

Séance 1 – Document de reference

Historiquement.....	3
Gamme automates.....	5
Comparaison entre CompactLogix et ControlLogix	6
Le PAC.....	7
Différence entre PLC et PAC.....	9
Cas d'usages PLC vs PAC	10
Ecosystème Rockwell	11
Type de donnée.....	14
Qu'est-ce qu'un UDT ?	15
Comparatif rapide Siemens	16
Allen-Bradley	16
Siemens.....	16
Équivalence.....	16
AOI + UDT.....	16
FB + DB instance.....	16
Logique + données	16
Controller Fault Handler – Une spécificité Allen-Bradley.....	17
Description de l'environnement	19
Les différents type de Task	21
Définir une Task et sa priorité	23
Astuce : Tags et sauvegarde	25
Création et utilisation d'un UDT et de programmes dans Studio 5000.....	26

Historiquement

Allen-Bradley trouve ses origines au tout début du XX^e siècle, lorsque Lynde Bradley, jeune ingénieur électricien, met au point une résistance de contrôle destinée à réguler la vitesse des moteurs à courant continu. En 1904, il s'associe avec le Dr Stanton Allen, convaincu du potentiel de cette invention. Ensemble, ils fondent à Milwaukee, dans le Wisconsin, la Compression Rheostat Company, qui deviendra en 1910 la Allen-Bradley Company, en hommage à ses deux fondateurs.

Au fil des décennies, l'entreprise se spécialise dans les composants électriques tels que les boutons-poussoirs, contacteurs et minuteriers, bâtissant ainsi sa réputation dans le domaine industriel. Cette expertise l'amène naturellement à s'imposer dans le secteur émergent de l'automatisation.

En 1985, Allen-Bradley est rachetée par Rockwell International pour 1,651 milliard de dollars, l'une des acquisitions industrielles les plus importantes de l'époque. Cette opération stratégique permet à Rockwell de renforcer sa présence dans l'automatisation industrielle en s'appuyant sur le savoir-faire d'Allen-Bradley dans les automates programmables et les interfaces homme-machine.

Aujourd'hui, Allen-Bradley constitue la marque phare de Rockwell Automation pour le matériel d'automatisation. Elle regroupe les automates MicroLogix, CompactLogix et ControlLogix, les entrées/sorties déportées, les variateurs de vitesse PowerFlex, les interfaces opérateurs PanelView, ainsi qu'un large éventail de capteurs et de solutions de sécurité. Combinée aux suites logicielles FactoryTalk de Rockwell Automation, cette gamme permet de proposer des systèmes d'automatisation et de supervision complets, largement utilisés dans l'industrie, notamment en Amérique du Nord.

Aujourd'hui, Allen-Bradley est la marque phare de Rockwell Automation pour le matériel d'automatisation et domine le marché américain non seulement grâce à la qualité et la performance de ses produits, mais aussi grâce à une implantation historique profonde. Présente dès les débuts de l'automatisation aux États-Unis, la marque a su se positionner comme un partenaire incontournable des grands secteurs industriels tels que l'automobile, la pétrochimie et l'agroalimentaire. La standardisation précoce de leurs solutions dans de nombreuses usines a renforcé cette position, rendant coûteux et complexe tout changement de fournisseur. En s'appuyant sur la synergie avec Rockwell Automation pour offrir des solutions intégrées matériel-logiciel parfaitement adaptées aux besoins nord-américains, Allen-Bradley a consolidé une fidélité client exceptionnelle et une position de leader incontesté sur son marché domestique.



Le PLC-2 d'Allen-Bradley est représentatif des premières générations d'automates programmables industriels conçus dans les années 1980. Ce modèle, robuste et modulaire, permettait déjà de programmer et superviser des processus industriels, mais avec des outils et des capacités limités par rapport aux standards actuels.

Le logiciel de programmation fonctionnait sous des systèmes d'exploitation comme Windows 95, 98, 2000, XP ou Vista, avec compatibilité en mode 16 bits sous Windows 8. La programmation se faisait en langage Ladder, avec possibilité de travailler hors ligne, de transférer le programme vers ou depuis le PLC, et de sauvegarder les projets sur disque dur.

Les fonctionnalités disponibles incluaient l'affichage en ligne, la supervision, le transfert de programmes, ainsi que la gestion de commandes de base telles que Examine ON (I |), Examine OFF (I / |), Output (), verrouillage et déverrouillage de sortie (Latch / Unlatch), effacement général (Master Clear), temporisation à l'activation (TON) et à la désactivation (TOF).

Si ces automates étaient considérés comme très avancés à l'époque, et reconnus pour leur grande robustesse, leur architecture restait strictement séquentielle, leurs capacités réseau limitées, et leur puissance de traitement relativement modeste. Les modèles modernes, comme les CompactLogix ou ControlLogix, ont depuis intégré des fonctions de communication avancées (Ethernet/IP, OPC UA), des traitements plus puissants, et une architecture orientée objet permettant de gérer des systèmes complexes et interconnectés, marquant ainsi la transition du PLC classique vers le PAC.

Gamme automates

Micro800

Description :

Série d'automates compacts destinés aux petites machines ou systèmes autonomes. Idéals pour les applications simples nécessitant peu d'entrées/sorties et une logique séquentielle de base.

Caractéristiques :

- Programmation avec Connected Components Workbench (CCW)
- Modules compacts et modulaires pour ajouter des E/S ou des communications
- Compatibles avec Ethernet/IP sur certains modèles
- Bon rapport qualité/prix pour petites applications

Limitation :

Capacité mémoire et puissance de calcul limitées, peu adaptées aux systèmes complexes ou aux environnements nécessitant de nombreuses communications et fonctionnalités avancées.



CompactLogix

Description :

Automates de taille moyenne, adaptés aux lignes de production ou aux systèmes nécessitant un contrôle plus avancé que les Micro800, tout en restant compacts et modulaires.

Caractéristiques :

- Programmation avec Studio 5000 Logix Designer
- Compatibles Ethernet/IP natif
- Support du contrôle de mouvement (motion control) pour plusieurs axes
- Mémoire et puissance de calcul adaptées à des applications industrielles de taille moyenne
- Modules E/S locaux et distants

Limitation :

Moins extensibles et moins puissants que la gamme ControlLogix, donc pas idéaux pour les systèmes très complexes ou critiques.



ControlLogix

Description :

Gamme haut de gamme (PAC) destinée aux applications complexes et critiques, intégrant un haut niveau de puissance, de modularité et de communication.

Caractéristiques :

- Architecture modulaire orientée objet
- Programmation avec Studio 5000 Logix Designer
- Multi-protocoles : Ethernet/IP, ControlNet, DeviceNet, et possibilité d'ajouter d'autres modules de communication
- Contrôle de mouvement avancé, synchronisation multi-axes en temps réel
- Grande capacité mémoire et traitement rapide pour process continus et systèmes complexes
- Compatible avec les solutions logicielles Rockwell Automation (FactoryTalk, MES, SCADA)

Limitation :

Coût plus élevé à l'achat et à la maintenance, nécessitant une équipe formée à la programmation et au diagnostic des PAC Rockwell.



Comparaison entre CompactLogix et ControlLogix

Point de comparaison	CompactLogix	ControlLogix
Positionnement	Gamme moyenne, adaptée aux petites et moyennes machines (200 à 300 points d'entrée sortie)	Gamme haut de gamme, destinée aux grandes applications industrielles
Taille du système	Plus compacte, pour machines individuelles ou petits process industriels	Très modulaire, idéale pour grandes usines, réseaux complexes et grand nombre de points d'E/S
Montage	Boîtier unique (compact) ou petit rack pour les modèles étendus	Châssis avec modules enfichables (rack)
CPU / Processeur	Moins puissant, une seule CPU par contrôleur	Très puissant, possibilité de plusieurs CPU dans un même châssis
Mémoire programme	Mémoire plus réduite	Grande capacité mémoire
Communication réseau	Principalement EtherNet/IP, plus quelques réseaux basiques	Supporte de nombreux réseaux industriels : EtherNet/IP, ControlNet, DeviceNet, etc. (pour communiquer avec des variateurs ou des robots)
Redondance / Haute disponibilité	Pas de redondance native	Possible (CPU redondantes, communication redondante)
Extensions	Nombre limité de modules d'extension	Large choix de modules d'extension : spécifiques, sécurité, motion control, etc.
Utilisation typique	Machines isolées, petites lignes de production, automatisation locale	Usines complètes, process continus, infrastructures critiques
Logiciel de programmation	Studio 5000 Logix Designer (même environnement que ControlLogix)	Studio 5000 Logix Designer
Prix indicatif	≈ 2 500 €	≈ 7 910 €

Le PAC

À la fin des années 1960, les premiers automates programmables industriels (PLC) sont apparus pour remplacer les systèmes à relais câblés, qui étaient lourds à modifier et difficiles à dépanner. Ces PLC exécutaient principalement de la logique séquentielle simple, adaptée aux machines et aux petites lignes de production.

Avec l'évolution de l'industrie, les besoins ont changé :

- Plus de puissance de calcul pour gérer des procédés complexes
- Plus d'intégration réseau pour communiquer avec d'autres machines, des bases de données ou des systèmes de supervision
- Plus de flexibilité logicielle pour adapter rapidement les systèmes aux nouvelles exigences

C'est dans ce contexte qu'est apparu le PAC (Programmable Automation Controller), à partir des années 2000, popularisé notamment par Rockwell Automation avec ses gammes CompactLogix et ControlLogix.

Le PAC reprend la fiabilité et la robustesse du PLC, tout en y ajoutant les capacités avancées d'un PC industriel, afin de répondre aux besoins de l'Industrie 4.0.

Un PAC est un contrôleur industriel moderne qui combine les fonctions d'un automate programmable classique (PLC) avec la puissance et la flexibilité d'un PC industriel.

Il est conçu pour gérer non seulement la logique séquentielle des machines, mais aussi des processus complexes, avec un haut niveau d'intégration réseau et de traitement de données.

Caractéristique

1. Architecture avancée

- Modulaire et orientée objet
- Souvent basée sur un système d'exploitation temps réel
- Permet une évolution et une personnalisation plus facile du système

2. Polyvalence de programmation

- Langages IEC 61131-3 (Ladder, FBD, ST, SFC...)
- Support possible de langages avancés (C/C++, scripts, etc.)

3. Capacités réseau étendues

- Multi-protocoles : Ethernet/IP, OPC UA, Modbus, etc.
- Intégration avec systèmes IT/OT, bases de données SQL, Web services

4. Puissance de calcul

- Adaptée aux systèmes complexes, calculs intensifs et synchronisation en temps réel

5. Applications typiques

- Lignes de production intégrées
- Contrôle de mouvement (motion control) multi-axes
- Robotique, vision industrielle, procédés continus

En résumé :

Le PAC est l'évolution du PLC, conçu pour répondre aux besoins de l'Industrie 4.0, où l'automatisation, la communication et le traitement intelligent des données sont indispensables.

Différence entre PLC et PAC

Critère	PLC (Automate Programmable Industriel)	PAC (Contrôleur d'Automatisation Programmable)
Origine	Développé pour remplacer les relais et systèmes câblés	Évolution du PLC, avec une architecture orientée PC et objets
Architecture	Architecture traditionnelle, séquentielle	Architecture modulaire, orientée objet, souvent sur OS temps réel
Langages de programmation	Langages IEC 61131-3 (LD, FBD, ST, SFC...)	Les mêmes langages IEC, avec en plus C/C++, scripts, fonctions avancées
Capacités réseau	Limitée : Modbus, Profibus, Ethernet/IP	Étendue : Ethernet/IP, OPC UA, SQL, Web Services, intégration IT/OT
Puissance de traitement	Adaptée aux applications simples à moyennes	Plus élevée, adaptée aux systèmes complexes et au temps réel
Applications typiques	Machines industrielles, petites lignes de production	Systèmes complexes, robotique, vision, contrôle multi-axes
Maintenance	Simplicité, robustesse, maintenance facile	Plus complexe, mais diagnostics et analyse intégrés
Exemples de constructeurs	Siemens S7-1200/1500, Schneider M221/M241, Omron CP1H	Rockwell CompactLogix/ControlLogix, Siemens S7-1500T, Schneider M580, Beckhoff

Cas d'usages PLC vs PAC

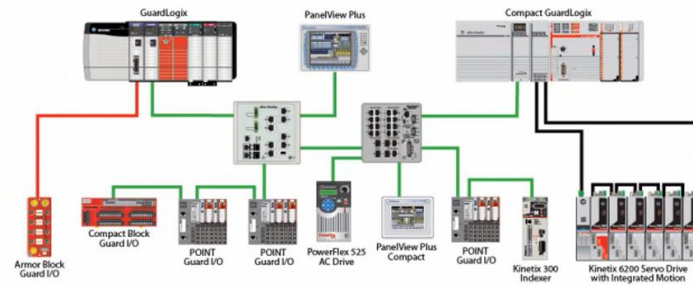
PLC (Automate Programmable Industriel)

Cas d'usage typique	Description	Exemple concret
Machine d'assemblage simple	Commande de moteurs, vérins, capteurs sur une petite machine autonome	Une sertisseuse dans une ligne de production
Transporteurs	Gestion marche/arrêt, détection pièces, séquences simples	Convoyeur de palettes dans un entrepôt
Station de pompage	Gestion des pompes, niveaux, alarmes, cycles de démarrage	Poste de relevage dans un réseau d'eau
Machine d'emballage	Séquences répétitives avec peu de calculs complexes	Ensacheuse pour produits alimentaires

PAC (Contrôleur d'Automatisation Programmable)

Cas d'usage typique	Description	Exemple concret
Ligne de production intégrée	Coordination de plusieurs machines, échanges de données avec ERP/MES	Ligne complète de fabrication de bouteilles
Motion Control multi-axes	Synchronisation précise de plusieurs moteurs en temps réel	Imprimante industrielle haute vitesse
Robotique et vision industrielle	Contrôle des robots + traitement d'images	Cellule robotisée pour soudage avec inspection caméra
Procédé continu complexe	Régulation multi-boucles, sécurité, analyse en ligne	Raffinerie, usine chimique, production pharmaceutique
Intégration IT/OT	Collecte et envoi de données vers le cloud, analyse big data	Usine connectée Industrie 4.0 avec prédiction de maintenance

Ecosystème Rockwell



Rouge : Sécurité machine

Vert : Process

Noir : Motion

L'image présente l'écosystème matériel Allen-Bradley, qui constitue l'offre intégrée de Rockwell Automation pour l'automatisation industrielle. Cet ensemble de produits couvre toutes les fonctions clés d'une installation : contrôle, sécurité, communication, mouvement et supervision.

Chaque élément de cet écosystème est conçu pour fonctionner de manière native dans l'environnement logiciel Rockwell, principalement Studio 5000 Logix Designer pour la programmation et FactoryTalk pour la supervision et l'analyse. L'ensemble repose sur des protocoles communs comme EtherNet/IP, CIP Safety et CIP Motion, garantissant une interopérabilité totale entre tous les composants.

Cet écosystème permet de bâtir aussi bien de petites applications autonomes que des systèmes complexes et interconnectés, en conservant une cohérence matérielle et logicielle sur l'ensemble de la chaîne d'automatisation.

1. Contrôleurs GuardLogix et Compact GuardLogix

- GuardLogix (à gauche) et Compact GuardLogix (à droite) sont des automates programmables avec sécurité intégrée (SIL 3 / PLe).
- Ils gèrent à la fois la logique standard (production) et les fonctions de sécurité (arrêt d'urgence, interverrouillage, contrôle sécuriser de mouvement).
- Le ControlLogix GuardLogix est modulaire (rack enfichable), tandis que le Compact GuardLogix est plus compact pour des installations de taille moyenne.

2. I/O de sécurité et standard

- Armor Block Guard I/O et Compact Block Guard I/O : modules d'E/S de sécurité IP67 pour montage déporté sur machine, souvent près des actionneurs/capteurs.
- POINT Guard I/O : modules d'E/S sécurisés au format Point I/O, souvent montés dans des coffrets.
- Ces I/O sont reliés aux automates via EtherNet/IP en respectant la communication sécurisée CIP Safety.

3. Interfaces opérateur (HMI PanelView Plus)

- Les PanelView Plus permettent l'affichage, la supervision et le pilotage des machines par les opérateurs.
- Connectés à l'automate via Ethernet/IP, ils affichent aussi les alarmes et les informations de sécurité.

4. Variateurs et servo variateurs

- Power Flex 525 AC Drive : variateur de fréquence pour le contrôle de moteurs asynchrones (vitesse, couple) avec fonctions d'efficacité énergétique.
- Kinetix 300, 6200 Servo Drive : servo variateurs pour le contrôle précis de moteurs synchrones ou asynchrones, avec intégration du contrôle de mouvement via EtherNet/IP CIP Motion.
- Ces variateurs peuvent intégrer des fonctions de sécurité de mouvement comme STO (*Safe Torque Off*).

5. Communication industrielle

- Le réseau principal est basé sur EtherNet/IP, qui transporte à la fois les données standard et les données de sécurité (CIP Safety).
- L'architecture peut aussi intégrer ControlNet (réseau temps réel déterministe) ou DeviceNet (pour capteurs/actionneurs intelligents), selon les besoins.

Avantages de cette architecture

- **Sécurité intégrée** : un seul automate gère à la fois la logique standard et la sécurité, simplifiant le câblage et réduisant les coûts.
- **Modularité** : possibilité d'ajouter ou de retirer des modules selon l'évolution des besoins.
- **Flexibilité** : architecture adaptable du petit poste autonome à l'usine complète.

Interopérabilité : tous les composants communiquent via des protocoles standardisés (EtherNet/IP, CIP Safety).

L'automate a en deux fonctions principales dans son architecture interne :

1. **Traitement / Exécution (CPU)**

- C'est le processeur de l'automate, qui exécute le programme logique (Ladder, ST, FBD...) en suivant le **cycle d'exécution** : lecture des entrées → traitement du programme → mise à jour des sorties.
- Il gère aussi les calculs, la communication réseau, le contrôle de mouvement, la sécurité (si intégrée).

2. **Mémoire**

- C'est là que sont stockés les programmes, les données temporaires et persistantes, ainsi que les variables système.
- Dans un automate, on parle souvent d'**image mémoire** : au début de chaque cycle, l'automate lit toutes les entrées physiques et les copie dans une zone mémoire appelée *image des entrées*. Il exécute ensuite le programme en utilisant cette copie, puis écrit les résultats dans *l'image des sorties*, qui sera envoyée aux sorties physiques en fin de cycle.
- La mémoire est divisée en zones :
 - **Programme** : instructions et routines
 - **Données** : tags, variables, tables
 - **Mémoire système** : pour le fonctionnement interne

RAM → les transistors gardent leur état seulement tant qu'il y a du courant. Dès qu'on coupe, ils "oublient".

Mémoire Flash/EEPROM → les transistors sont conçus pour emprisonner des électrons dans une zone isolée (floating gate). Même sans courant, les électrons restent piégés, donc la mémoire « se souvient » pendant des années.

Type de donnée

Type de donnée	Description	Valeurs possibles	Taille
BOOL	Booléen	FALSE (0) ou TRUE (1)	1 bit
BYTE	Octet	0 à 255 ($2^8 - 1$)	8 bits = 1 octet
WORD	Mot de 16 bits	0 à 65 535 ($2^{16} - 1$)	16 bits = 2 octets
DWORD	Mot de 32 bits	0 à 4 294 967 295 ($2^{32} - 1$)	32 bits = 4 octets
Nombre entier			
USINT	Entier non signé court (8 bits)	0 à 255	8 bits
SINT	Entier signé court (8 bits)	-128 à 127	8 bits
INT	Entier signé (16 bits)	-32 768 à 32 767	16 bits
DINT	Entier signé (32 bits)	-2 147 483 648 à 2 147 483 647	32 bits
UDINT	Entier non signé (32 bits)	0 à 4 294 967 295	32 bits
LINT	Entier signé (64 bits)	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	64 bits
ULINT	Entier non signé (64 bits)	0 à 18 446 744 073 709 551 615	64 bits
Nombre réels			
REAL	Nombre à virgule flottante (32 bits)	$\approx -3,14 \times 10^{38}$ à $3,14 \times 10^{38}$	32 bits
LREAL	Nombre à virgule flottante (64 bits)	$\approx -1,7 \times 10^{308}$ à $1,7 \times 10^{308}$	64 bits

Qu'est-ce qu'un UDT ?

Définition simple

Un User Data Type est un type de données structuré personnalisé qui regroupe plusieurs variables de types différents (BOOL, INT, REAL, TIMER...) sous un même nom.

C'est l'équivalent d'une STRUCT dans d'autres environnements comme Siemens.

Principe

- On définit un modèle (la structure et le nom des champs).
- On crée des instances (tags) qui utilisent ce modèle et héritent automatiquement de sa structure.
- Cela s'inscrit dans une logique de programmation orientée objet : on définit un "gabarit" réutilisable, puis on crée des "objets" à partir de ce gabarit.

Exemple de modèle UDT :

UDT Moteur {

Cmd_Marche : BOOL

Cmd_Arret : BOOL

Consigne : REAL

Vitesse : REAL

Temp : REAL

Default : BOOL

CodeAlarme : DINT

T_Repos : TIMER

}

Pourquoi c'est utile ?

- **Lisibilité & standardisation** : tous les équipements similaires partagent la même structure.
- **Réutilisation** : un simple copier/coller instancie des dizaines de variables à partir d'un seul modèle.
- **Maintenance simplifiée** : si tu ajoutes un champ dans l'UDT, toutes les instances sont mises à jour.
- **Flexibilité** : un UDT peut être un tableau (Moteur[10]) ou imbriqué (ex. UDT "Alarme" dans UDT "Moteur").

Création dans Studio 5000

1. Controller Organizer → Data Types → User-Defined.
2. Définir les champs (nom, type, commentaire).
3. Déclarer les tags en Controller Tags ou Program Tags avec ce type.
4. Les HMI (PanelView, FactoryTalk) accèdent directement aux sous-membres : MTR_Convoyeur.Vitesse.

Bonnes pratiques

- **Préfixes clairs** : Cmd_ pour les commandes, Sts_ pour les états, Meas_ pour les mesures, Alm_ pour les alarmes.
- **Séparer Commande et Status** : créer 2 UDT si nécessaire pour éviter les confusions.
- **Initialiser les valeurs par défaut** (consignes, temporisations).
- **Utiliser des tableaux** plutôt que des tags séparés (Moteur[20]).
- **Associer avec AOI** : l'UDT contient les données, l'AOI la logique (ex. AOI_Motor utilisant MotorCmd et MotorSts).

Points de vigilance

- **Modification d'un UDT** : peut impacter HMI, scripts, MES → tester en maquette.
- **Copie sécurisée** : utiliser CPS (Copy Synchronous) pour éviter les lectures partielles.
- **Optimisation mémoire** : regrouper les BOOL ensemble, puis les REAL/DINT pour éviter le padding.
- **Synchronisation inter-tâches** : prévoir un bit "DataValid" ou un double-buffer si partagé entre tâches.

Comparatif rapide Siemens

Allen-Bradley	Siemens	Équivalence
UDT	STRUCT / UDT	Structure de données
AOI + UDT	FB + DB instance	Logique + données

Controller Fault Handler – Une spécificité Allen-Bradley

Un atout propre à Allen-Bradley

Contrairement à d'autres constructeurs (ex. Siemens, Schneider, Omron) où la gestion des erreurs critiques est souvent implicite ou dispersée dans différents blocs système, Allen-Bradley propose une routine dédiée et centralisée pour traiter les fautes :
le Controller Fault Handler.

Cela donne au programmeur un point unique pour :

- Détecter les anomalies critiques
- Réagir immédiatement
- Conserver la maîtrise sur l'état final du système

Chez Siemens, par exemple, il faudrait combiner des OB spécifiques (OB80, OB121, OB122...) pour obtenir une logique équivalente, et les réactions ne sont pas regroupées dans une seule routine universelle.

Est-ce que c'est de la sécurité (Safety) ?

Non, ce n'est pas une logique Safety au sens normatif (EN ISO 13849 ou IEC 61508).

- Un Fault Handler reste un élément logiciel, dépendant du bon fonctionnement du CPU et de son programme.
- En cas de panne matérielle grave, plantage CPU ou perte totale d'alimentation, il ne pourra pas s'exécuter.
- Les systèmes Safety sont conçus avec du matériel certifié (CPU et modules de sécurité), des circuits redondants et des diagnostics matériels.

On peut donc se "fier" au Fault Handler pour améliorer la robustesse logicielle et réagir aux erreurs logicielles ou I/O, mais jamais pour remplacer un arrêt d'urgence ou une chaîne de sécurité matérielle.

Rôle idéal dans une architecture

- En production standard : détecter et mettre en sécurité les actionneurs si un calcul ou un accès mémoire pose problème.
- En complément du Safety :
 - Le Safety arrête physiquement les énergies dangereuses (coupure STO, arrêt pompe, etc.).
 - Le Fault Handler, lui, documente la faute, ferme proprement les séquences, prévient l'opérateur et prépare un redémarrage sûr.

Ce que ça change pour le programmeur

- Centralisation : toutes les actions d'urgence logicielle au même endroit.
- Réduction du temps de diagnostic (une seule routine à consulter).
- Meilleure expérience HMI : affichage automatique de messages d'erreur détaillés.
- Plus grande homogénéité d'un projet à l'autre : dans un standard entreprise, on peut toujours savoir où et comment sont gérées les fautes.

Schéma simplifié

1. Programme principal → exécution normale
2. Erreur détectée par CPU (division par zéro, perte I/O...)
3. Interruption → appel de la routine Fault Handler
4. Routine :
 - Couper sorties critiques
 - Enregistrer log
 - Afficher message opérateur
5. Retour ou mise en fault contrôlée selon configuration

Conclusion :

Le Controller Fault Handler est un outil puissant et propre à l'écosystème Rockwell pour la gestion proactive des erreurs logicielles.

Il renforce la disponibilité et la traçabilité, mais ne remplace pas un système de sécurité certifié.

En l'intégrant bien, on obtient une machine plus robuste, plus facile à diagnostiquer et plus rapide à remettre en service.

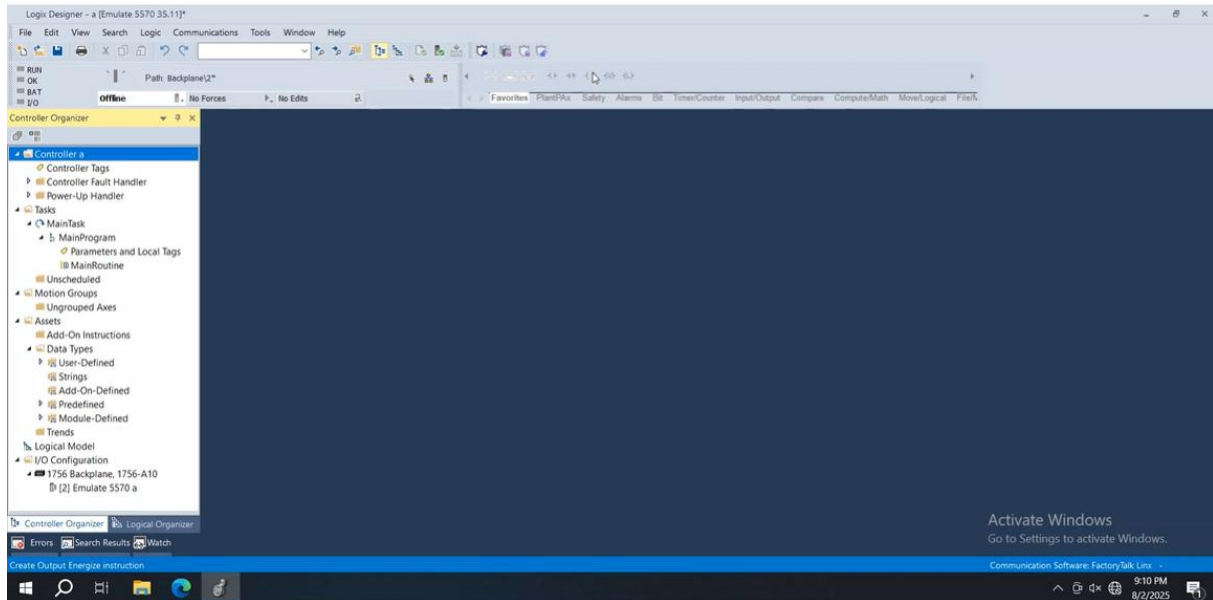
Description de l'environnement

Contient tout les fonctionnalité standard tel que, l'état du contrôleur lorsqu'il est en ligne et les erreur qui peuvent être présente lors de la simulation, ensuite les fonctionnalités standard

En haut à gauche l'état du contrôleur

A droite les outils pour la programmation, et définis sous forme d'onglet, Entrée, bobine ...

L'arborescence appeler Controller Organizer, c'est ici ou réside tous les outils pour maintenir un projet



Controller Tags

Il y'a une notion de Alias for, enfaite de ce que j'ai compris la variable va être dépendante d'une autre variable qui va être son image

Controller Tags - a(controller)									
Name	Alias For	Base Tag	Data Type	Description	External Access	Constant	Style		
TAG1	TAG2	TAG2	BOOL		Read/Write	☐	Decimal		
TAG2			BOOL		Read/Write	☐	Decimal		
						☐			

Controller Fault Handler

Routine évoqué auparavant qui s'exécute automatiquement lorsqu'on a une faute majeur d'un contrôleur, une erreur critique qui empêche l'exécution normale programme. Concrètement ce sont les variables globales.

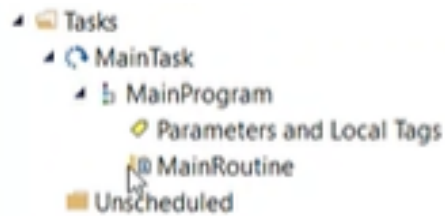
Le paramètres local de l'état local de ce programme, donc c'est une partie programmable également. Donc concrètement on peut mettre la machine en sécurité en cas d'erreur grave, d'enregistrer le code d'erreur ou bien envoyer des alarmes a l'opérateur. Couper les sorties avant que l'automate ne passe en mode faux, c'est-à-dire dire de s'arrêter.

Power Up Handler

Routine a chaque mise en tension, ça aide à réinitialiser des variables au démarrage à être assuré des sorties avant de lancer la production, d'initialiser une séquence pour la remettre à zéro, comme des compteur par exemple.

Tasks

Le programme d'API repose sur trois niveaux hiérarchiques, les taches ou on trouve les programmes, et dans les programme on trouve les routines.



Qu'est-ce qu'une Task, une task est une unité d'exécution dans Studio 5000 ?

Elle définit quand et comment les programmes (routines) sont exécuté par l'automate.

Les différents type de Task

Dans Studio 5000, le fonctionnement de l'automate repose sur l'exécution de tâches (Tasks).

Chaque tâche détermine comment et quand le programme est exécuté. Trois types principaux existent :

Continuous Task

- Fonctionnement : exécutée en **boucle continue**, sans interruption, jusqu'à ce que la CPU n'ait plus d'instructions à traiter.
- Avantages :
 - Simple à mettre en œuvre.
 - Idéale pour de **petits programmes** qui n'ont pas de contraintes de temps strictes.
- Inconvénients :
 - Peut ralentir la CPU si le programme est long.
 - Pas de garantie de cycle fixe → temps d'exécution variable.

Utilisation recommandée : programmes simples, sans exigences fortes en temps réel.

Periodic Task

- Fonctionnement : exécutée à **intervalle fixe** défini par l'utilisateur (exemple : toutes les 10 ms).
- Particularité : peut **interrompre** une Continuous Task si nécessaire.
- Avantages :
 - Garantit un cycle de contrôle stable et prévisible.
 - Indispensable pour les applications temps réel, comme les boucles **PID**.
- Inconvénients :
 - Si l'intervalle est trop court, risque de surcharge CPU.

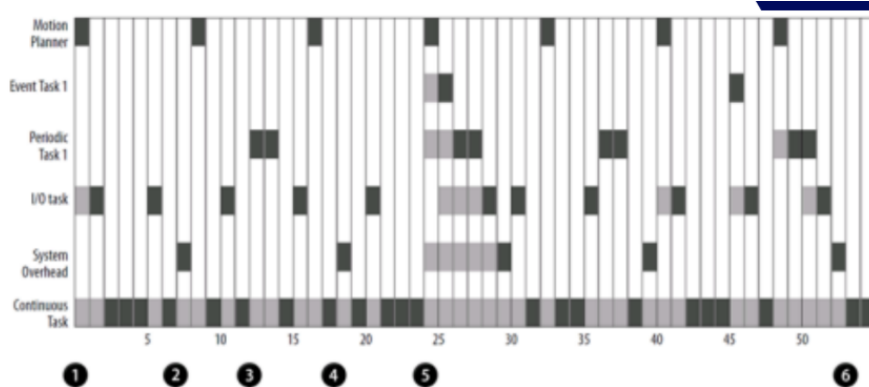
Utilisation recommandée : contrôle de procédés nécessitant de la régularité (régulation, asservissement moteur, traitement cyclique).

Event Task

- Fonctionnement : déclenchée uniquement lorsqu'un **événement** survient.
 - L'événement peut être : une interruption, un changement de valeur de tag, ou un signal provenant d'un module.
- Avantages :
 - Réaction **très rapide et prioritaire**.
 - Exécution uniquement quand c'est nécessaire, donc optimisation de la charge CPU.
- Inconvénients :
 - Demande une configuration plus poussée (définition des événements déclencheurs).

Utilisation recommandée : gestion d'entrées critiques (ex. arrêt d'urgence, top codeur incrémental, interruption module de communication).

Exemple visuel d'ordonnancement



Définir une Task et sa priorité

1. Création d'une Task

Dans Studio 5000 :

1. Clic droit sur Tasks dans l'arborescence.
2. Sélectionner New Task.
3. Choisir le type de tâche (Continuous, Periodic ou Event).
4. Configurer ses propriétés, notamment la priorité.

Comprendre la logique de priorité

La priorité d'une tâche est définie par un nombre entier.

- Plus le nombre est petit, plus la priorité est élevée.
- Une tâche à priorité 1 est exécutée avant une tâche à priorité 10.

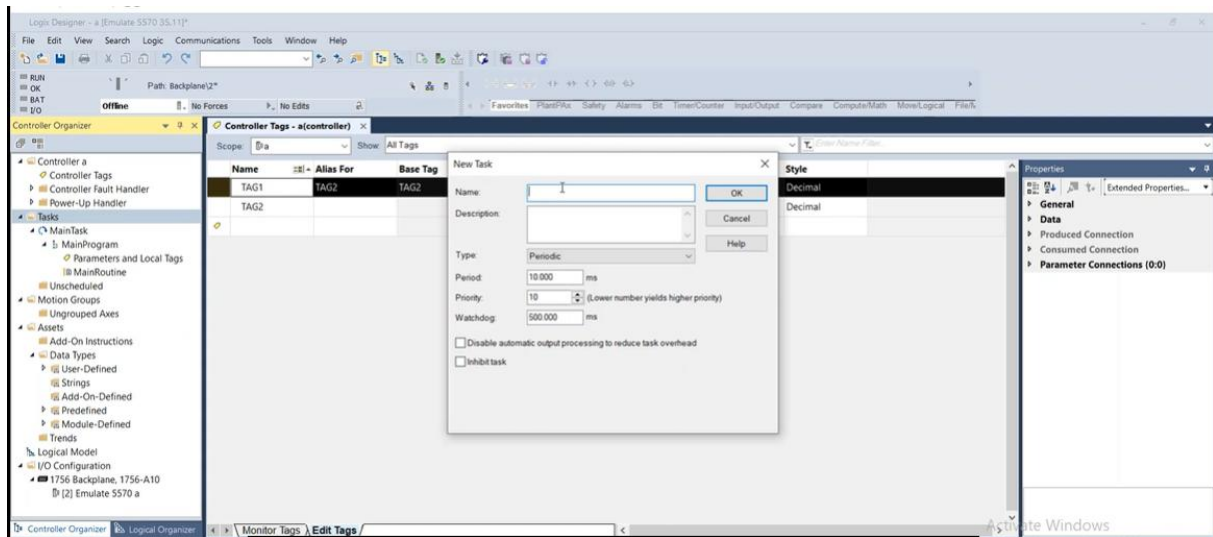
C'est l'inverse d'un classement scolaire : ici, 1 est "VIP" et 10 est "en attente".

Exemple de configuration

- Priorité 1 → Tâches critiques, urgentes, nécessitant un temps de réaction très rapide (ex. Event Task liée à un arrêt d'urgence).
- Priorité 5 → Tâches intermédiaires (ex. boucles de régulation PID exécutées périodiquement).
- Priorité 10 → Tâches non critiques, pouvant attendre (ex. Continuous Task de supervision ou calculs non urgents).

Bonnes pratiques

- Réserver les priorités basses (1 à 3) pour les tâches vitales au process.
- Utiliser les priorités moyennes (4 à 7) pour les contrôles réguliers et les boucles PID.
- Affecter les priorités hautes (8 à 15) aux tâches de fond, de supervision ou de diagnostic.
- Éviter de donner une priorité trop élevée à des tâches longues, car elles risquent de bloquer les autres.



« (Lower number yields higher priority) » (Un nombre plus bas donne une priorité plus élevée)

Astuce : Tags et sauvegarde

À ne pas confondre :

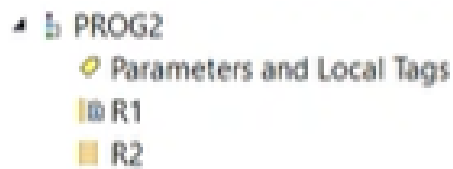
- **Controller Tags** → variables globales accessibles dans tout le projet.
- **Parameters and Local Tags** → variables locales propres au programme ou à la routine.

Icône disquette :

Lorsqu'une petite disquette apparaît à côté d'un élément (ex. une routine), cela signifie que les modifications **n'ont pas encore été sauvegardées**.

(Exemple ci-dessous : PROG2 avec R1 et R2 non sauvegardés)

 R1, la disquette à côté signifie que ça n'a pas encore été sauvegarder.



Création et utilisation d'un UDT et de programmes dans Studio 5000

1. Création d'un UDT (User-Defined Data Type)

- Dans l'arborescence du projet, ouvrir Assets puis **Data Types**.
- Faire un **clic droit** sur *User-Defined* puis sélectionner **New Data Type**.
- Définir les éléments du UDT (tags, types de données, organisation), un peu comme on le ferait en **programmation orientée objet** : cela permet de créer une structure réutilisable regroupant plusieurs variables logiquement liées.
- Nommer et enregistrer le UDT pour qu'il soit disponible dans le projet.

2. Création d'un programme et de ses routines

- Dans **Main Program**, on peut ajouter de nouveaux programmes avec leurs **variables locales** et **routines locales**.
- Pour créer un nouveau programme : **clic droit** sur *Main Program* → **Add New Program** (exemple : Prog1).
- Pour ajouter une routine à ce programme : **clic droit** sur le programme → **Add New Routine**.
- Chaque routine peut être écrite en Ladder, Structured Text (ST), ou Function Block Diagram (FBD), selon les besoins.