Types de données

8

Objet de ce chapitre

Ce chapitre décrit tous les types de données qu'il est possible d'utiliser dans une application.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

8.1 Types de données élémentaires (EDT) au format Binaire

Objet de cette section

Cette section décrit le type de données au format binaire. qui sont :

- les types Booléens,
- les types Entiers,
- le type Heure.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format binaire

Présentation

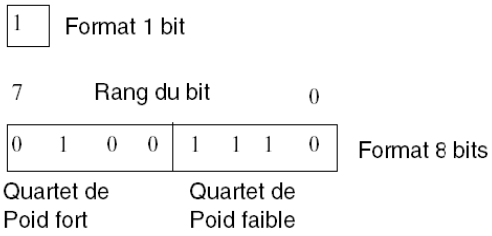
Les types de données au format Binaire appartiennent à la famille EDT (Elementary Data Type [Type de données élémentaires]), qui comprend des types de données **simples** plutôt que dérivées (tableaux, structures, blocs fonctions).

Rappel sur le format binaire

Une donnée au format binaire est constituée d'un ou de plusieurs bits ou chacun d'entre eux est représenté par un des chiffres de la base 2 soit **0** ou **1**.

L'échelle de la donnée dépend du nombre de bit(s) qui la compose.

Exemple :



Une donnée peut être:

- signée. Dans ce cas le bit de rang le plus haut est le bit de signe :
  - 0 indique une valeur positive,
  - 1 indique une valeur négative.

  La plage des valeurs est:

$$[-2^{\langle Bits-1\rangle}, 2^{\langle Bits-1\rangle}-1]$$

- non signée. Dans ce cas tous les bits représentent la valeur
  La plage des valeurs est:

$$[0, 2^{Bits}-1]$$

Bits=nombre de bits (format).

Types de données au format binaire

Liste des types de données:

Type	Désignation	Format (bits)	Valeur par défaut
BOOL	Booléen	8	0=(False)
EBOOL	Booléen avec détection de fronts et forçage	8	0=(False)
INT	Entier	16	0
DINT	Entier double	32	0
UINT	Entier non signé	16	0
UDINT	Entier double non signé	32	0
TIME	Entier double non signé	32	T=0s

Types booléens

Présentation

Il existe deux types Booléens. qui sont:

- le type BOOL, qui contient uniquement la valeur FALSE (=0) ou TRUE (=1),
- le type EBOOL, qui contient la valeur FALSE (=0) ou TRUE (=1) mais aussi des informations concernant la gestion des fronts (montants ou descendants) et le forçage.

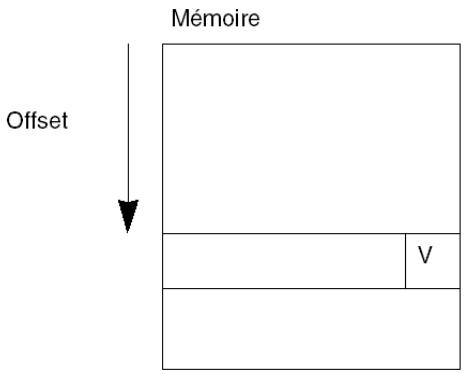
Principe du type BOOL

Ce type occupe un octet en mémoire, mais la valeur est stockée seulement dans un bit.

La valeur pas défaut de ce type est FALSE (=0).

Il est accessible par une adresse contenant l'offset sur l'octet correspondant:

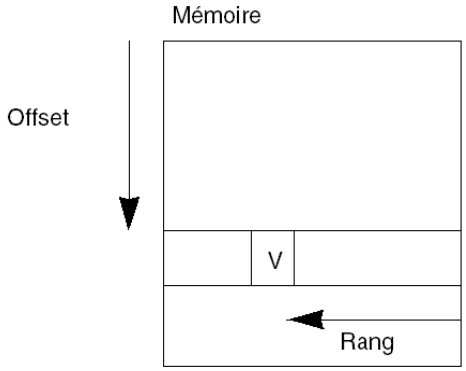
Adressage:



Dans le cas **du bit extrait de mot**, il est accessible par une adresse contenant les informations suivantes:

- un offset sur l'octet correspondant,
- le rang définissant sa position dans le mot

Adressage:



Principe du type EBOOL

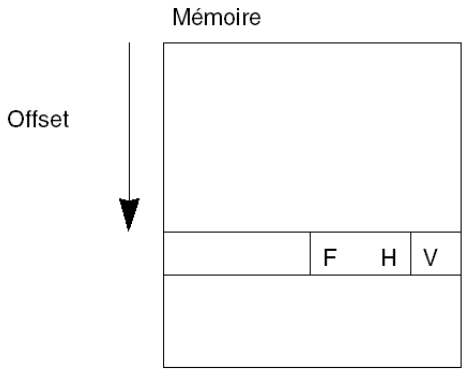
Ce type occupe un octet en mémoire dans lequel il y a :

- le bit pour de la valeur (V),
- le bit historique (H) pour la gestion des fronts (montants ou descendants). ,
- le bit contenant l'état de forçage (F). Egal à 0 si l'objet n'est pas forcé et égal à 1 si l'objet est forcé.

La valeur par défaut des bits associés au type EBOOL est FALSE (=0).

Il est accessible par une adresse spécifiant l'offset sur l'octet correspondant.

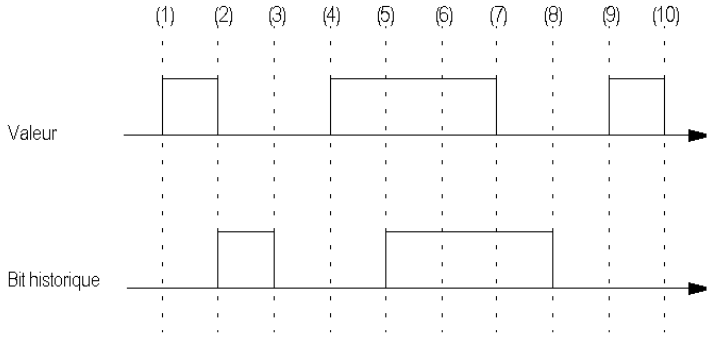
Adressage:



Chronogrammeet historique

Le chronogramme ci-dessous présente le principe des états des bits (valeur et historique) associés au type EBOLL.

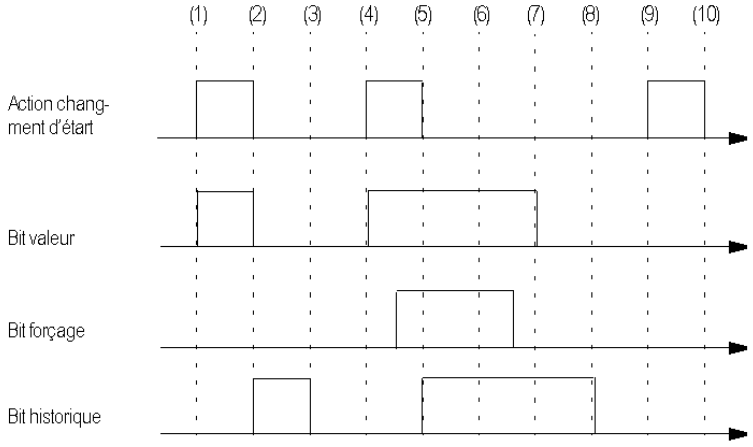
Les fronts montants du bit valeur (1, 4) sont copiées dans le bit historique au cycle automate suivant (2, 5). Les fronts descendants du bit valeur (2, 7) sont copiées dans le bit historique au cycle automate suivant (3, 8).



Chronogramme et forcage

Le chronogramme ci-dessous présente le principe des états des bits (valeur, historique, forçage) associés au type EBOLL.

Les fronts montants du bit valeur (1, 4) sont copiées dans le bit historique au cycle automate suivant (2, 5). Les fronts descendants du bit valeur (2, 7) sont copiés dans le bit historique au cycle automate suivant (3,8). Entre (4 et 5) le bit de forçage est égal à 1, les bits valeur et historique sont maintenus à 1.



Variables automate appartenant aux types booléens

Liste des variables

Variable	Type
Bit interne	EBOOL
Bit système	BOOL
Bit extrait de mot	BOOL
Entrées %I	
Bit erreur module	BOOL
Bit erreur voie	BOOL
Bit d'entrée	EBOOL
Sorties %Q	
Bit de sortie	EBOOL

Compatibilité entre BOOL et EBOOL

Les opérations autorisées entre ces deux types de variables sont:

- la copie de valeur,
- la copie d'adresse.

Copie entre types

	Destination BOOL	Destination EBOOL
Source BOOL	Oui	Oui
Source EBOOL	Oui	Oui

Compatibilité entre les paramètres des fonctions élémentaires (EF)

Paramètre effectif (externe à l'EF)	Paramètre formel BOOL (interne à l'EF)	Paramètre formel EBOOL (interne l'EF)
BOOL	Oui	Non
EBOOL	In ->Oui In-Out ->Non Out -> Oui	Oui

Compatibilité entre les paramètres des blocs fonctions (EFB\DFB)

Paramètre effectif (externe au FB)	Paramètre formel BOOL (interne au FB)	Paramètre formel EBOOL (interne au FB)
BOOL	Oui	In ->Oui In-Out ->Non Out -> Oui
EBOOL	In ->Oui In-Out ->Non Out -> Oui	Oui

Compatibilité entre variables de tableau

	Destination ARRAY[i..j) OF BOOL	Destination ARRAY[i..j) OF EBOOL
Source ARRAY[i..j) OF BOOL	Oui	Non
Source ARRAY[i..j) OF EBOOL	Non	Oui

Compatibilité entre variables statique

	Adressage direct BOOL (%MW:xi)	Adressage direct EBOOL (%Mi)
Variable declarée BOOL (Var:BOOL)	Oui	Non
Variable déclarée EBOOL (Var:EBOOL)	Non	Oui

Compatibilités

Le type de données EBOOL suit les règles suivantes :

- Une variable de type EBOOL ne peut être passée comme paramètre d'entrée/sortie de type BOOL.
- Les tableaux de EBOOL ne peuvent être passées comme paramètres de type ANY d'un FFB.
- Les tableaux de BOOL et de EBOOL ne sont pas compatibles pour l'instruction d'affectation (même règle que pour les paramètres de FFB).
- Sur Quantum :
  - Les variables localisées de type EBOOL ne peuvent être passées comme paramètres d'entrées/sorties de type EBOOL.
  - Les tableaux de EBOOL ne peuvent être passés comme paramètres d'un DFB.

Types Entier

Présentation

, qui sont :

- la base 10 (décimal) par défaut. Dans ce cas, la valeur est signée ou non signée. Cela dépend du type de l'entier.
- la base 2 (binaire). Dans ce cas, la valeur est non signée et le préfixe est **2#**.
- la base 8 (octal). Dans ce cas la valeur est non signée, le préfixe est **8#**.
- la base 16 (hexadécimal). Dans ce cas la valeur est non signée, le préfixe est **16#**.

NOTE : En représentation décimale si le type choisi est signé, la valeur peut être précédée du signe + ou du signe - (le signe + est optionnel).

Type Entier (INT)

Type signé ayant un format sur 16 bits.

Ce tableau donne la plage dans chaque base.

Base	**de...**	**à...**
Decimal	-32768	32767
Binaire	2#1000000000000000	2#0111111111111111
Octale	8#100000	8#077777
Hexadécimal	16#8000	16#7FFF

Type Entier double (DINT)

Type signé ayant un format sur 32 bits.

Ce tableau donne la plage dans chaque base.

Base	**de...**	**à...**
Decimal	-2147483648	2147483647
Binaire	2#10000000000000000000000000000000	2#01111111111111111111111111111111
Octale	8#20000000000	8#17777777777
Hexadécimale	16#80000000	16#7FFFFFFF

Type Entier non signé (UINT)

Type non signé ayant un format sur 16 bits.

Ce tableau donne la plage dans chaque base.

Base	de...	à...
Decimal	0	65535
Binaire	2#0	2#1111111111111111
Octal	8#0	8#177777
Hexadécimal	16#0	16#FFFF

Type Entier double non signé (UDINT)

Type non signé ayant un format sur 32 bits.

Ce tableau donne la plage dans chaque base.

Base	de...	à...
Decimal	0	4294967295
Binaire	2#0	2#11111111111111111111111111111111
Octal	8#0	8#37777777777
Hexadécimal	16#0	16#FFFFFFFF

Le type Heure

Présentation

Le type Heure **T#** ou **TIME#** est représenté par un type d'entier double non signé (UDINT) *(voir page 248)*.

Il exprime une durée en millisecondes, qui représente environ la durée maximale de 49 jours.

Les unités de temps autorisées pour la représentation de la valeur sont les suivantes :

- jours **(J)**
- heures **(H)**
- minutes **(M)**
- secondes **(S)**
- millisecondes **(MS)**

Saisie d'une valeur

Ce tableau montre les possibilités de saisie de la valeur maximale du type **Temps** en fonction des unités de temps autorisées.

Schéma	Commentaire
T#4294967295**MS**	valeur en millisecondes
T#4294967**S**_295**MS**	valeur en secondes\millisecondes
T#71582**M**_47**S**_295**MS**	valeur en minutes\secondes\millisecondes
T#1193**H**_2**M**_47**S**_295**MS**	valeur en heures\minutes\secondes\millisecondes
T#49**J**_17**H**_2**M**_47**S**_295**MS**	valeur en jours\heures\minutes\secondes\millisecondes

8.2 Types de données élémentaires (EDT) au format BCD

Objectif de la section

Cette section décrit les types de données au format BCD (Binary Coded Decimal) qui sont :

- le type Date,
- le type Time of Day (TOD),
- le type Date and Time (DT).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format BCD

Présentation

Les types de données au format BCD appartiennent à la famille de données élémentaires EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structures, blocs fonction).

Rappel sur le format BCD

Le format Décimal codé Binaire (Binary coded Decimal) permet de représenter les chiffres décimaux **compris entre 0 et 9** à l'aide d'un ensemble de quatre bits (quarté).

Dans ce format, les quatre bits utilisés pour coder les nombres décimaux ont une plage de combinaisons inutilisées.

Table de correspondance:

Décimal	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
	1010 (inutilisée)
	1011 (inutilisée)
	1100 (inutilisée)
	1101 (inutilisée)
	1110 (inutilisée)
	1111 (inutilisée)

Exemple de codage sur un format de 16 bits:

Valeur décimale 2450	2	4	5	0
Valeur Binaire	0010	0100	0101	0000

Exemple de codage sur un format de 32 bits:

Valeur décimale 78993016	7	8	9	9	3	0	1	6
Valeur Binaire	0111	1000	1001	1001	0011	0000	0001	0110

Types de données au format BCD

Trois types de données:

Type	Désignation	Echelle (bits)	Valeur par défaut
DATE	Date	32	D#1990-01-01
TIME_OF_DAY	Heure du jour	32	TOD#00:00:00
DATE_AND_TIME	Date et heure	64	DT#1990-01-01-00:00:00

Le type Date

Présentation

Le type **Date** codé sur un format de 32 bits contient les informations suivantes:

- l'année codée dans un champ de 16 bits (4 quartés de poids forts),
- le mois codé dans un champ de 8 bits (2 quartés),
- le jour codé dans un champ de 8 bits (2 quartés de poids faibles).

Représentation au format BCD de la date 2001-09-20:

Année (2001)	**Mois (09)**	**Jour (20)**
0010 0000 0000 0001	0000 1001	0010 0000

Régles de syntaxe

La saisie du type **Date** est la suivante : **D#**<Année>**-**<Mois>**-**<Jour>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	**Bornes**	**Commentaire**
Année	[1990,2099]	
Mois	[01,12]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)

Exemple :

Saisie	**Commentaires**
D#2001-1-1	Le 0 de gauche du mois et jour peut être omis
d#1990-02-02	Le préfixe peut être en minuscule

Le type Time of Day (TOD)

Présentation

Le type **Time of Day** codé sur un format de 32 bits contient les informations suivantes:

- l'heure codée dans un champ de 8 bits (2 quartés de poids forts),
- les minutes codées dans un champ de 8 bits (2 quartés),
- les secondes codées dans un champs de 8 bits (2 quartés).

NOTE : Les 8 bits de poids faible sont inutilisés.

Représentation au format BCD de l'heure du jour 13:25:47:

Heure (13)	Minutes (25)	Secondes (47)	Octet de poid faible
0001 0011	0010 0101	0100 0111	Inutilisés

Régles de syntaxe

Le type Time Of Day doit être saisi comme suit :
TOD#<Heure>**:**<Minutes>**:**<Secondes>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	Bornes	Commentaire
Heure	[00,23]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Minute	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Seconde	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.

Exemple :

Saisie	Commentaire
TOD#1:59:0	Les 0 de gauche des heures et secondes peuvent être omis
tod#23:10:59	Le préfixe peut être en minuscule
Tod#0:0:0	Le préfixe peut être mixte (minuscule\majuscule)

Le type Date and Time (DT)

Présentation

Le type **Date and Time** codé sur un format de 64 bits contient les informations suivantes:

- l'année codée dans un champ de 16 bits (4 quartés de poids forts),
- le mois codé dans un champ de 8 bits (2 quartés),
- le jour codé dans un champ de 8 bits (2 quartés),
- l'heure codée dans un champ de 8 bits (2 quartés),
- les minutes codées dans un champ de 8 bits (2 quartés),
- les secondes codées dans un champs de 8 bits (2 quartés).

NOTE : Les 8 bits de poids faible sont inutilisés.

Exemple : Représentation au format BCD de la date et heure 2000-09-20:13:25:47.

Année (2000)	Mois (09)	Jour (20)	Heure (13)	Minute(25)	Secondes (47)	Octet de poid faible
0010 0000 0000 0000	0000 1001	0010 0000	0001 0011	0010 0101	0100 0111	Inutilisés

Régles de syntaxe

La saisie du type **Date and Time** est la suivante:

DT#<Année>**-**<Mois>**-**<Jour>**-**<Heure>**:**<Minutes>**:**<Secondes>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	Bornes	Commentaire
Année	[1990,2099]	
Mois	[01,12]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)
Heure	[00,23]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Minute	[00,59]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Seconde	[00,59]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie

Exemple :

Saisie	Commentaire
DT#2000-1-10-0:40:0	Les 0 de gauche des mois\heure\seconde peuvent être omis
dt#1999-12-31-23:59:59	Le préfixe peut être en minuscule
Dt#1990-10-2-12:02:30	Le préfixe peut être mixte (minuscule\majuscule)

8.3 Types de données élémentaires (EDT) au format Réel

Présentation du type de données Real

Introduction

Les types de données au format Binaire appartiennent à la famille EDT (Elementary Data Type [Type de données élémentaires]), qui comprend des types de données **simples** plutôt que dérivés (tables, structures, blocs fonction).

Rappel concernant le format Real

Le format Real (virgule flottante conformément au standard ANSI/IEEE 754) est codé sur 32 bits, qui correspond aux nombres à virgule flottante à décimale unique.

Les 32 bits représentant la valeur à virgule flottante sont organisés en trois champs distincts, qui sont :

- **S**, le bit du signe qui peut prendre la valeur :
  - 0 pour un nombre à virgule flottante positif,
  - 1 pour un nombre à virgule flottante négatif.
- **e**, l'exponentiel codé dans un champ de 8 bits (nombre entier au format binaire),
- **f**, la partie à virgule fixe, codée dans un champ de 23 bits (nombre entier au format binaire).

Représentation :

La valeur de la partie à virgule fixe (Mantissa) se situe dans l'intervalle [0, 1[, et est calculée à l'aide de la formule suivante.

$F = 2^{-23} * M$

Types de nombres pouvant être représentés

Ces nombres sont les suivants :
- normalisés,
- dénormalisés,
- de valeurs infinies,
- avec des valeurs +0 et -0.

Ce tableau fournit les valeurs contenues dans les différents champs selon le type de nombre.

e	f	S	Type de nombre
]0, 255[	[0, 1[	0 ou 1	normalisé
0	[0, 1[	environ ($1,4^{E-45}$)	dénormalisé DEN
255	0	0	+ infini (INF)
255	0	1	- infini (-INF)
255	]0,1[et bit 22 = 0	0 ou 1	SNAN
255	]0,1[et bit 22 = 1	0 ou 1	QNAN
0	0	0	+0
0	0	1	-0

NOTE :

La norme CEI 559 définit deux classes de NAN (pas un nombre) : QNAN et SNAN.
- QNAN : est un NAN dont le bit 22 est défini sur 1
- SNAN : est un NAN dont le bit 22 est défini sur 0

Ils se comportent comme suit :
- Les QNAN ne déclenchent pas d'erreurs lorsqu'ils sont utilisés comme opérandes d'une fonction ou d'une expression.
- Les SNAN déclenchent une erreur lorsqu'ils sont utilisés comme opérandes d'une fonction ou d'une expression arithmétique (voir %SW17 *(voir page 175)* et %S18 *(voir page 154)*).

Ce tableau fournit la formule de calcul de la valeur du nombre à virgule flottante :

Nombre à virgule flottante	Valeur
Normalisée	$(-1)^S \times 2^{(e-127)} \times (1+f)$
Dénormalisée (DEN)	$(-1)^S \times 2^{-126} \times f$

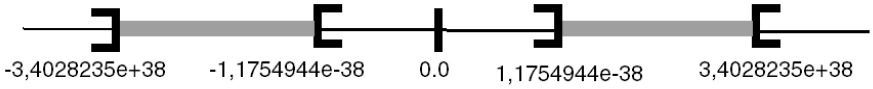
NOTE : Un nombre réel, entre -1,1754944e-38 et 1,1754944e-38, est un DEN dénormalisé. Lorsqu'un opérande est un DEN, le résultat n'est pas garanti. Les bits %SW17 *(voir page 175)* et %S18 *(voir page 154)* sont augmentés, sauf pour le Modicon M340. Les automates Modicon M340 peuvent utiliser les opérandes dénormalisés mais avec une perte de précision due à leur format. Le dépassement par valeur inférieure est signalé selon le fonctionnement uniquement lorsque le résultat est 0 (dépassement total par valeur inférieure) ou lorsque le résultat est dénormalisé (dépassement progressif par valeur inférieure avec perte de précision).

Type Real

Présentation :

Type	Echelle (bits)	Valeur par défaut
REAL	32	0.0

Plage de valeurs (parties grisées) :



Lorsqu'un résultat est :
- compris entre -1,1754944e-38 et 1,1754944e-38, le nombre est un DEN ;
- inférieur à -3,4028234e+38, le symbole -INF (pour -infini) s'affiche ;
- supérieur à +3,4028234e+38, le symbole INF (pour + infini) s'affiche ;
- indéfini (racine carrée d'un nombre négatif), le symbole NAN s'affiche.

Exemples d'imprécision de la valeur normalisée

7,986 est codé par l'application comme suit :

S	E=129	M=8359248
0	1000001	11111111000110101010000

A l'aide de la formule :

$$(-1)^0 \times 2^{(129-127)} \times \left(1 + \frac{8359248}{2^{23}}\right) = (7{,}986000061035145625)$$

Le nombre 7,986 doit avoir une valeur significative de :

$$\left(\frac{7{,}986}{2^2} - 1\right) \times 2^{23} = (8359247{,}872)$$

La valeur significative étant exprimée sous la forme d'un entier, elle ne peut être codée que sous la forme 8359248 (arrondie à la limite la plus proche).

Aucun nombre ne peut être codé entre les signifiants 8359247 et 8359248, ou entre les nombres réels 7,985999584197998046875 et 7,98600006103515625.

L'importance du bit de poids faible (écart) est, en précision absolue :

$$\frac{2^{(129-127)}}{2^{23}} = 2^{-21} = 0{,}000000476837158203125$$

L'écart devient très important pour les valeurs élevées, comme indiqué ci-dessous :

Valeur	$Range \le 2^{n-1} \le Value \le 2^n$	M=8359248
100 000 000	Entre 2^{26} et 2^{27}	$\frac{2^{26}}{2^{23}} = 2^3 = 8$
2^{127}	2^{127}	$\frac{2^{127}}{2^{23}} = 2^{104} = 2{,}02 \times 10^{31}$

NOTE : L'écart correspond à l'importance du bit de poids faible.

Pour obtenir la résolution souhaitée, il est nécessaire de définir la plage maximale de calcul, à l'aide de la formule suivante :

$$e = \frac{\left|Ln(p \times 2^{23})\right|}{Ln(2)}$$

p étant la précision et e l'exposant (e = E-127)

Par exemple, si la précision requise est de 0,001, la partie à virgule fixe est :

$$F = (-1)^S \times 2^{(e)} \times \left(1 + \frac{M}{2^{23}}\right) = 2^{14} = 16384$$

avec :

$$13 = \frac{\left|Ln(0,001 \times 2^{23})\right|}{Ln(2)}$$

Au-delà de cette limite F, la précision est perdue.

Cas typique : compteurs

La virgule flottante doit être utilisée avec précaution, notamment lorsqu'il s'agit d'ajouter un petit nombre à lui-même.

Si les incréments sont petits, le compteur ne fonctionnera pas correctement. Il donne des résultats erronés et interrompt l'incrémentation lorsque la valeur à ajouter est inférieure à celle du bit de poids faible du compteur.

Pour obtenir les valeurs correctes, il est recommander d'utiliser un double entier (`UDINT`) et de multiplier le résultat par l'incrément.

Exemple :

- Incrémentez une valeur de 0,001 entre 33000 et 1000000.
- Comptez de 33 000 000 à 1 000 000 000 (valeur x 1000) avec 1 comme incrément.
- Obtenez le résultat en multipliant la valeur par 0,001.

La précision **F** minimale par plage est :

De...à...	F (minimum)
3300...65536	0.004
65536...131072	0.008
...	...
524288...1000000	0.063

Ce compteur peut fonctionner jusqu'à 4 294 967 295 x 0,001 = 4 294 967,5 avec une précision minimale de 0,5.

NOTE : La valeur réelle ici est la valeur binaire encodée. Elle peut différer de celle affichée dans un écran d'exploitation, si elle est arrondie (4,294968e+006).

8.4 Types de données élémentaires (EDT) au format chaine de caractères

Présentation des types de données au format chaîne de caractères

Introduction

Le type de données au format chaîne de caractères appartient à la famille de données élémentaires EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structures, blocs fonction).

Le type chaîne de caractères

Le format chaîne de caractères permet de représenter une chaîne de caractères ASCII, chaque caractère étant codé sur un format de 8 bits.

Les caractéristiques du type chaîne de caractères sont les suivantes :

- 16 caractères par défaut dans la chaîne (caractère fin de chaîne exclu),
- une chaîne est composée de caractères ASCII compris entre 16#20 et 16#FF (représentation hexadécimale).
- dans une chaîne vide, le caractère de fin de chaîne (code ASCII « NUL ») est le premier de la chaîne.
- la taille maximale d'une chaîne est de 65 535 caractères.

La taille de la chaîne de caractères peut être optimisée lors de la définition du type par la commande **STRING[<taille>]**, <taille> étant un entier non signé UINT pouvant définir une chaîne de 1 à 65 535 caractères ASCII.

NOTE : les caractères ASCII 0 à 127 sont communs à toutes les langues, mais les caractères 128 à 255 varient selon les langues. Faites attention si la langue de Unity Pro n'est pas la même que celle du SE. Si ces langues diffèrent, la communication CHAR MODE peut être perturbée et l'envoi de caractères supérieurs à 127 risque d'être incorrect. En particulier, si le caractère « Arrêt en réception » est supérieur à 127, il n'est pas pris en compte.

Règles de syntaxe

La saisie est précédée et terminée par le caractère apostrophe « ’ » (code ASCII 16#27).

Le signe $ (dollar) est un caractère spécial. Suivi de certaines lettres, il indique :
- $L ou $l, aller à la ligne suivante (line feed),
- $N ou $n, aller au début de la ligne suivante (new line),
- $P ou $p, aller à la page suivante (go to next page),
- $R ou $r, retour chariot (carriage return),
- $T ou $t, tabulation (Tab),
- $$, représente le caractère $ dans une chaîne,
- $’, représente le caractère apostrophe dans une chaîne.

L'utilisateur peut utiliser la syntaxe $nn pour afficher, dans une variable STRING, les caractères à ne pas imprimer. Il peut par exemple s'agir d'un retour chariot (code ASCII 16#0D).

Exemples

Exemples de saisies :

Type	**Entrée**	**Contenu de la chaîne** **• représente le caractère de fin de chaîne** *** représente les octets vides**
STRING	’ABCD’	ABCD•********** (16 caractères)
STRING[4]	jean•	jean•
STRING[10]	’It$’s jean’	It’s jean•*
STRING[5]	’’	•*****
STRING[5]	’$’’	’•****
STRING[5]	’le nombre’	le no•
STRING[13]	’0123456789’	0123456789•***
STRING[5]	’RL’	<cr><lf>•***
STRING[5]	’$$1.00’	$1.00•

Déclaration d’une variable de type STRING

Une variable de type STRING peut être déclarée de deux manières différentes :
- STRING et
- STRING[<Nombre d'éléments>]

Selon l'usage, le comportement varie :

Type	Déclaration d'une variable	Paramètre d'entrée d'un FFB	Paramètre de sortie d'un EF	Paramètre de sortie d'un FB
STRING	Taille fixe : 16 caractères	La taille est égale à la taille réelle du paramètre d'entrée.	La taille est égale à la taille réelle du paramètre d'entrée.	Taille fixe de 16 caractères
STRING[<n>]	Taille fixe : n caractères	La taille est égale à la taille réelle du paramètre d'entrée limitée à n caractères.	L'EF écrit un maximum de n caractères.	Le FB écrit un maximum de n caractères.

Les chaînes et la broche ANY

Lorsque vous utilisez une variable de type STRING comme paramètre de type ANY il est fortement recommandé de vérifier que la taille de la variable est inférieure à la taille maximum déclarée.

Exemple :

```
Utilisation de STRING sur la fonction SEL (Sélecteur).
String1 : STRING[8]
String2 : STRING[4]
String3 : STRING[4]

String1:= 'AAAAAAAA';
String3:= 'CC';
Cas 1 :
String2:= 'BBBB';
(* la taille de la chaîne est égale à la taille maximum
déclarée *)
String1:= SEL(FALSE, String2, String3);
(* le résultat sera : 'BBBBAAAA' *)
Cas 2 :
String2:= 'BBB';
(* la taille de la chaîne est inférieure à la taille maximum
déclarée*)
String1:= SEL(FALSE, String2, String3);
(* le résultat sera : 'BBB' *)
```

8.5 Types de données élémentaires (EDT) au format chaîne de bits

Objet de cette section

Cette section décrit le type de données au format chaîne de bits. qui sont :

- le type Byte,
- le type Word,
- le type Dword.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format chaîne de bits

Présentation

Les types de données au format chaine de bits appartiennent à la famille de données élémentaire EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structure, bloc fonctions).

Rappel sur le format chaîne de bits

La particularité de ce format est que l'ensemble des bits qui le compose ne représente pas une valeur numérique, mais une combinaison de bits séparés.

Les données appartenant aux types de ce format peuvent être représentées sous trois bases qui sont :

- l'hexadécimal (16#),
- l'octal (8#),
- le binaire (2#).

Types de données au format chaîne de bits

Trois types de données:

Type	Echelle (bits)	Valeur par défaut
BYTE	8	0
WORD	16	0
DWORD	32	0

Types de chaînes de bits

Le type Byte

Le type Byte est codé sur un format de 8 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FF
Octal	8#0	8#377
Binaire	2#0	2#11111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
00001000	16#8
00110011	8#63
00110011	2#110011

Le type Word

Le type Word est codé sur un format de 16 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FFFF
Octal	8#0	8#177777
Binaire	2#0	2#1111111111111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

Le type Dword

Le type Dword est codé sur un format de 32 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FFFFFFFF
Octal	8#0	8#37777777777
Binaire	2#0	2#11111111111111111111111111111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

8.6 Types de données dérivés (DDT/IODDT)

Objet de cette section

Cette section décrit les types de données dérivés qui sont :

- les tableaux (DDT),
- d'EDT
  - les structures concernant les données d'entrées\sorties (IODDT),
  - les structures concernant les autres données (DDT).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Tableaux

Qu'est-ce qu'un tableau ?

Un tableau est un élément de données incluant un **ensemble** de données **de type identique**, tel que :

- données élémentaires (EDT),
  par exemple :
  - un groupe de mots BOOL,
  - un groupe de mots entiers UINT,
  - etc.
- données dérivées (DDT),
  par exemple :
  - un groupe de tables WORD,
  - un groupe de structures,
  - etc.

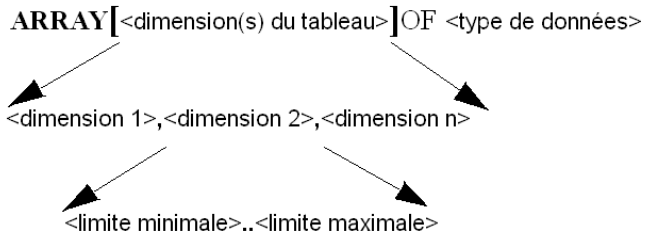
Caractéristiques

Un tableau est caractérisé par deux paramètres :

- un paramètre définissant sa présentation (dimension(s) du tableau),
- un paramètre définissant le type de données qu'il contient.

NOTE : la présentation la plus complexe est le tableau à **six dimensions**.

La syntaxe incluant ces deux paramètres se présente comme suit :



<limite minimale> **inférieure** à <limite maximale>

Définition et instanciation d'un tableau

Définition d'un type de tableau :

```
X : ARRAY[1.0,10] OF BOOL
```

Instanciation d'un tableau :

```
Tab_1: X
Tab_2: ARRAY[1.0,10] OF BOOL
```

Les instances Tab_1 et Tab_2 sont de même type et ont le même nombre de dimensions ; la seule différence concerne ce qui se passe lors de l'instanciation :

- Le type Tab_1 est nommé X.
- Le type Tab_2 doit être défini (la table correspondante ne porte pas de nom).

NOTE : il est préférable d'attribuer un nom au type, puisque chaque modification requise ne sera effectuée qu'une seule fois (en effet, dans le cas contraire, le nombre de modifications serait aussi important que le nombre d'instances).

Exemples

Le tableau ci-dessous répertorie des instances de tableaux dont le nombre de dimensions diffère :

Entrée	Commentaires
Tab_1: ARRAY[1..2] OF BOOL	Tableau à 1 dimension contenant 2 mots de type booléen
Tab_2: ARRAY[-10..20] OF WORD	Tableau à 1 dimension contenant 31 structures de type WORD (structure définie par l'utilisateur)
Tab_3: ARRAY[1..10, 1..20] OF INT	Tableaux à 2 dimensions contenant 10 x 20 nombres entiers
Tab_4: ARRAY[0..2, -1..1, 201..300, 0..1] OF REAL	Tableaux à 4 dimensions contenant 3 x 3 x 100 x 2 nombres réels

NOTE : un grand nombre de fonctions (READ_VAR, WRITE_VAR, par exemple) ne reconnaissent pas l'index d'un tableau de mots commençant par un nombre autre que 0. Si vous avez recours à ce type d'index, les fonctions examinent le nombre de mots du tableau, mais pas celui figurant au niveau du premier ensemble de l'index dans la définition du tableau.

⚠ AVERTISSEMENT
COMPORTEMENT INATTENDU DE L'APPLICATION - INDEX DE TABLEAU NON VALIDE Lors de l'application de fonctions à des variables de type tableau, vérifiez que les fonctions sont compatibles avec la valeur de l'index de début des tableaux lorsque cette valeur est supérieure à 0. **Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.**

Accès à un élément de données dans le tableau Tab_1 ou Tab_3 :

```
Tab_1[2]
;Pour accéder au deuxième élément

Tab_3[4][18]
;Pour accéder au dix-huitième élément du quatrième sous-tableau
```

Règles d'affectation entre les tableaux

Les 4 tableaux suivants sont disponibles :

```
Tab_1:ARRAY[1..10] OF INT
Tab_2:ARRAY[1..10] OF INT
Tab_3:ARRAY[1..11] OF INT
Tab_4:ARRAY[101..110] OF INT


Tab_1: = Tab_2; affectation autorisée
Tab_1: = Tab_3; affectation refusée (non conforme CEI)
Tab_1: = Tab_4; affectation refusée (non conforme CEI)
```

Structures

Qu'est ce qu'une structure?

C'est une donnée qui contient un **ensemble** de données **de type différent** telles que:

- un ensemble de BOOL, WORD, UINT, etc., (structure EDT),
- un ensemble de tableaux (structure de DDT),
- un ensemble de REAL, DWORD, tableaux, etc..., (structure d'EDT et DDT).

NOTE : vous pouvez réaliser des structures imbriquées (DDT imbriqués) sur 8 niveaux. **Les structures (DDT) récursives ne sont pas autorisées.**

Caractéristiques

Une structure est constituée de données qui sont chacunes caractérisées par:

- un type,
- un nom, qui permet de l'identifier,
- un commentaire (optionnel) décrivant son rôle.

Définition d'un type de structure :

```
IDENT
    Nom : STRING[12]
    Prenom : STRING[16]
    Age : UINT

;La structure de type IDENT contient une donnée de type UINT et
deux données de types STRING
```

Définition de deux instances de donnée de la structure de type IDENT :

```
Personne_1 : IDENT
Personne_2 : IDENT

;Les instances Personne_1 et Personne_2 sont du type Structure
IDENT
```

Accés à une données d'une structure

Accès à des données de l'instance Personne_1 de type IDENT :

```
Personne_1.Nom  ;Permet d'acceder au nom de Personne_1

Personne_1.Age    ;Permet d'acceder à l'age de Personne_1
```

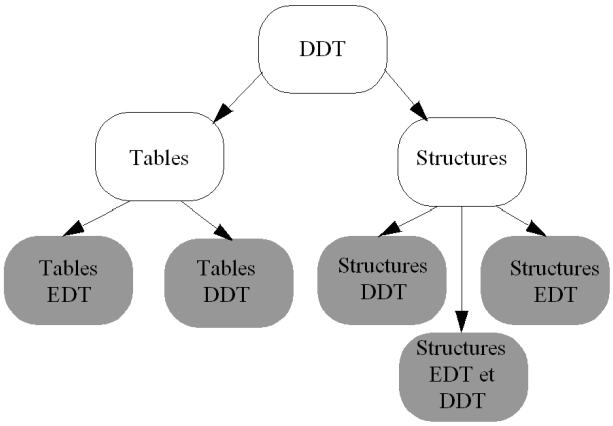
Vue d'ensemble de la famille de type de données dérivées (DDT)

Introduction

La famille DDT (Derived Data Type) inclut les types de données dit "**dérivés**" tels que :

- les tables ;
- les structures.

Illustration :



Caractéristiques

Un élément de données appartenant à la famille DDT comporte :

- le nom du type *(voir page 238)* (32 caractères maximum) défini par l'utilisateur (facultatif pour les tables, mais recommandé) *(voir page 272)* ;
- le type (structure ou table) ;
- un commentaire facultatif de 1 024 caractères maximum (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- la description (dans le cas d'une structure) de ces éléments :
  - le nom de l'élément *(voir page 238)* (32 caractères maximum) ;
  - le type d'élément ;
  - un commentaire facultatif de 1 024 caractères maximum expliquant son rôle (les caractères autorisés correspondent aux codes ASCII 32 à 255).
- des informations :
  - le numéro de version du type ;
  - la date de la dernière modification du code, des variables internes ou des variables de l'interface ;
  - un fichier de 32 767 caractères décrivant la fonction bloc et ses différentes modifications (facultatif).

NOTE : La taille totale d'une table ou structure n'excède pas 64 Ko.

Exemples

Définition des types

```
COORD
    X : INT
    Y : INT
 ; Structure de type COORD

SEGMENT
    Origin: COORD
    Destination: COORD
 ; Structure de type SEGMENT contenant 2 structures de
type COORD

OUTLINE: ARRAY[0..99] OF SEGMENT
 ; Table de type OUTLINE contenant 100 structures de
type SEGMENT

DRAW
    Color : INT
    Anchor : COORD
    Pattern : ARRAY[0..15,0..15] OF WORD
    Contour : OUTLINE
 ; Structure de type DRAW
```

Accès aux données d'une instance de structure de type DRAW

```
Cartoon: DRAW
 ; Instance de structure de type DRAW

Cartoon.Pattern[15,15]
 ; Accès au dernier élément de données dans la table
Pattern de la structure Cartoon

Cartoon.Contour[0].Origin.X
 ; Accès à l'élément de données X de la structure COORD
appartenant à la première structure SEGMENT de la table
Contour.
```

DDT : règles d'affectation

Vue d'ensemble

Les DDT sont stockés dans la mémoire de l'automate selon l'ordre dans lequel leurs éléments sont déclarés.

Il existe cependant certaines règles que voici.

Principe pour Premium et Quantum

Le principe de stockage pour Premium et Quantum est le suivant :

- Les éléments sont stockés dans l'ordre selon lequel ils sont déclarés dans la structure.
- L'élément de base est l'octet (alignement des données sur les octets mémoire).
- Chaque élément possède une règle d'alignement :
  - Les types `BOOL` et `BYTE` sont indifféremment alignés sur les octets pairs ou impairs.
  - Tous les autres types élémentaires sont alignés sur les octets pairs.
  - Les structures et tableaux sont alignés selon la règle d'alignement des types `BOOL` et `BYTE` s'ils ne contiennent que des éléments de type `BOOL` et `BYTE` ; sinon, ils sont alignés sur les octets pairs de la mémoire.

⚠ AVERTISSEMENT

RISQUE D'INCOMPATIBILITE APRES LA CONVERSION DE CONCEPT

Avec l'application de programmation **Concept**, les structures de données ne traitent aucun décalage dans les offsets (chaque élément est défini l'un après l'autre dans la mémoire, quel que soit son type). Par conséquent, nous vous recommandons de tout vérifier, en particulier la cohérence des données lors de l'utilisation des `DDT` situés dans la « RAM d'état » (risque de décalages) ou des fonctions pour la communication avec d'autres équipements (transferts avec une taille différente de celle programmée dans Concept).

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

Principe pour Modicon M340

Le principe de stockage pour les automates Modicon M340 est le suivant :

- Les éléments sont stockés dans l'ordre selon lequel ils sont déclarés dans la structure.
- L'élément de base est l'octet.
- Une règle d'alignement et une fonction de l'élément :
  - Les types `BOOL` et `BYTE` sont alignés sur les octets pairs ou impairs.
  - Les types `INT`, `WORD` et `UINT` sont alignés sur les octets pairs.
  - Les types `DINT`, `UDINT`, `REAL`, `TIME`, `DATE`, `TOD`, `DT` et `DWORD` sont alignés sur les mots doubles.
  - Les structures et les tables sont alignées selon les règles de leurs éléments.

⚠ AVERTISSEMENT

MAUVAIS ECHANGES ENTRE UN MODICON M340 ET UN PREMIUM OU UN QUANTUM.

Vérifiez si la structure des données échangées présente les mêmes alignements dans les deux projets.

Si tel n'est pas le cas, les données ne seront pas correctement échangées.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

NOTE : il est possible que l'alignement des données ne soit pas identique si le projet est transféré du simulateur de Unity Pro vers un automate M340. Vérifiez par conséquent la structure des données du projet.

NOTE : Unity Pro indique où l'alignement semble différent. Vérifiez les instances correspondantes dans l'éditeur de données. Voir la page Options du projet pour savoir comment activer cette option.

Exemples

Le tableau ci-dessous donne quelques exemples de structures de données. Dans ces exemples, les DDT de type structure sont adressés à `%MWi`. Le 1er octet du mot correspond aux 8 bits de poids faible et le 2e octet du mot correspond aux 8 bits de poids fort.

Pour toutes les structures suivantes, la première variable est affectée à l'adresse `%MW100` :

Première adresse mémoire		**Description de la structure**	
Modicon M340	Premium	**Para_PWM1**	
`%MW100` (1er octet)	`%MW100` (1er octet)		t_period : TIME

Première adresse mémoire		Description de la structure	
%MW102 (1er octet)	%MW102 (1er octet)		t_min : TIME
%MW104 (1er octet)	%MW104 (1er octet)		in_max : REAL
		Mode_TOTALIZER	
%MW100 (1er octet)	%MW100 (1er octet)		hold : BOOL
%MW100 (2e octet)	%MW100 (2e octet)		rst : BOOL
		Info_TOTALIZER	
%MW100 (1er octet)	%MW100 (1er octet)		outc : REAL
%MW102 (1er octet)	%MW102 (1er octet)		cter : UINT
%MW103 (1er octet)	%MW103 (1er octet)		done : BOOL
%MW103 (2e octet)	%MW103 (2e octet)		**Réservé pour l'alignement**

Le tableau ci-dessous donne deux exemples de structures de données avec tableaux :

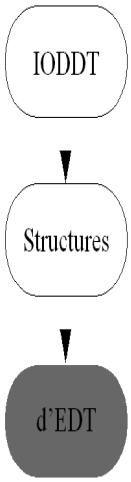
Première adresse mémoire		Description de la structure	
Modicon M340	Premium	**EHC105_Out**	
%MW100 (1er octet)	%MW100 (1er octet)		Quit : BYTE
%MW100 (2e octet)	%MW100 (2e octet)		Control : ARRAY [1.0,5] OF BYTE
%MW104 (1er octet)	%MW103 (1er octet)		Final : ARRAY [1..5] OF DINT
		CPCfg_ex	
%MW100 (1er octet)	%MW100 (1er octet)		Profile_type : INT
%MW101 (1er octet)	%MW101 (1er octet)		Interp_type : INT
%MW102 (1er octet)	%MW102 (1er octet)		Nb_of_coords : INT
%MW103 (1er octet)	%MW103 (1er octet)		Nb_of_points : INT
%MW104 (1er octet)	%MW104 (1er octet)		reserved : ARRAY [0..4] OF BYTE
%MW106 (2e octet)	%MW106 (2e octet)		**Réservé pour l'alignement de la variable Master_offset sur octets pairs**
%MW108 (1er octet)	%MW107 (1er octet)		Master_offset : DINT
%MW110 (1er octet)	%MW109 (1er octet)		Follower_offset : INT
%MW111 (mot entier)	-		**Réservé pour l'alignement**

Aperçu des types de données dérivés d'entrée/de sortie (IODDT)

Présentation

Les types de données dérivés d'entrées\sorties IODDT (Input Output Derived Data Type) **sont prédéfinis par le constructeur**, ils contiennent des objets langage de la famille EDT appartenant à la voie d'un module métier.

Illustration :



Les type IODDT sont des structures dont la taille (nombre d'éléments qui les composent) dépend de la voie ou du module d'entrées\sorties qu'elles représentent.

Un module d'entrées\de sorties donné peut avoir plus d'un IODDT.

La différence avec une structure classique est que:

- la structure IODDT est prédéfinie par le constructeur,
- les éléments composant la structure IODDT n'ont pas une allocation mémoire contigüe, ils ont une adresse spécifique dans le module.

Exemples

Structure IODDT pour une voie d'entrée\de sortie d'un module analogique

```
ANA_IN_GEN    ;Structure de type ANA_IN_GEN
   Value:INT ;Valeur de l'entrée
   Err:  BOOL ;Erreur voie
```

Accès à des données d'une instance de type ANA_IN_GEN :

```
 Cistern_Level: ANA_IN_GEN
;Instance de type ANA_IN_GEN qui correspond par exemple
à un capteur de niveau de cuve

Cistern_Level.Value ;Lecture valeur d'entrée de la voie
Cistern_Level.Err ;Lecture bit d'erreur de la voie
```

Accès par adressage direct :

```
S'il s'agit de la voie 0 du module 2 du rack 0 nous avons :

Cistern_Level       correspond à %CH0.2.0
Cistern_Level.Value correspond à %IW0.2.0.0
Cistern_Level_Err   correspond à %I0.2.0.ERR
```

8.7 Types de données blocs fonctions (DFB\EFB)

Objet de cette section

Cette section décrit les types de données de bloc fonction. qui sont :

- blocs fonction utilisateur (DFB) ;
- blocs fonction élémentaires (EFB).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

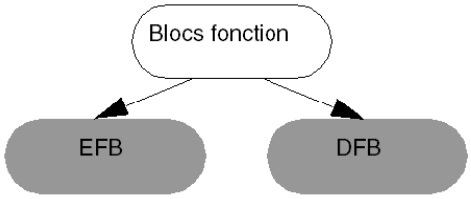
Vue d'ensemble des familles de type de données de bloc fonction

Présentation

Les familles de type de données blocs fonction sont:

- la famille de type Blocs fonction élémentaires (EFB) *(voir page 233)*,
- la famille de type Blocs fonction utilisateur (DFB) *(voir page 233)*.

Illustration :

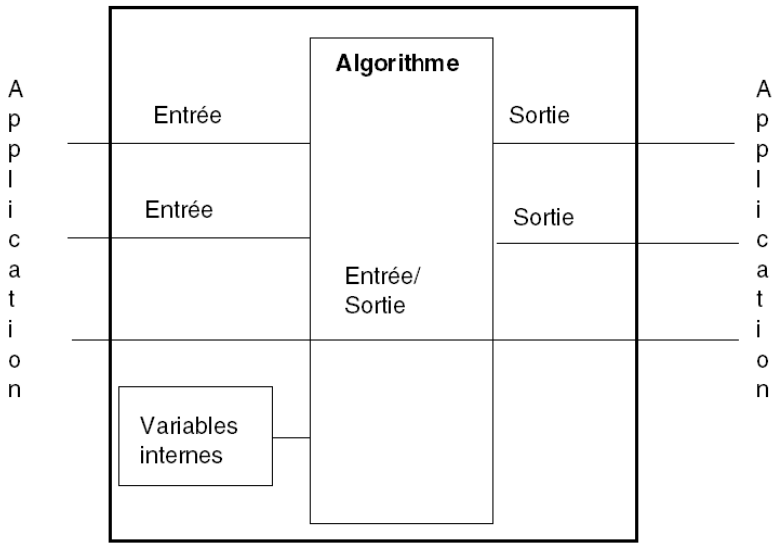


Les blocs fonctions sont des entités contenant:

- des variables d'entrées et de sorties servant d'interface avec l'application,
- un algorithme de traitement exploitant les variables d'entrées et renseignant les variables de sorties,
- des variables internes privées et publiques exploitées par l'algorithme de traitement.

Illustration

Bloc fonction :



Bloc fonction utilisateur (DFB)

Les types blocs fonction utilisateur (blocs fonction dérivés) sont développés par l'utilisateur avec un ou plusieurs langages (suivant le nombre de sections). Ces langages sont les suivants :

- à contacts,
- langage littéral structuré,
- langage liste d'instructions,
- langage en blocs fonctionnels FBD.

Un type DFB peut avoir une ou plusieurs instances, chaque instance est référencée par un nom (symbole) et possède les données du type de DFB.

les blocs fonction élémentaires (EFB)

Les types blocs fonctions élémentaires (EFB) sont fournis par le constructeur, ils sont programmés en langage C.

L'utilisateur peut créer ses EFB, pour cela il doit disposer d'un outil logiciel optionnel **"SDKC"**

Un type EFB peut avoir une ou plusieurs instances, chaque instance est référencée par un nom (symbole) et possède les données du type de l'EFB.

Caractéristiques des types de données de bloc fonction (EFB\DFB)

Définition du type

Le type d'un bloc fonction EFB ou DFB est défini par :

- le nom du type *(voir page 238)*, défini par l'utilisateur pour les DFB,
- un commentaire facultatif. (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- les données d'interface avec l'application:
  - les entrées, pas accessibles en lecture\écriture depuis l'application, mais lues par le code du bloc fonction,
  - les entrées\sorties, pas accessibles en lecture\écriture depuis l'application, mais lues et écrites par le code du bloc fonction,
  - les sorties, accessibles en lecture depuis l'application et lues et écrites par le code du bloc fonction.
- les donnéees internes:
  - publiques, accessibles en lecture\écriture depuis l'application, lues et ecrites par le code du bloc fonction,
  - privées, pas accessibles depuis l'application lues et écrites par le code du bloc fonction.
- le code:
  - pour les DFB il est écrit par l'utilisateur en langage d'automatisme (littéral structuré, liste d'instructions, langage à contacts, langage à blocs fonctionnels), il est structuré en une seule section si l'option IEC est active, ou peut être structuré en plusieurs sections si cette option est inactive
  - pour les EFB il est écrit en langage C.
- des informations telles que:
  - le numéro de version du type,
  - la date de la derniére modification du code, ou des variables internes, ou des variables interfaces.
  - une fiche descriptive facultative (32767 caractères), décrivant la fonction du bloc et ses différentes modifications.

Caractéristiques

Ce tableau donne les caractéristiques des éléments composant un type:

Elément	EFB	DFB
Nom	32 caractères	32 caractères
Commentaire	1024 caractères	1024 caractères
Données d'Entrées	32 maximum	32 maximum
Données d'Entrées/Sorties	32 maximum	32 maximum
Données de sortie	32 maximum	32 maximum
Nombre d'interfaces (Entrées+Sorties+Entrées/Sorties)	32 maximum **(2)**	32 maximum **(2)**
Données publiques	Pas de limites **(1)**	Pas de limites **(1)**
Données privées	Pas de limites **(1)**	Pas de limites **(1)**
Langage de programmation	Langage C	Langage: ● littéral structuré, ● liste d'instructions, ● à contacts, ● à blocs fonctionnels.

Section		Une section est définie par:
		● un nom (32 caractères maximum),
		● une condition de validation,
		● un commentaire (256 caractéres maximum),
		● une protection:
		  ● sans,
		  ● lecture,
		  ● lecture\écriture.
		Une section ne peut pas accéder aux variables déclarées dans l'application sauf les:
		● doubles mots systéme %SDi,
		● mots système %SWi,
		● bits système %Si.

(1): la seule limitation est la taille mémoire de l'automate.

(2): ne sont pas prises en compte l'entrée EN et la sortie ENO.

Caractéristiques d'éléments appartenant aux blocs fonction

Qu'est-ce qu'un élément ?

Chaque élément (données d'interface ou données internes) est défini par :

- un nom *(voir page 238)* (comportant 32 caractères maximum), attribué par l'utilisateur ;
- un type,
  qui fait partie de l'une des familles ci-après :
  - données élémentaires (EDT) ;
  - données dérivées (DDT) ;
  - données de blocs fonction (EFB/DFB).
- un commentaire facultatif de 1 024 caractères maximum (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- une valeur initiale ;
- un droit d'accès issu du programme d'application (pour afficher des sections de l'application ou la section appartenant aux DFB, reportez-vous à la rubrique Définition du type de bloc fonction (variables d'interface et internes) *(voir page 285)*) ;
- un droit d'accès provenant des requêtes de communication ;
- un indicateur de sauvegarde de variables publiques.

Types de données autorisés pour un élément appartenant à un DFB

Les types de données autorisés sont indiqués dans le tableau ci-dessous :

Elément du DFB	**Types EDT**	**Types DDT**				**ANY...**	**Types de bloc fonction**
		IODDT	Tables sans nom	ANY_ARRAY	autre		
Données d'entrée	Oui	Non	Oui	Oui	Oui	Oui**(2)**	Non
Données d'entrée/de sortie	Oui**(1)**	Oui	Oui	Oui	Oui	Oui**(2)**	Non
Données de sortie	Oui	Non	Oui	Non	Oui	Oui **(2) (3)**	Non
Données publiques	Oui	Non	Oui	Non	Oui	Non	Non
Données privées	Oui	Non	Oui	Non	Oui	Non	Oui

(1) : non autorisé pour les données statiques de type EBOOL utilisées sur les automates Quantum

(2) : non autorisé pour les données de type BOOL et EBOOL

(3) : achèvement lors de l'exécution du DFB et utilisation impossible en dehors du DFB

Types de données autorisés pour un élément appartenant à un EFB

Les types de données autorisés sont indiqués dans le tableau ci-dessous :

Elément de l'EFB	**Types EDT**	**Types DDT**				**ANY...**	**Types de bloc fonction**
		IODDT	Tables sans nom	ANY_ARRAY	autre		
Données d'entrée	Oui	Non	Non	Oui	Oui	Oui**(1)**	Non
Données d'entrée/de sortie	Oui	Oui	Non	Oui	Oui	Oui**(1)**	Non
Données de sortie	Oui	Non	Non	Non	Oui	Oui **(1) (2)**	Non
Données publiques	Oui	Non	Non	Non	Oui	Non	Non
Données privées	Oui	Non	Non	Non	Oui	Non	Oui

(1) : non autorisé pour les données de type BOOL et EBOOL

(2) : achèvement lors de l'exécution de l'EFB et utilisation impossible en dehors de l'EFB

Valeurs initiales pour un élément appartenant à un DFB

Le tableau suivant précise si les valeurs initiales entrées sont issues de la définition du type DFB ou de l'instance DFB :

Elément du DFB	**Provenant du type DFB**	**Provenant de l'instance EFB**
Données d'entrée (pas de type ANY...)	Oui	Oui
Données d'entrée (de type ANY...)	Non	Non
Données d'entrée/de sortie	Non	Non
Données de sortie (pas de type ANY...)	Oui	Oui
Données de sortie (de type ANY...)	Non	Non
Données publiques	Oui	Oui
Données privées	Oui	Non

Valeurs initiales pour un élément appartenant à un EFB

Le tableau suivant précise si les valeurs initiales entrées sont issues de la définition du type EFB ou de l'instance EFB :

Elément de l'EFB	Provenant du type EFB	Provenant de l'instance EFB
Données d'entrée (pas de type ANY..., voir generic data types *(voir page 290)*)	Oui	Oui
Données d'entrée (de type ANY...)	Non	Non
Données d'entrée/de sortie	Non	Non
Données de sortie (pas de type ANY...)	Oui	Oui
Données de sortie (de type ANY...)	Non	Non
Données publiques	Oui	Oui
Données privées	Oui	Non

⚠ AVERTISSEMENT

COMPORTEMENT INATTENDU DE L'APPLICATION - INDEX DE TABLEAU NON VALIDE

Lorsque des EFB et des DFB sont utilisés sur des variables de type tableau, faites uniquement appel à des tableaux avec un index de début égal à 0.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

8.8 Types de données génériques (GDT)

Vue d'ensemble des types de données génériques

Présentation

Les types de données génériques sont des ensembles de types classiques (EDT, DDT) ayant pour vocation de déterminer la compatibilité entre ces types classiques.

Ces ensembles sont identifiés par le préfixe ANY_ARRAY, mais ces préfixes ne peuvent en aucun cas être utilisés pour instancier des données.

Leurs champs d'utilisation concernent les familles de type de données blocs fonctions (EFB\DFB) et les fonctions élémentaires (EF), afin de définir les types de données compatibles avec leur interface :

- entrées
- entrées/sorties
- sorties

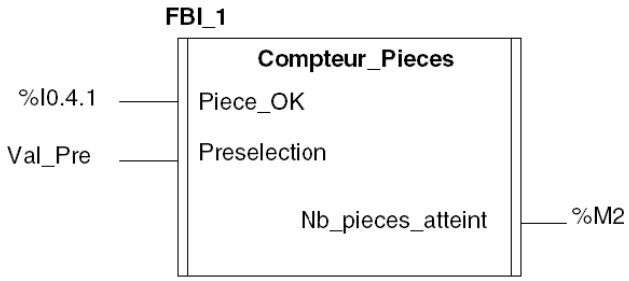
Types de données génériques (GDT) disponibles

Les types de données génériques disponibles dans Unity Pro sont les suivants :

- ANY_ARRAY_WORD
- ANY_ARRAY_UINT
- ANY_ARRAY_UDINT
- ANY_ARRAY_TOD
- ANY_ARRAY_TIME
- ANY_ARRAY_STRING
- ANY_ARRAY_REAL
- ANY_ARRAY_INT
- ANY_ARRAY_EBOOL
- ANY_ARRAY_DWORD
- ANY_ARRAY_DT
- ANY_ARRAY_DINT
- ANY_ARRAY_DATE
- ANY_ARRAY_BYTE
- ANY_ARRAY_BOOL

Exemple

Soit le DFB suivant :



Le paramètre d'entrée **Présélection** peut être défini de type GDT.

NOTE : Les objets autorisés pour les différents paramètres sont définis dans ce tableau *(voir page 573)*.

8.9 Types de données appartenant aux diagrammes fonctionnels en séquence (SFC)

Vue d'ensemble des types de données de la famille du diagramme fonctionnel en séquence

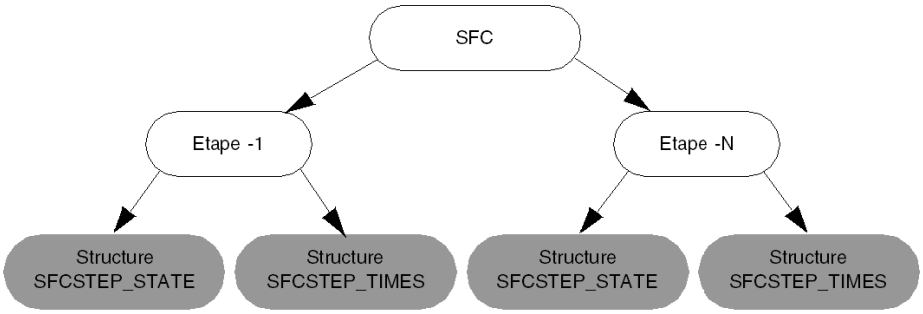
Présentation

La famille des types de données diagrammes fonctionnels en séquence SFC (Sequential function chart) regroupe des types de données **dits composés** tels que des structures restituant les propriétés et l'état du graphe (Chart) et des actions le composant.

Chaque étape est représentée par deux structures qui sont :

- la structure **SFCSTEP_STATE**,
- la structure **SFCSTEP_TIMES**.

Illustration :



NOTE : Les deux types de structures **SFCSTEP_STATE** et **SFCSTEP_TIMES** sont rattachées aussi à chaque Macro-étape du diagramme fonctionnel en séquence.

Définition de la structure de type SFCSTEP_STATE

Cette structure rassemble toutes les données liées à l'état de l'étape ou de la Macro-étape.

Ces données sont:

- **x**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE quand l'étape est active,
- **t**: donnée élémentaire (EDT) de type TIME contenant le temps d'activité de l'étape. Lorsqu'il est désactivé, la valeur de l'étape est maintenue jusqu'à la prochaine activation,
- **tminErr**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE si le temps d'activité de l'étape est inférieur au temps d'activité minimal programmé,
- **tmaxErr**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE si le temps d'activité de l'étape est supérieur au temps d'activité maximal programmé,

Ces données sont accessibles à partir de l'application en lecture seule.

Définition de la structure de type SFCSTEP_TIMES

Cette structure rassemble les données liées aux paramètrages du temps d'exécution de l'étape ou de la Macro-étape.

Ces données sont:

- **delay**: donnée élémentaire (EDT) de type TIME définissant le temps de retard de scrutation de la transition situé en aval de l'étape active,
- **tmin**: donnée élémentaire (EDT) de type TIME contenant la valeur minimale durant laquelle l'étape doit être exécutée, Si cette valeur n'est pas respectée, les données tmin.Err deviennent la valeur TRUE,
- **tmax**: donnée élémentaire (EDT) de type TIM contenant la valeur maximale durant laquelle l'étape doit être exécutée. Si cette valeur n'est pas respectée, les données tmax.Err deviennent la valeur TRUE.

Ces données sont accessibles uniquement à partir de l'éditeur du SFC.

Syntaxe d'accès à des données de la structure SFCSTEP_STATE

Les noms d'instances de cette structure correspondent aux noms des étapes ou macro-étapes du diagramme fonctionnel en séquence

Syntaxe	Commentaire
Nom_Etape.x	Permet de connaitre l'état de l'étape (active\inactive)
Nom_Etape.t	Permet de connaitre le temps d'activation en cours ou total de l'étape
Nom_Etape.tminErr	Permet de connaitre si le temps minimal d'activation de l'étape est inférieur au temps programmé dans Nom-Etape.tmin
Nom_Etape.tmaxErr	Permet de connaitre si le temps maximal d'activation de l'étape est supérieur au temps programmé dans Nom-Etape.tmax

8.10 Compatibilité entre types de données

Compatibilité entre des types de données

Présentation

Ci-dessous sont présentées les différentes règles de compatibilité entre types **à l'intérieur** de chaque famille suivantes :

- la famille Type de données élémentaires (EDT),
- la famille Type de données dérivées (DDT),
- la famille Type de données génériques (GDT).

Famille Type de données élémentaires (EDT)

La famille de types de données élémentaires (EDT) contient ses sous-familles qui sont :

- la sous-famille de types de données au format binaire,
- la sous-famille de types de données au format BCD,
- la sous-famille de types de données au format Réel,
- la sous-famille de types de données au format chaîne de caractères,
- la sous-famille de types de données au format chaîne de bits.

Il n'y a pas de compatibilité entre deux types de données quels qu'ils soient, même s'ils appartiennent à la même sous-famille.

Famille Type de données dérivées (DDT)

La famille de types de données dérivées (DDT) contient ses sous-familles qui sont :

- la sous-famille de type tableaux,
- la sous-famille de type structures :
  - les structures concernant les données d'entrées\sorties (IODDT),
  - structures concernant les autres données.

Règles concernant les structures :

Deux structures sont compatibles si leurs éléments sont :

- de même nom,
- de même type,
- organisés suivant le même ordre.

Soit quatre types de structures :

```
ELEMENT_1
  My_Element : INT
  Other_Element : BOOL
; Structure de type ELEMENT_1


ELEMENT_2
  My_Element: INT
  Other_Element : BOOL
; Structure de type ELEMENT_2

ELEMENT_3
  Element : INT
  Other_Element : BOOL
; Structure de type ELEMENT_3

ELEMENT_4
  Other_Element : BOOL
  My_Element : INT
; Structure de type ELEMENT_4
```

Compatibilité entre les types de structures

Types	**ELEMENT_1**	**ELEMENT_2**	**ELEMENT_3**	**ELEMENT_4**
ELEMENT_1		OUI	NON	NON
ELEMENT_2	OUI		NON	NON
ELEMENT_3	NON	NON		NON
ELEMENT_4	NON	NON	NON	

Règles concernant les tableaux

Deux tableaux sont compatibles si :

- leurs dimensions et l'organisation de leurs dimensions sont identiques,
- chaque dimension correspondante est de même type.

Soit cinq types de tableaux :

```
TAB_1 : ARRAY[10..20]OF INT
; Tableau une dimension de type TAB_1

TAB_2: ARRAY[20..30]OF INT
; Tableau une dimension de type TAB_2

TAB_3: ARRAY[20..30]OF INT
; Tableau une dimension de type TAB_3

TAB_4: ARRAY[20..30]OF TAB_1
; Tableau une dimension de type TAB_4

TAB_5: ARRAY[20..30,10..20]OF INT
; Tableau deux dimensions de type TAB_5
```

Compatibilité entre les types de tableaux:

Le type...	**et le type...**	**sont...**
TAB_1	TAB_2	incompatibles
TAB_2	TAB_3	compatibles
TAB_4	TAB_5	compatibles
TAB_4[25]	TAB_5[28]	compatibles

Famille Type de données génériques (GDT)

La famille de types de données génériques (GDT) est composée d'ensembles organisés de façon hiérarchique qui contiennent des types de données appartenant aux familles :

- de types de données élémentaires (EDT),
- de types de données dérivées (DDT).

Règles :

Un type de données classique est compatible avec les types de données génériques qui lui sont hiérarchiques.

Un type de données génériques est compatible avec les types de données génériques qui lui sont hiérarchiques.

Exemple :

```
Le type INT est compatible avec les types ANY_INT ou ANY_NUM ou
ANY_MAGNITUDE.

Le type INT n'est pas compatible avec le type ANY_BIT ou ANY_REAL.

Le type générique ANY_INT est compatible avec le type ANY_NUM.

Le type générique ANY_INT n'est pas compatible avec le type
ANY_REAL.
```