forcé et invalide

Voici un exemple du bloc « moyenne » (en visualisation dynamique) :

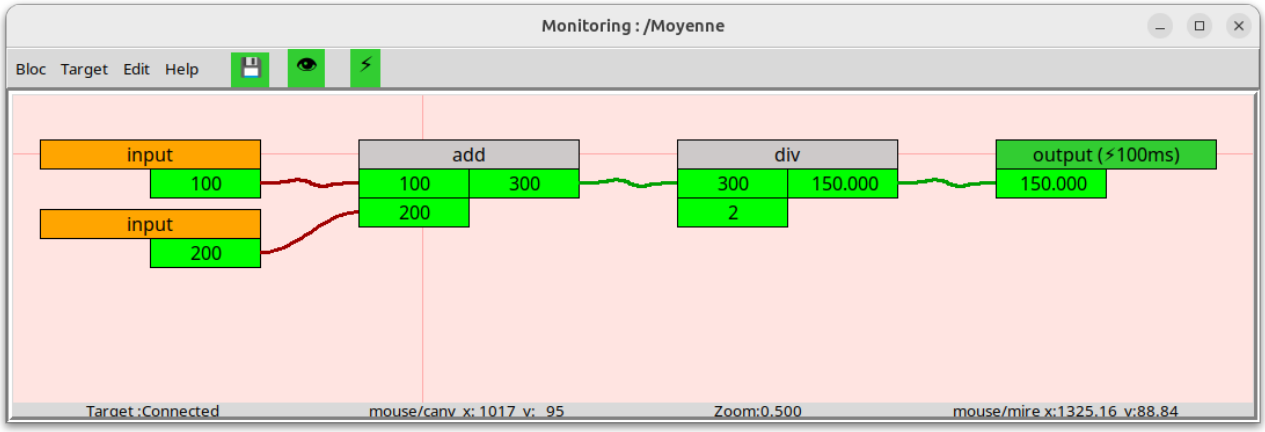


Figure 3 : bloc en mode visu dynamique

- 3.5 - Bit de validité

Les bits de validité peuvent être lus ou modifiés en utilisant les blocs **system** suivants :

- <valideRead> : permet de lire le bit de validité, cela permet par exemple de sélectionner une position de replie dans le cas de variables « invalide »
- <valideWrite> : permet l'écriture du bit de validité, autorisant l'utilisateur à définir sa propre équation de validité.

- 4 - L’Exécuteur

- 4.1 - Généralités

Rappel : L’**exécuteur** est installé sur un PC appelé « Machine cible », il exécute les programmes (blocs **user**) qui pilotent l’équipement industriel en utilisant les entrées/sorties déportées.

Certaines commandes liées aux blocs **user** déjà présent dans l’**exécuteur**, telles que : le démarrage, l’arrêt, l’initialisation, le swap, la suppression, peuvent être accomplies par l’**exécuteur** par le biais de son serveur web intégré vous chapitre #7.5.

Les opérations de modification d’un bloc **user** ou visualisation en temps réel les valeurs des variables des blocs **user** en cours d’exécution, ne sont pas disponible qu’à partir de l’**Éditeur de blocs**.

- 4.2 - Méthode d’exécution

Pour qu’un chaînage de blocs contenu dans un bloc **user** soient exécutable, il est impératif que ce chaînage se termine par un bloc <output> et que ce dernier soit associé à un événement (comme l’événement périodique à 100ms par exemple). l’**Exécuteur** évalue périodiquement tous les blocs <output> du bloc **user**, déclenchant ainsi l’évaluation du bloc précédent qui nécessite évaluation du bloc précédent et ainsi de suite. Cette approche élimine la nécessité de définir explicitement l’ordre d’exécution.

Un bloc **user** peut contenir plusieurs blocs <output>, chaque <output> pouvant êtres associés à un événement périodique différent. Il est donc possible d’inclure dans un même blocs **user** des chaînes de blocs qui seront exécutés avec des périodicités différentes. Si il existe des liens entre les chaînes de blocs de périodicité différente, sur chaque lien inter-chaîne il faudra insérer un bloc **system** <previous>, afin que la chaîne qui doit être exécuté lentement ne soit pas exécuté à la période de la chaîne la plus rapide.

- 4.3 - Chargement à Chaud (hotswap)

Pour que le chargement à la volée soit possible, l’**Exécuteur** dispose pour chaque bloc **user** (programme) de deux versions **Shift A** et **Shift B**. La nouvelle version d’un bloc **user** est toujours téléchargée dans le **Shift** qui n’est pas en cours d’exécution, lorsqu’un « hot swap » est initié, les valeurs courantes des variables du **Shift** en cours d’exécution sont transférés à l’autre **Shift**, qui prend alors la relève.

- 4.4 - Bit de validité

De manière générale tant qu’un bloc n’a pas était évalué ses sorties sont « invalides »

La validité associée à chaque sortie est évaluée automatiquement lorsque le bloc est exécuté. Si une des entrées nécessaires à l’évaluation de la sortie est « invalide » ou si la sortie du bloc ne peut être évalué à cause d’un confit de type ou autre, la sortie sera « invalide », sinon elle sera « valide ».

- 4.5 - Serveur Web

- 4.5.1 - Généralités

Un serveur HTTP est intégré à l'**Exécuteur,** il permet à l'utilisateur par le biais d'un simple navigateur web de visualiser l'état de la Machine Cible :

- Visualiser les taches disponibles et le nombre de <output> associé
- Visualiser la charge CPU générale et par tache
- Visualiser la liste des blocs disponible dans la Machine Cible et l'état des **Shift**s **(**A &t B) (Pending, Ready, Running)
- Visualiser l'état et la valeur des <output> de tous les blocs pour les **Shift**s (A & B)
- Visualisé les connexions Ethernet en cours gérées par l'**Exécuteur**

Il permet aussi de passer des commandes simples telle que :

- Arrêter l'exécution d'un bloc **user** (commande : Stop)
- Démarrer l'exécution d'un bloc **user** (commande : Run)
- Initialiser un bloc **user** (commande : Initialize)
- Supprimer un bloc **user** (commande : Delete)
- Intervertir les **Shift A** et **Shift B**
  - lorsqu'aucun **Shift** est « Running » (commande : ClodSwap)
  - lorsqu'un **Shift** est « Running » (commande : HotSwap)
- Lancer des commandes dédiés au débug
  - (commande : print list_compiled)
  - (commande : print list_threads)

- 4.5.2 - Exemple de page « Task list »

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu
Task list Bloc & Output list Connexions print list_compiled print list_threads

Task list								
Setting			Feedback					
name	id	period	cycle time	cycle time min	cycle time max	counter	number of output	CPU load
10s	0	10s	10.000011s	9.999804s	10.000738s	155	0	0.000%
3s	1	3s	3.000265s	2.999798s	3.000808s	518	0	0.000%
1s	2	1s	1.000031s	0.999822s	1.002054s	1555	2	0.004%
200ms	3	200ms	200.046ms	199.769ms	201.699ms	7775	0	0.001%
100ms	4	100ms	99.994ms	99.745ms	103.087ms	15547	0	0.003%
50ms	5	50ms	49.937ms	49.749ms	52.764ms	31083	0	0.010%
20ms	6	20ms	19.953ms	19.744ms	22.160ms	77657	0	0.020%
10ms	7	10ms	9.996ms	9.743ms	12.555ms	155273	0	0.043%
5ms	8	5ms	5.028ms	4.743ms	7.809ms	310276	0	0.062%
2ms	9	2ms	2.022ms	1.708ms	3.533ms	792271	0	0.379%

User tasks CPU load = 0.52%

Refresh

- 4.5.3 - Exemple de page « Bloc & Output list »

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu
Task list Bloc & Output list Connexions print list_compiled print list_threads

bloc list			
bloc name	status		order
	shift A	shift B	
call2loops	Pending	Running	Stop Initialize HotSwap

bloc output list									
bloc			output					task	
name	shift	building time (yyyy/mm/dd)	name	id	value	validity	status	name	id
call2loops	A	2025/07/30 - 18:19:10	out_1	5	...	...	Down	1s	2
call2loops	A	2025/07/30 - 18:19:10	out_1	10	...	...	Down	1s	2
call2loops	B	2025/07/30 - 18:25:40	out_1	5	1303	😊	Running	1s	2
call2loops	B	2025/07/30 - 18:25:40	out_1	10	6515	😊	Running	1s	2

Refresh

- 4.5.4 - Exemple de page « Connexions »

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu
Task list Bloc & Output list Connexions print list_compiled print list_threads

HTTP protocol			
...	**@ip**	**port**	**Host name**
Server	192.168.5.58	80	jbar-HLYL-WXX9
Last Client	192.168.122.1	39032	Jbar-HLYL-WXX9

TCP/IP protocol		
...	**@ip**	**port**
server	192.168.5.58	8000
client[0]	192.168.122.1	49062

Refresh

- 4.6 - Créer un bloc system

Pour ajouter un bloc à la bibliothèque standard, il faut :

1. Créer l'interface externe (interface graphique) du nouveau bloc.
2. Codez la fonction « Python » associée au nouveau bloc et ajouter le nom du bloc à la liste des blocs de la bibliothèque standard.

Notez qu'il faudra redémarrer l'**Exécuteur** et l'**Éditeur de blocs** pour prendre en compte ce nouveau bloc **system.**

- 4.6.1 - Créer l'interface externe

Ouvrir l'**Éditeur de blocs**.

Dans la barre de menu, cliquez sur « Bloc » puis « Change Properties », afin d'ouvrir la fenêtre « bloc info » suivante :

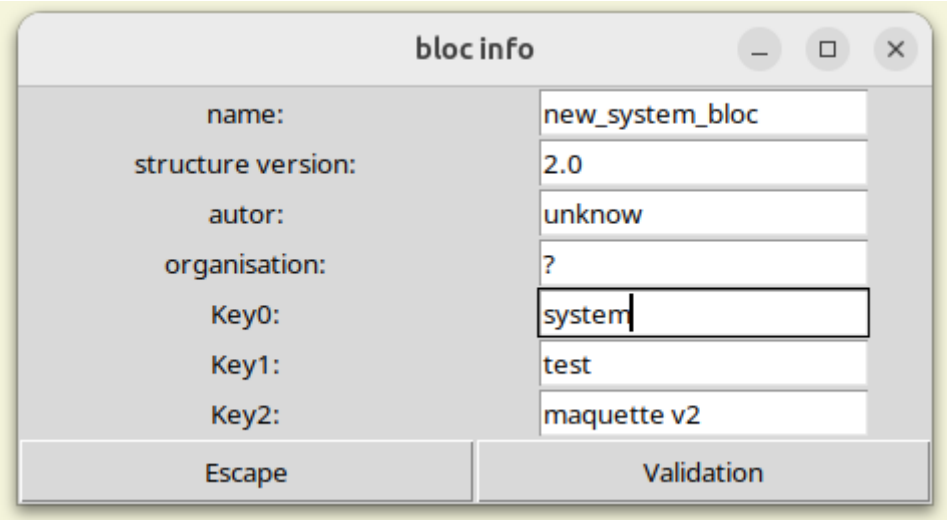


Figure : bloc info

Saisissez le nom du nouveau bloc **system** dans le champ « name: ». Saisissez « system » dans le champ « Key0: ». Confirmez votre saisie en cliquant sur « Vadidation »

Dans la zone graphique, ajoutez des blocs **system** <input> et/ou <output> selon le nombre d'entrées et de sorties nécessaires, renommez ces entrées/sorties et leurs donner une valeur par défaut, puis appuyez sur la touche [F3] pour sauvegardez.

Pour vérifier que l'interface a bien été créée, ouvrez une nouvelle fenêtre puis insérez un bloc **system**, le bloc fraîchement ajouté doit figuré dans la liste, sélectionnez le, et vérifiez que l'interface externe correspond à ce que vous souhaitez.

- 4.6.2 - Créer le code Python

Les fonctions associées aux bloc **system** sont dans le fichier /Target/exec.py, par exemple, voici la fonction Python associée au bloc **system** <and> :

```python
def c_exesubloc_and (pebloc, pieb, pio, pcounter):
    """ exécution du bloc a et b (dans la boucle récurcive)"""
    global exec_level
    exec_level +=1
    cesubloc = pebloc.sublocs[pieb]
    if debug_blocs:
        trace_txt = trace_proc(cesubloc, inspect.currentframe().f_code.co_name, exec_level)
        print (trace_txt, "début AND les paramètres sont: pieb=", pieb, ",   pio=", pio, ",   counter=", pcounter)
    if cesubloc.header['counter'] == pcounter:
        if debug_blocs: print (trace_txt, "  cesubloc['counter'] == pcounter: =", pcounter, "   (output[", pio, "] inchangée)")
    else:
        cesubloc.header['counter'] = pcounter
        try:
            pebloc.c_exebloc_recup_inputs(pieb, pcounter)
            cesubloc.c_exesubloc_validation_standard()

            cesubloc.outputs[0]['var'] = cesubloc.inputs[0]['var'] and cesubloc.inputs[1]['var']
        except:
            trace_txt = trace_proc(cesubloc, inspect.currentframe().f_code.co_name, exec_level)
            print (trace_txt, PARAM_TEXT_EXCEPTION)
            cesubloc.outputs[0]['valide'] = False
    if debug_blocs: print (trace_txt, " AND retourne l'outout [", pio, "]: var=", cesubloc.outputs[pio]['var'], "val=", cesubloc.outputs[pio]['valide'])
    exec_level -=1
    return cesubloc.outputs[pio]['var'], cesubloc.outputs[pio]['valide']
```

Figure: fonction « c_exesubloc_and »

Si on fait abstraction des éléments liés au débogage il ne reste que ceci :

```
1 def c_exesubloc_and (pebloc, pieb, pio, pcounter):
2       cesubloc = pebloc.sublocs[pieb]
3       if cesubloc.header['counter'] == pcounter:
4               pass
5       else:
6               cesubloc.header['counter'] = pcounter
7               try:
8                       pebloc.c_exebloc_recup_inputs(pieb, pcounter)
9                       cesubloc.c_exesubloc_validation_standard()
10                      cesubloc.outputs[0]['var'] = cesubloc.inputs[0]['var'] and cesubloc.inputs[1]['var']
11              except:
12                      cesubloc.outputs[0]['valide'] = False
13      return cesubloc.outputs[pio]
```

Explication du code :

Ligne 1 : Le nom de la fonction est « c_exsubloc_ » suivi du *nom du bloc system.* Les paramètres sont :

- **pebloc** : est la structure « exécutable » du bloc **user** en cours d'exécution.
- **pieb** : est l'index du bloc **system** à exécuter
- **pio** : est l'index de la sortie dont il faut retourner la valeur

Ligne 2 : permet de pointer sur le bloc à exécuter.

Ligne 3 : test si le bloc a déjà été exécuté

Ligne 6 : marque le bloc comme déjà exécuté

Ligne 8 : récupère toutes les entrées du bloc

Ligne 9: affecte tous les bits de validation des sorties du bloc en fonction de ses entrées

Ligne 10: affecte la sortie en fonction des entrées (ET logique)

Ligne 12 : affecte le bit de validité à « invalide » dans le cas où le calcul du bloc ne se soit pas passé correctement

Ligne 13 : retourne la sortie demandée (sortie d'index pio)

Dans la plupart des cas votre nouvelle fonction sera identique à celle de bloc **system** <and>, sauf :

- la ligne 1 : puisque le nom de la fonction sera différent.
- la ligne 10 : puisque vous affecterez les sorties en fonction des entrées selon vos besoins.

Toujours dans le fichier: /Target/exec.py, il faut ajouter l'association entre le nom de votre nouveau bloc **system** et la fonction Python fraîchement créée, ceux-ci se passe dans la fonction « recup_procedure ».

Notez que cette fonction est utilisée par l'**Éditeur de blocs** lorsque qu'un bloc **user** est compilé :

```
def recup_procedure(psubloc):
    proc_name = "recupe_procedure"
    """ ajout l'adresse de la procédure qui correspond au nom du bloc"""
    if   psubloc.header['name'] == PARAM_NAME_BLOC_DT:          procedure= c_exesubloc_dt
    elif psubloc.header['name'] == PARAM_NAME_BLOC_OUTPUT:      procedure= c_exesubloc_output
    elif psubloc.header['name'] == PARAM_NAME_BLOC_INPUT:       procedure= c_exesubloc_input
    elif psubloc.header['name'] == PARAM_NAME_BLOC_PREVIOUS:    procedure= c_exesubloc_previous
    elif psubloc.header['name'] == PARAM_NAME_BLOC_SELECT:      procedure= c_exesubloc_select
    elif psubloc.header['name'] == PARAM_NAME_BLOC_VALIDREAD:   procedure= c_exesubloc_validRead
    elif psubloc.header['name'] == PARAM_NAME_BLOC_VALIDWRITE:  procedure= c_exesubloc_validWrite
    elif psubloc.header['name'] == PARAM_NAME_BLOC_COMP:        procedure= c_exesubloc_comp
    elif psubloc.header['name'] == PARAM_NAME_BLOC_AND:         procedure= c_exesubloc_and
    elif psubloc.header['name'] == PARAM_NAME_BLOC_OR:          procedure= c_exesubloc_or
    elif psubloc.header['name'] == PARAM_NAME_BLOC_NOT:         procedure= c_exesubloc_not
    elif psubloc.header['name'] == PARAM_NAME_BLOC_EDGE:        procedure= c_exesubloc_edge
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CABLIN:      procedure= c_exesubloc_cablin
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CABLOUT:     procedure= c_exesubloc_cablout
    elif psubloc.header['name'] == PARAM_NAME_BLOC_ADD:         procedure= c_exesubloc_add
    elif psubloc.header['name'] == PARAM_NAME_BLOC_SUB:         procedure= c_exesubloc_sub
    elif psubloc.header['name'] == PARAM_NAME_BLOC_MULT:        procedure= c_exesubloc_mult
    elif psubloc.header['name'] == PARAM_NAME_BLOC_DIV:         procedure= c_exesubloc_div
    elif psubloc.header['name'] == PARAM_NAME_BLOC_MINMAX:      procedure= c_exesubloc_minmax
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CONST_PI:    procedure= c_exesubloc_const_pi
    else:
        print (proc_name, " ERROR: function not defined for this bloc <"+psubloc.header['name']+">")
    return procedure
```

Figure : fonction « recup_procedure »

Pour cela vous devrez ajouter une ligne avant le « else » est créer un nouveau paramètre dont le nom sera constitué de :

« PARAM_NAME_BLOC_ » suivi du nom de votre nouveau bloc **system** en majuscule.

Il faudra aussi modifier le fichier /Target/PARAM_NAME_BLOC.py, en y ajouter une ligne pour définir la constante « PARARM_NAME_BLOC_*votrebloc* », la chaine de caractère située à droite su signe « = » correspond au nom de ficher votre nouveau bloc **system.**

```
# ATTENTION: la source de ce fichier ce trouve dans le répertoire "Target"

# nom des blocs "system"
PARAM_NAME_BLOC_DT = "dt"
PARAM_NAME_BLOC_OUTPUT = "output"
PARAM_NAME_BLOC_INPUT = "input"
PARAM_NAME_BLOC_PREVIOUS = "previous"
PARAM_NAME_BLOC_SELECT = "select"
PARAM_NAME_BLOC_VALIDREAD = "validRead"
PARAM_NAME_BLOC_VALIDWRITE = "validWrite"
PARAM_NAME_BLOC_MINMAX = "minmax"
PARAM_NAME_BLOC_COMP = "comp"
PARAM_NAME_BLOC_AND = "and"
PARAM_NAME_BLOC_OR = "or"
PARAM_NAME_BLOC_NOT = "not"
PARAM_NAME_BLOC_EDGE = "edge"
PARAM_NAME_BLOC_ADD = "add"
PARAM_NAME_BLOC_SUB = "sub"
PARAM_NAME_BLOC_MULT = "mult"
PARAM_NAME_BLOC_DIV = "div"
PARAM_NAME_BLOC_CABLIN = "cablin"
PARAM_NAME_BLOC_CABLOUT = "cablout"
PARAM_NAME_BLOC_CONST_PI = "const.pi"

```

Figure : fichier PARAM_NAME_BLOC.py