

## Informatique Industrielle Avancée

- Connaître les systèmes de contrôle de type Automate Programmable Industriel
- Connaître les langages **IEC 61131-3** et leurs extensions sur **CoDeSys3.X**
  - ✓ Langage ST : Structured Text
  - ✓ Langage SFC : Sequential Function Chart
  - ✓ Langage FDB : Function Block Diagram
- ✓ **Apport des Blocs Fonctionnels pour une Programmation Orientée Objet (POO (OOP))**
- ✓ **Approche Objet pour l'Automatisation (OOIP)**



## ❑ PROGRAMMATION ORIENTEE OBJET

- **Technique** qui vise à **calquer la manière rationnelle** dont nous percevons et conceptualisons le monde réel.

### • Notion d'Objet

- ✓ Ensemble de **Propriétés : statique**
  - « **est comme ci, et comme ça !** »
    - des données membres
- ✓ Ensemble de **Services : dynamique**
  - « **sait faire ceci et cela !** »
    - des fonctions membres ou méthodes

### • Notion de Classe

- ✓ **Structure regroupant un ensemble de propriétés** avec les **services qui les régissent.**  
(données membres et des fonctions membres)

### • Objet Etape

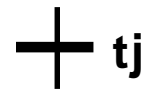
- Propriétés :
  - *Bit d'activité  $X_i$ , durée d'activation ( $t/X_i$ ), son repère « i », qui lui sont propres...*
- Services
  - *INITIALISER, ACTIVER, DESACTIVER, ...*



### • Classe CStep

- Propriétés / Données **membres**:
  - *Bit d'activité, durée d'activation, un repère... ( $m\_xActivityBit$ ,  $m\_tActivationDuration$ , ...)*
- Services / **Méthodes**
  - *MInit, MActivating, MDeActivating, ...*

### • Classe CTransition



- Propriétés :
  - *Sa condition de franchissement, son repère, ses étapes d'entrée, de sortie, ...*
- Services / Méthodes :
  - *FRANCHIR, VALIDEE?, FRANCHISSABLE?,*

## ❑ PROGRAMMATION ORIENTEE OBJET

- ✓ ***Du point de vue externe, un objet forme un tout indissociable. C'est une «Boîte Noire» dont les détails d'implémentation nous sont cachés et qui interagit avec l'extérieur via des fonctions d'interface.***
- ✓ ***L'Encapsulation :***
  - C'est la possibilité de **regrouper** dans une **même structure syntaxique** appelée **Classe**, des **données** qui décrivent l'objet et les **fonctions (méthodes)** qui les manipulent.
  - C'est également aussi **la possibilité de cacher la représentation interne de l'objet** et de ne **laisser visibles de l'extérieur** que les **fonctions d'interface** (notion de **protection**)
- ✓ ***L'Héritage***
  - **Construire de nouvelles Classes par dérivation d'une Classe existante** (vers une **spécialisation**). La nouvelle Classe **hérite des données membres** et des **méthodes** de la **Classe** dont elle est issue. Elle peut ajouter de **nouvelles données** et de **nouvelles fonctionnalités** qui lui sont propres, et/ou **modifier (surcharger)** les méthodes héritées
- ✓ ***Le Polymorphisme***
  - **Possibilité de donner un même nom à des fonctionnalités similaires** bien que celles-ci ne s'appliquent pas sur les mêmes objets. **Leur implémentation peut être différente.**
  - **Le choix de la bonne méthode** est effectué **contextuellement** en fonction de l'objet qui sollicite ce service.

## ❑ Programmation Orientée Objet sous CoDeSys 3.X


- **Principe : Utilisation particulière des Blocs Fonctionnels (BF)**
  - ✓ *La **durée de vie des variables locales (VAR)** d'une fonction est limitée à la durée d'exécution de cette fonction, alors que pour un bloc fonctionnel, elles sont toujours présentes* puisque l'utilisation d'un **BF** nécessite d'en créer une instance.
  - ✓ *Ainsi, ce sont les **variables locales (VAR)** d'un BF qui seront candidates pour constituer les données membres de l'objet réalisé avec une instance de BF.*
  - ✓ *Afin de **distinguer** dans les **variables locales (VAR)** d'un BF, celles qui deviendront les données membres de la classe associée au BF, on utilisera les conventions de notation suivantes :*

```

FUNCTION_BLOCK <CName> (* définition de la classe <CName> *)
VAR
    <type><Name> : <type>;    (* variable locale non accessible de l'extérieur *)
    m_<type><Name> : <type>; (* membre de l'objet accessible de l'extérieur
                                sous certaines conditions *)
END_VAR
    
```




## ❑ LES MOYENS OFFERTS

### • **METHOD :**

- ✓ **Elle peut définir comme une fonction:**
  - son contexte d'E, d'E/S, et de S (**VAR\_INPUT**, **VAR\_IN\_OUT**, **VAR\_OUTPUT**),
  - ses variables locales (**VAR**) dont les **durées de vie** sont liées à sa **durée d'exécution**,
  - le type de la valeur retournée
- ✓ **mais aussi utiliser le contexte de l'instance à laquelle elle est associée.**
  - accéder aux variables locales du  du **Bloc Fonctionnel** auquel elle appartient.
- ✓ **et déclarer son attribut d'accessibilité** (cachée/non cachée)
  - Choix entre **{PRIVATE, INTERNAL, PROTECTED / PUBLIC}**

### • **PROPERTY :**

- ✓ **Elles permettent à l'aide de **METHODs** dites **accesseurs** d'accéder aux variables internes auxquelles elles font référence depuis l'extérieur.** Par exemple, autoriser l'accès aux données membres associées aux variables locales (**VAR**) d'un bloc fonctionnel.
- ✓ Convention de nom pour une **PROPERTY** associée à une donnée membre :
 

(* données membres *)	<=>	(*  <b>PROPERTYs</b> associées *)
<i>m_&lt;type&gt;&lt;Name&gt; : &lt;type&gt;</i>	<=>	 <b>PROPERTY</b> <type><Name> : <type>
<i>m_rGainP : REAL</i>	<=>	 .....




## ❑ LES MOYENS OFFERTS

### • **METHOD :**

- ✓ **Elle peut définir comme une fonction:**
  - son contexte d'E, d'E/S, et de S (**VAR\_INPUT**, **VAR\_IN\_OUT**, **VAR\_OUTPUT**),
  - ses variables locales (**VAR**) dont les **durées de vie** sont liées à sa **durée d'exécution**,
  - le type de la valeur retournée
- ✓ **mais aussi utiliser le contexte de l'instance à laquelle elle est associée.**
  - accéder aux variables locales du **POU**, du **Bloc Fonctionnel** auquel elle appartient.
- ✓ **et déclarer son attribut d'accessibilité** (cachée/non cachée)
  - Choix entre **{PRIVATE, INTERNAL, PROTECTED / PUBLIC}**

### • **PROPERTY :**

- ✓ **Elles permettent à l'aide de **METHODs** dites accesseurs d'accéder aux variables internes auxquelles elles font référence depuis l'extérieur.** Par exemple, autoriser l'accès aux données membres associées aux variables locales (**VAR**) d'un bloc fonctionnel.
- ✓ Convention de nom pour une **PROPERTY** associée à une donnée membre :
 

(* données membres *)	<=>	(*  <b>PROPERTYs associées</b> *)
<b>m_&lt;type&gt;&lt;Name&gt; : &lt;type&gt;</b>	<=>	 <b>PROPERTY &lt;type&gt;&lt;Name&gt; : &lt;type&gt;</b>
<b>m_rGainP : REAL</b>	<=>	 <b>PROPERTY rGainP : REAL</b>

[Pascal.Aygalinc@univ-smb.fr](mailto:Pascal.Aygalinc@univ-smb.fr)



## ❑ LES MOYENS OFFERTS

### • INTERFACE :


#### ✓ *Contrat pour organiser et standardiser :*

- les prototypes des **METHODs** en imposant leur contexte (**VAR\_INPUT**, **VAR\_IN\_OUT**, **VAR\_OUTPUT**), le **type** de la **valeur retournée**,
- les **PROPERTYs** (leur **type**, leurs **accesseurs**)

#### ✓ *mais sans en donner l'implémentation.*

### • **EXTENDS** : *mot clé pour la dérivation*

#### ✓ Elle peut s'appliquer :

- aux **DUT**  e type **STRUCT**,
- aux **INTERFACES**,
- aux **Blocs Fonctionnels** (**FUNCTION\_BLOCKS**)

#### ✓ Objectifs : **hériter** de leur structure, de leurs **METHODs**, de leurs **PROPERTYs** :

- Pour **gagner en temps de développement**
- récupérer du **savoir-faire**,
- et éventuellement les **adapter** (*notion de surcharge*) à une **situation plus particulière**.



## ❑ LES MOYENS OFFERTS

- **POINTEUR** *THIS* (\* **case sensitif** : *this* ≠ *THIS* \*)
  - ✓ *Pointeur* qui est automatiquement **disponible dans tout bloc fonction.**
  - ✓ *Paramètre implicite* qui désigne l'objet receveur sur lequel les **METHODs** s'appliquent.
  - ✓ *Son utilisation n'est valide que* dans le cadre d'une **POO**.
    - dans *le corps du bloc fonctionnel*
    - dans les **METHODs** associées
  - ✓ *THIS* permet à l'aide de *l'opérateur* ^ :
    - *d'expliciter* l'accès aux **composantes** d'un objet (*membres, méthodes*)
      - *THIS*^.MUseMe ( ); (\* ⇔ MUseMe( ); \*)
    - *de démasquer* les composantes d'un objet si des variables, des **PROPERTYs** portent *les mêmes noms*.
      - *THIS*^.ilIndex := ilIndex; (\* démasquage de la composante **ilIndex** par rapport à la variable **ilIndex** \*)
    - *de désigner l'objet tout entier* (*instance du bloc fonctionnel*) lors d'un passage de paramètre à une fonction
      - *MyFunction*( pFunctionBlock := *THIS*);



## ❑ LES MOYENS OFFERTS

### • POINTEUR **SUPER**

- ✓ **Pointeur** qui est automatiquement **disponible dans tout bloc fonction créé par dérivation**.
- ✓ **Son utilisation n'est valide** que dans le cadre d'une **POO**
  - dans **le corps des blocs fonctionnels construit par dérivation**
  - dans les **METHODs** associées à ces blocs fonctionnels dérivés
- ✓ **SUPER** permet à l'aide de **l'opérateur ^** :
  - **d'accéder aux METHODs** du **bloc fonctionnel parent** quand **celles-ci** ont été **surchargées** dans les **blocs fonctionnels obtenus par dérivation de ce dernier**.
  - **FUNCTION\_BLOCK CParent** (\* Classe de base CParent \*)  
 **MUseMe()**
  - **FUNCTION\_BLOCK CExtParent EXTENDS CParent** (\* Classe dérivée de CParent \*)  
 **MUseMe()** (\* surcharge de la METHOD MUseMe() de la classe CParent \*)
  - **Dans la classe dérivée CExtBF :**
    - **THIS^**. MUseMe(); (\* ⇔ appel de la surcharge MUseMe() \*)
    - **SUPER^**. MUseMe(); (\* ⇔ appel de la METHOD MUseMe() de la classe CParent \*)

## ❑ LES MOYENS OFFERTS

### • Initialisation des objets (*Construction*)

- ✓ **Construire un objet** consiste à lui donner **des valeurs spécifiques d'initialisation à chacun de ses membres** (généralement différentes de 0). **Deux possibilités sont offertes :**
  - Soit à l'aide d'une **METHOD** dite **constructeur** dont le nom imposé est ici **FB\_Init**,
  - Soit **lors de la déclaration/réservation de l'objet** via **une affectation des PROPERTYs** qui **donnent accès aux données membres** de l'objet.
- ✓ **L'initialisation via la METHOD FB\_Init est plus générale car toutes les PROPERTYs ne sont pas forcément accessibles en écriture.**
- ✓ **L'initialisation qui s'effectue en premier est celle obtenue via le constructeur FB\_Init. Elle s'effectue de façon transparente et cachée. Celle donnée via les PROPERTYs lors de la déclaration/réservation de l'objet s'effectuera après.**
- ✓ **La METHOD FB\_Init est déclarée implicitement. Pour pouvoir définir une initialisation particulière, il faut la déclarer explicitement pour la surcharger. Son prototype est donné par :**




**METHOD FB\_Init : BOOL**  
**VAR\_INPUT**

**blnitRetains : BOOL;**  
**blnCopyCode : BOOL;**

**END\_VAR**

**Pascal.Aygalinc@univ-smb.fr**


- ❑  **METHOD** <Attribut> <Method\_Name> | : <return\_data\_type>
  - <Attribut> : permet de spécifier l'accessibilité de la **METHOD**.
    - ✓ **PUBLIC** : accessibilité totale (pas forcément judicieux !!)
    - ✓ **PRIVATE** : accessibilité réduite à l'entité à laquelle elle est associée
    - ✓ **PROTECTED** : **PRIVATE** pour l'extérieur / **PUBLIC** pour la dérivation.
    - ✓ **INTERNAL** : accessibilité réduite à l'espace de noms de la bibliothèque
  - Règle de nommage : Préfixe M
    - ✓ **METHOD PROTECTED** MDolt : **BOOL**
  - L'appel d'une **METHOD** peut se faire via un autre **METHOD**, un **POINTER**, via une **INTERFACE**. D'où :
    - ✓ Les **METHODs** favorisent la **modularité** des **blocs fonctionnels** puisqu'il est **possible de décomposer** le corps d'un **BF** en **plusieurs appels** de **METHODs**.
    - ✓ Les **METHODs** favorisent aussi **leur intégrité** puisque l'**accessibilité** d'une **METHOD** peut être **réduite** (cf. <Attribut>)
  - Les **METHODs** sont **héritées** en cas de dérivation. Il est alors possible d'en effectuer une **surcharge** pour les adapter, ou de les **supprimer** si celles héritées conviennent.

## ❏ **PROPERTY** <Property\_name> : <Type> (standard ou DUT)

- L'**encapsulation des données** est obtenue grâce à la **notion** de **PROPERTY**. Cette dernière permet un **accès externe** aux données : **locales VAR** d'une instance d'un **POU** ou d'un **BF**, d'une **Global Variable List (GVL)**. Elle agit en même temps comme un **filtre** ou une **surcharge**.
- Une **PROPERTY** fournit **par défaut** un ensemble de **METHODs** appelées **accesseurs** dont les noms imposés sont **Get** et **Set** :
  - ✓ **Set** permet de **fixer la valeur** (écriture) **de la donnée membre** à laquelle elle est associée, alors que **Get** permet **d'en connaître la valeur** (lecture).

 **C<Name>.<typeName>.Get**

 <typeName> := **THIS**^.m\_<typeName>;

 **C<Name>.<typeName>.Set**
**THIS**^.m\_<typeName> := <typeName>;

- ✓ Pour qu'une **PROPERTY** puisse être **uniquement lue** (respectivement **uniquement écrite**), il suffit de détruire la **METHOD** accesseur qui contredit cette spécification (pour la lecture, détruire **Set**, pour l'écriture, détruire **Get**).
- Les **PROPERTYs** sont aussi **héritées** en cas de **dérivation**. Il est alors possible d'en effectuer une **surcharge**, ou de **les supprimer si celles héritées** (de base pour la première dérivation) **conviennent**.

## ❏ **INTERFACE** <ITF\_Name> (directive : **IMPLEMENTS**)

- Une **INTERFACE** permet **d'établir** pour un **bloc fonctionnel** la liste des **METHODs** et **PROPERTYs** **qu'il doit prendre en charge** en précisant :
  - ✓ Pour les **PROPERTYs** : son **nom**, son **type** et ses accesseurs **Get et/ou Set**.
  - ✓ Pour les **METHODs** : **uniquement** leurs **prototypes** en précisant les paramètres **E**, **E/S**, et **S** (**VAR\_INPUT**, **VAR\_IN\_OUT** et **VAR\_OUTPUT**), le type de la valeur retournée, mais sans en préciser le **code**.
  - ✓ Les **INTERFACES** sont donc des **contrats** pour **standardiser et organiser** l'accès aux **METHODs** et aux **PROPERTYs** d'un bloc fonctionnel.
- Un **même bloc fonctionnel** peut être soumis à **plusieurs INTERFACES**.
  - ✓ **FUNCTION\_BLOCK** <fb\_name> **IMPLEMENTS** <ITF\_name\_0> |, <ITF\_name\_1> |, <ITF\_name\_n>
- On peut aussi **dériver** une **INTERFACE** pour la compléter.
  - ✓ **INTERFACE** <ITF\_Ext\_Name> **EXTENDS** <ITF\_Name>
- **L'appel d'une METHOD d'un BF** peut s'effectuer à l'aide de son **INTERFACE** (moyen utile pour satisfaire le **polymorphisme**)

## ❏ **INTERFACE** <ITF\_Name> (directive : **IMPLEMENTS**)

- Si *des blocs fonctionnels de fonctionnalité différente* ont en commun *une même INTERFACE* (ils *s'accordent* donc sur un *même contrat*), ils *peuvent être traités* alors comme un *groupe homogène*.
- ✓ Ce qui autorise un *appel plus simple* des **METHODs** qu'ils ont en commun à l'aide de cette **INTERFACE** (on satisfait la notion du **polymorphisme**).

❏ **INTERFACE** <ITF\_Com>

 **METHOD** [PUBLIC] MUseMe

**FUNCTION\_BLOCK** CObj1 **IMPLEMENTS** <ITF\_Com>, <ITF\_spObj1>, ...

**FUNCTION\_BLOCK** CObj2 **IMPLEMENTS** <ITF\_Com>, <ITF\_spObj2>, ...

**VAR**

obj1 : CObj1 ; (\* réservation d'un objet de la classe CObj1 \*)

obj2 : CObj2 ;

itfCom : <ITF\_Com>; (\* réservation de type <ITF\_Com> \*)

**END\_VAR**

itfCom.MUseMe( ); (\* appel de la **METHOD** MUseMe via l'**INTERFACE** itfCom

⇔ obj1.MUseMe( ) si itfCom=obj1,

⇔ obj2.MUseMe( ) si itfCom=obj2 \*)








## ❑ Mise en œuvre

🔑 **INTERFACE** *ITF\_ObjStep* (\* **contrat pour la Classe CStep** \*)

```

 PROPERTY .....
 PROPERTY .....
 PROPERTY .....
...
 METHOD .....
 METHOD .....
 METHOD .....
...
  
```

```

FUNCTION_BLOCK .....
VAR
  .....
  .....
  .....
END_VAR
  
```

```

  .....
  .....
  .....
  .....
  .....
  
```



## ❑ Mise en œuvre

```

INTERFACE ITF_ObjStep (* contrat pour la Classe CStep *)
  PROPERTY xActivityBit : BOOL (* bit d'activité : accesseur Get *)
  PROPERTY tActivationDuration : TIME (* durée d'activité : Get *)
  PROPERTY tTaskPeriod : TIME (* période de la tâche: Set/Get *)
  ...
  METHOD .....
  METHOD .....
  METHOD .....
  ...
  
```

```

FUNCTION_BLOCK .....
VAR
  .....
  .....
  .....
END_VAR
  
```

```

.....
.....
.....
.....
.....
  
```



## ❑ Mise en œuvre

```

INTERFACE ITF_ObjStep (* contrat pour la Classe CStep *)
  PROPERTY xActivityBit : BOOL (* bit d'activité : accesseur Get *)
  PROPERTY tActivationDuration : TIME (* durée d'activité : Get *)
  PROPERTY tTaskPeriod : TIME (* période de la tâche: Set/Get *)
  ...
  METHOD MInit;
  METHOD MActivating;
  METHOD MDeActivating;
  ...
  
```

```

FUNCTION_BLOCK .....
VAR
  .....
  .....
  .....
END_VAR
  
```

```

  .....
  .....
  .....
  .....
  .....
  
```



## ❑ Mise en œuvre

```

INTERFACE ITF_ObjStep (* contrat pour la Classe CStep *)
  PROPERTY xActivityBit : BOOL (* bit d'activité : accesseur Get *)
  PROPERTY tActivationDuration : TIME (* durée d'activité : Get *)
  PROPERTY tTaskPeriod : TIME (* période de la tâche: Set/Get *)
  ...
  METHOD MInit;
  METHOD MActivating;
  METHOD MDeActivating;
  ...
  
```

```

FUNCTION_BLOCK CStep IMPLEMENTS ITF_ObjStep
VAR
    m_xActivityBit          : BOOL;
    m_tActivationDuration   : TIME ;
    m_tTaskPeriod           : TIME ;
END_VAR
  
```







```

.....
.....
.....
.....
.....
  
```



## ❑ Mise en œuvre

```

INTERFACE ITF_ObjStep (* contrat pour la Classe CStep *)
     PROPERTY xActivityBit : BOOL (* bit d'activité : accesseur Get *)
     PROPERTY tActivationDuration : TIME (* durée d'activité : Get *)
     PROPERTY tTaskPeriod : TIME (* période de la tâche: Set/Get *)
    ...
     METHOD Minit ;
     METHOD MActivating ;
     METHOD MDeActivating;
    ...
    
```

**FUNCTION\_BLOCK** CStep **IMPLEMENTS** ITF\_ObjStep

**VAR**

```

        m_xActivityBit           : BOOL;
        m_tActivationDuration     : TIME ;
        m_tTaskPeriod             : TIME ;
    
```

**END\_VAR**

**IF** (THIS^.m\_xActivityBit) **THEN**

```

        THIS^.m_tActivationDuration :=      THIS^.m_tActivationDuration
        + THIS^.m_tTaskPeriod ;
    
```

**END\_IF**

...



## ❑ Mise en œuvre


**FUNCTION\_BLOCK** CStep **IMPLEMENTS** ITF\_ObjStep  
**VAR**


m\_xActivityBit : **BOOL**;  
 m\_tActivationDuration : **TIME**;  
 m\_tTaskPeriod : **TIME**;


**END\_VAR**


- **Accesseurs associés aux *PROPERTYs* (xActivityBit, ...)**

```

 CStep. ....
.....

 CStep. ....
.....

 CStep. ....
.....

 CStep ....
.....
.....
.....
.....
.....

```




## ❑ Mise en œuvre

**FUNCTION\_BLOCK** CStep **IMPLEMENTS** ITF\_ObjStep  
**VAR**


m\_xActivityBit : **BOOL**;  
 m\_tActivationDuration : **TIME**;  
 m\_tTaskPeriod : **TIME**;


**END\_VAR**

- **Accesseurs associés aux *PROPERTYs* (xActivityBit, ...)**

 **CStep.xActivityBit.Get**  
                   xActivityBit := **THIS**^.m\_xActivityBit ;

 **CStep** .....  
 .....  
 .....

 **CStep.** .....  
 .....  
 .....

 **CStep** .....  
 .....  
 .....  
 .....  
 .....

## ❑ Mise en œuvre

**FUNCTION\_BLOCK** CStep **IMPLEMENTS** ITF\_ObjStep  
**VAR**

m\_xActivityBit : **BOOL**;  
 m\_tActivationDuration : **TIME**;  
 m\_tTaskPeriod : **TIME**;

**END\_VAR**

- **Accesseurs associés aux *PROPERTYs* (xActivityBit, ...)**

 **CStep.xActivityBit.Get**

xActivityBit := **THIS**^.m\_xActivityBit ;

 **CStep.tActivationDuration.Get**

tActivationDuration := **THIS**^.m\_tActivationDuration ;

 **CStep.** .....

 **CStep** .....

.....  
 .....  
 .....  
 .....  
 .....  
 .....

## ❑ Mise en œuvre

```
FUNCTION_BLOCK CStep IMPLEMENTS ITF_ObjStep
VAR
    m_xActivityBit          : BOOL;
    m_tActivationDuration   : TIME;
    m_tTaskPeriod           : TIME;
END_VAR
```

- **Accesseurs associés aux *PROPERTYs* (xActivityBit, ...)**

```

CStep.xActivityBit.Get
    xActivityBit := THIS^.m_xActivityBit ;

CStep.tActivationDuration.Get
    tActivationDuration := THIS^.m_tActivationDuration ;

CStep.tTaskPeriod.Get
    tTaskPeriod := THIS^.m_tTaskPeriod ;

CStep.tTaskPeriod.Set (* valeur admissible [T#1ms, T#10s]*)
    IF ((tTaskPeriod >=T#1ms) AND (tTaskPeriod <=T#10s)) THEN
        THIS^.m_tTaskPeriod := tTaskPeriod ;
    END_IF
```

## ❑ Mise en œuvre

 **METHOD** *FB\_Init* : **BOOL** (\* constructeur pour la Classe CStep \*)

**VAR\_INPUT**

    bInitRetains           : **BOOL**;

    bInCopyCode         : **BOOL**;

**END\_VAR**



.....

.....

.....

.....

 **CStep.MInit**

.....

.....

 **CStep.MActivating**

.....

.....

.....

.....

 **CStep.MDeActivating**

.....

## ❑ Mise en œuvre

**METHOD** *FB\_Init* : **BOOL** (\* constructeur pour la Classe CStep \*)

**VAR\_INPUT**

    bInitRetains           : **BOOL**;

    bInCopyCode          : **BOOL**;



**END\_VAR**

**THIS**^.m\_xActivityBit                := **FALSE** ;

**THIS**^.m\_tActivationDuration       := T#0s;

**THIS**^.m\_tTaskPeriod                := T#1ms;

**FB\_Init** := **TRUE**;

**CStep.MInit**

.....  
.....

**CStep.MActivating**

.....  
.....  
.....  
.....

**CStep.MDeActivating**

.....



## ❑ Mise en œuvre

 **METHOD** *FB\_Init* : **BOOL** (\* constructeur pour la Classe CStep \*)

**VAR\_INPUT**

    bInitRetains           : **BOOL**;

    bInCopyCode         : **BOOL**;



**END\_VAR**

**THIS**^.m\_xActivityBit                 := **FALSE** ;

**THIS**^.m\_tActivationDuration         := T#0s;

**THIS**^.m\_tTaskPeriod                 := T#1ms;

**FB\_Init** := **TRUE**;

 **CStep.MInit**

**THIS**^.m\_xActivityBit                 := **FALSE** ;

**THIS**^.m\_tActivationDuration         := T#0s;

 **CStep.MActivating**

.....

.....

.....

.....

 **CStep.MDeActivating**

.....



## ❑ Mise en œuvre

 **METHOD** *FB\_Init* : **BOOL** (*\* constructeur pour la Classe CStep \**)

**VAR\_INPUT**

    bInitRetains           : **BOOL**;

    bInCopyCode         : **BOOL**;



**END\_VAR**

**THIS**^.m\_xActivityBit                 := **FALSE** ;

**THIS**^.m\_tActivationDuration         := T#0s;

**THIS**^.m\_tTaskPeriod                 := T#1ms;

**FB\_Init** := **TRUE**;

 **CStep.MInit**

**THIS**^.m\_xActivityBit                 := **FALSE** ;

**THIS**^.m\_tActivationDuration         := T#0s;

 **CStep.MActivating**

**THIS**^.m\_xActivityBit                 := **TRUE** ;

**IF** (RisingEdge (**THIS**^.m\_xActivityBit)) **THEN**  
         **THIS**^.m\_tActivationDuration         := T#0s;

**END\_IF**

 **CStep.MDeActivating**


**THIS**^.m\_xActivityBit                 := **FALSE** ;

## ❑ Mise en œuvre

**PROGRAM** UserCase

**VAR**

```

    oStep      : CStep                := (tTaskPeriod := T#10ms);
    (*  FB_Init :      oStep.m_xActivityBit          ← FALSE,
                                   oStep.m_tActivationDuration      ← T#0s,
                                   oStep.m_tTaskPeriod              ← T#1ms *)
     CStep.tTaskPeriod.Set : oStep.m_tTaskPeriod    ← T#10ms *)

    itfStep     : ITF_ObjStep          := oStep;
    pStep       : POINTER TO CStep     := ADR(oStep);
    xFirstCycleRun : BOOL              := TRUE;
  
```

**END\_VAR**

**IF** (xFirstCycleRun) **THEN**

```

    .....
    xFirstCycleRun := FALSE;
  
```

**END\_IF**

```

    .....
    .....
    .....
    .....
  
```

## ❑ Mise en œuvre

**PROGRAM** UserCase

**VAR**

```

    oStep      : CStep                := (tTaskPeriod := T#10ms);
    (*  FB_Init :      oStep.m_xActivityBit          ← FALSE,
                                oStep.m_tActivationDuration      ← T#0s,
                                oStep.m_tTaskPeriod              ← T#1ms *)
     CStep.tTaskPeriod.Set : oStep.m_tTaskPeriod      ← T#10ms *)

    itfStep     : ITF_ObjStep          := oStep;
    pStep       : POINTER TO CStep    := ADR(oStep);
    xFirstCycleRun : BOOL             := TRUE;

```

**END\_VAR**

**IF** (xFirstCycleRun) **THEN**

```

    oStep.MActivating();
    xFirstCycleRun := FALSE;

```

**END\_IF**

**IF** (itfStep.tActivationDuration > T#30s) **THEN**

```

    pStep^.MDeActivating ();



```

**END\_IF**

**oStep();**

SAVOIE  
MONT BLANC

## ❑ Apports de la dérivation

- Comparaison entre Etape et Etape Initiale  
  - ✓ À  $t = 0$ , l'état du bit d'activité diffère ( $X_i$  vaut 0 alors que  $X_j$  vaut 1)
  - ✓ Elles possèdent **les mêmes services**, seule leur **initialisation change**.
  - ✓ **Les étapes initiales** sont donc des **cas particuliers d'étape**. En appliquant le **principe d'inclusion**, la classe **ClnitStep** sera obtenue par **dérivation** de la classe **CStep**.
  - ✓ Pour prendre en compte **cette différence d'état du bit d'activité à  $t = 0$** , il faut **surcharger la **METHOD** Mlnit** ainsi que le **constructeur de la classe parent CStep (METHOD FB\_Init)**.
  - ✓ **Comme les autres **METHODs** et les **PROPERTYs** héritées de la classe CStep conviennent pour la ClnitStep, il n'y a pas de nouveaux développements à faire, d'où l'intérêt de la dérivation.**

 **ClnitStep.Mlnit**

```
.....
.....
```

## ❑ Apports de la dérivation

### • Comparaison entre Etape et Etape Initiale



- ✓ À  $t = 0$ , l'état du bit d'activité diffère ( $X_i$  vaut 0 alors que  $X_j$  vaut 1)
- ✓ Elles possèdent **les mêmes services**, seule leur **initialisation change**.
- ✓ **Les étapes initiales** sont donc des **cas particuliers d'étape**. En appliquant le **principe d'inclusion**, la classe **ClnitStep** sera obtenue par **dérivation** de la classe **CStep**.
- ✓ Pour prendre en compte **cette différence d'état du bit d'activité à  $t = 0$** , il faut **surcharger la **METHOD** Mlnit** ainsi que le **constructeur de la classe parent CStep (METHOD FB\_Init)**.
- ✓ **Comme les autres **METHODs** et les **PROPERTYs** héritées de la classe CStep conviennent pour la ClnitStep, il n'y a pas de nouveaux développements à faire, d'où l'intérêt de la dérivation.**



**ClnitStep.Mlnit**

```
THIS^.m_xActivityBit := TRUE ;
THIS^.m_tActivationDuration := T#0s;
```

## ❑ Apports de la dérivation

 **ClnitStep.FB\_Init** (\* constructeur pour la Classe ClnitStep \*)



**METHOD FB\_Init : BOOL**

**VAR\_INPUT**

**blnitRetains : BOOL;**

**blnCopyCode : BOOL;**

**END\_VAR**

.....  
.....  
.....  
.....

**FUNCTION\_BLOCK** .....

**VAR**

**END\_VAR**

.....  
.....  
.....  
.....  
.....





## ❑ Apports de la dérivation

 **ClnitStep.FB\_Init** (\* constructeur pour la Classe ClnitStep \*)

**METHOD FB\_Init : BOOL**

**VAR\_INPUT**

    bInitRetains           : **BOOL**;

    bInCopyCode          : **BOOL**;

**END\_VAR**

**THIS**^.m\_xActivityBit           := **TRUE** ;

**THIS**^.m\_tActivationDuration   := **T#0s**;

**THIS**^.m\_tTaskPeriod           := **T#1ms**;

**FB\_Init := TRUE**;

**FUNCTION\_BLOCK** .....

**VAR**

**END\_VAR**

.....  
.....  
.....  
.....  
.....



## □ Apports de la dérivation

 **ClnitStep.FB\_Init** (\* constructeur pour la Classe ClnitStep \*)



**METHOD FB\_Init : BOOL**

**VAR\_INPUT**

**blnitRetains** : **BOOL**;

**blnCopyCode** : **BOOL**;

**END\_VAR**

**THIS**^.m\_xActivityBit := **TRUE** ;

**THIS**^.m\_tActivationDuration := **T#0s**;

**THIS**^.m\_tTaskPeriod := **T#1ms**;

**FB\_Init := TRUE**;

**FUNCTION\_BLOCK ClnitStep EXTENDS CStep**

**VAR**

**END\_VAR**

**IF (THIS**^.m\_xActivityBit) **THEN**

**THIS**^.m\_tActivationDuration := **THIS**^.m\_tActivationDuration  
     + **THIS**^.m\_tTaskPeriod ;

**END\_IF**

...

## □ Apports de la dérivation

**PROGRAM** UserCase

**VAR**

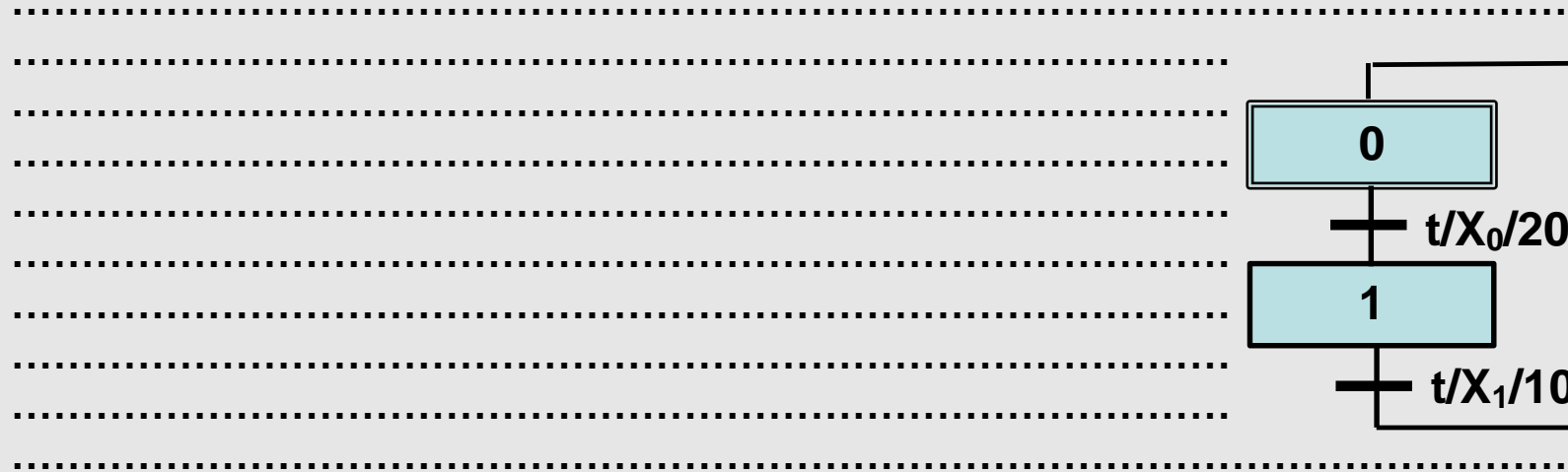
```

    oStep0      : CInitStep           := (tTaskPeriod := T#10ms);
    oStep1      : CStep               := (tTaskPeriod := T#10ms);
    itfStep     : ITF_ObjStep ;
    pStep       : POINTER TO CStep ;
  
```

**END\_VAR**

```

itfStep      := oStep0;
pStep       := ADR(oStep0);
  
```



## □ Apports de la dérivation

```
PROGRAM UserCase
```

```
VAR
```

```

  oStep0      : CInitStep           := (tTaskPeriod := T#10ms);
  oStep1      : CStep               := (tTaskPeriod := T#10ms);
  itfStep     : ITF_ObjStep ;
  pStep       : POINTER TO CStep ;

```

```
END_VAR
```

```
itfStep := oStep0;
```

```
pStep := ADR(oStep0);
```

```
IF ((oStep0.tActivationDuration >= T#20s) AND itfStep.xActivityBit) THEN
```

```
  pStep^.MDeActivating ();
```

```
  pStep := ADR(oStep1);
```

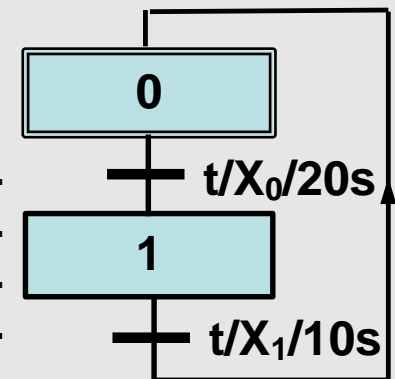
```
  pStep^.MActivating ();
```

```
END_IF
```

```

.....
.....
.....
.....
.....
.....
.....

```



## □ Apports de la dérivation

```
PROGRAM UserCase
```

```
VAR
```

```

  oStep0      : CInitStep           := (tTaskPeriod := T#10ms);
  oStep1      : CStep               := (tTaskPeriod := T#10ms);
  itfStep     : ITF_ObjStep ;
  pStep       : POINTER TO CStep ;

```

```
END_VAR
```

```
itfStep := oStep0;
```

```
pStep := ADR(oStep0);
```

```
IF ((oStep0.tActivationDuration >= T#20s) AND itfStep.xActivityBit) THEN
```

```
  pStep^.MDeActivating ();
```

```
  pStep := ADR(oStep1);
```

```
  pStep^.MActivating ();
```

```
END_IF
```

```
IF ((oStep1.tActivationDuration >= T#10s) AND oStep1.xActivityBit ) THEN
```

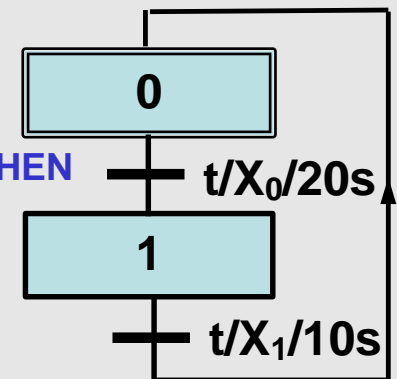
```
  oStep1.MDeActivating();
```

```
  oStep0.MActivating();
```

```
END_IF
```

```
oStep0();
```

```
oStep1();
```



## ❑ ABSTRACT

- Ce **qualificatif** appliqué aux **METHODs** et aux **FUNCTION\_BLOCKS** permet de définir des concepts puisque l'implémentation n'est pas à fournir, **seul le prototype** est à donner.

 **METHOD ABSTRACT** M<Name> : <type>

**FUNCTION BLOCK ABSTRACT** C<AbsName> **IMPLEMENTS** ITF\_ObjAbsName

- Ce qui entraine :
  - ✓ **Seule la surcharge** d'une **METHOD ABSTRACT** peut fournir un résultat.
  - ✓ Un **FUNCTION\_BLOCK ABSTRACT** ne peut donc pas donner naissance à des objets. Seul un **bloc fonctionnel issu de sa dérivation** pourra le faire.

**FUNCTION BLOCK** C<Name> **EXTENDS** CAbsName **IMPLEMENTS** ITF\_ObjName

- ✓ Dans un **FUNCTION\_BLOCK ABSTRACT**, toutes les **METHODs** ne sont pas **forcement abstraites**.
- ✓ Une **METHOD** (abstraite ou non) peut faire appel à des **METHOD** abstraites.

## ❑ ABSTRACT

- Dans la bibliothèque **standard**, **trois types de temporisateurs** sont disponibles.
  - ✓ Les temporisateurs qui ont un **retard à l'enclenchement** (TON),
  - ✓ Les temporisateurs qui ont **retard au déclenchement** (TOF),
  - ✓ Les temporisateurs de type **impulsionnel** (TP).
- Ils proposent tous la même interface :**

**FUNCTION\_BLOCK** Temporisateur

(\* Temporisateur  $\in \{ TON, TOF, TP \}$  \*)

**VAR\_INPUT**

**IN:** BOOL; (\* INput \*)

**PT:** TIME; (\* Preset Time \*)

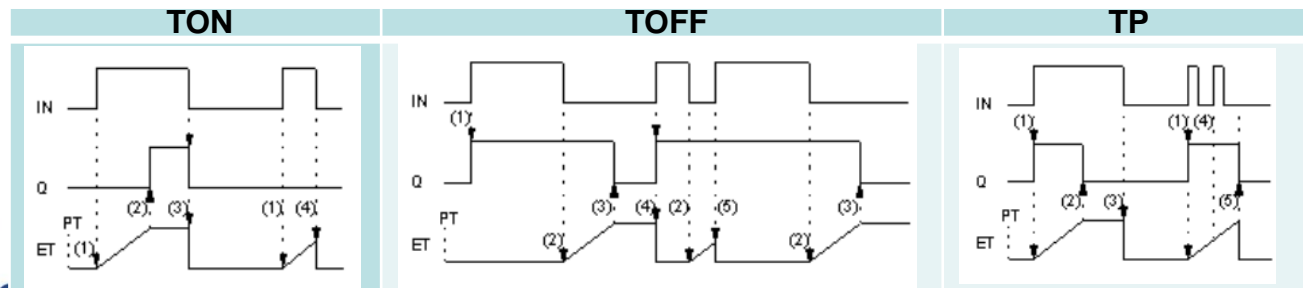
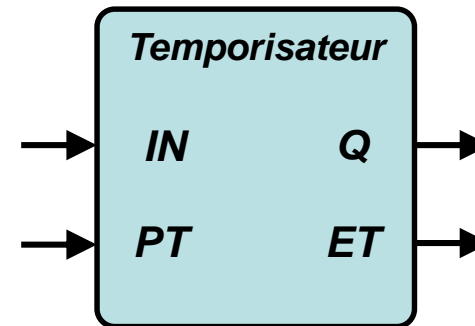
**END\_VAR**

**VAR\_OUTPUT**

**Q:** BOOL; (\* Output \*)

**ET:** TIME; (\* Elapsed Time \*)

**END\_VAR**



## ❑ ABSTRACT

- Comme les **trois types de temporisateurs** proposent tous la même **interface**, on peut définir une **classe abstraite CTimer** dont le rôle est de **fournir un concept** pour le développement d'objets temporisateur.

🔑 **INTERFACE ITF\_ObjTimer (\* contrat pour la Classe CTimer \*)**

```

PROPERTY .....
PROPERTY .....
PROPERTY .....
PROPERTY .....
METHOD .....
```

```

FUNCTION_BLOCK .....
VAR
    .....
    .....
    .....
    .....
END_VAR
```

```






.....
```



## ❑ ABSTRACT

- Comme les **trois types de temporisateurs** proposent tous la même **interface**, on peut définir une **classe abstraite CTimer** dont le rôle est de **fournir un concept** pour le développement d'objets temporisateur.

```

🔑 INTERFACE ITF_ObjTimer (* contrat pour la Classe CTimer *)
     PROPERTY xIN : BOOL (* entrée IN : accesseurs Get et Set *)
     PROPERTY tPT : TIME (* Preset Time : accesseurs Get et Set *)
     PROPERTY xQ : BOOL (* sortie Q : accesseur Get *)
     PROPERTY tET : TIME (* Elapsed Time : accesseur Get *)
     METHOD .....
```

```

FUNCTION_BLOCK .....
VAR
    .....
    .....
    .....
    .....
END_VAR
```






```

.....
SAVOIE
MONT BLANC
```

## ❑ ABSTRACT

- Comme les **trois types de temporisateurs** proposent tous la même **interface**, on peut définir une **classe abstraite CTimer** dont le rôle est de **fournir un concept** pour le développement d'objets temporisateur.

```

🔑 INTERFACE ITF_ObjTimer (* contrat pour la Classe CTimer *)
   PROPERTY xIN : BOOL (* entrée IN : accesseurs Get et Set *)
   PROPERTY tPT : TIME (* Preset Time : accesseurs Get et Set *)
   PROPERTY xQ : BOOL (* sortie Q : accesseur Get *)
   PROPERTY tET : TIME (* Elapsed Time : accesseur Get *)
   METHOD ABSTRACT MTimer;
```

```

FUNCTION_BLOCK .....
VAR
  .....
  .....
  .....
  .....
END_VAR
```

```

.....
SAVOIE
MONT BLANC
```

## ❑ ABSTRACT

- Comme les **trois types de temporisateurs** proposent tous la même **interface**, on peut définir une **classe abstraite CTimer** dont le rôle est de **fournir un concept** pour le développement d'objets temporisateur.

```

🔑 INTERFACE ITF_ObjTimer (* contrat pour la Classe CTimer *)
    PROPERTY xIN : BOOL (* entrée IN : accesseurs Get et Set *)
    PROPERTY tPT : TIME (* Preset Time : accesseurs Get et Set *)
    PROPERTY xQ : BOOL (* sortie Q : accesseur Get *)
    PROPERTY tET : TIME (* Elapsed Time : accesseur Get *)
    METHOD ABSTRACT MTimer;
```

```

FUNCTION_BLOCK ABSTRACT CTimer IMPLEMENTS ITF_ObjTimer
```

```

VAR
```

```

.....
.....
.....
.....
```

```

END_VAR
```

```

.....
```



## ❑ ABSTRACT

- Comme les *trois types de temporisateurs* proposent tous la même *interface*, on peut définir une *classe abstraite CTimer* dont le rôle est de *fournir un concept* pour le développement d'objets temporisateur.

```

🔑 INTERFACE ITF_ObjTimer (* contrat pour la Classe CTimer *)
    📄 PROPERTY xIN : BOOL (* entrée IN : accesseurs Get et Set *)
    📄 PROPERTY tPT : TIME (* Preset Time : accesseurs Get et Set *)
    📄 PROPERTY xQ : BOOL (* sortie Q : accesseur Get *)
    📄 PROPERTY tET : TIME (* Elapsed Time : accesseur Get *)
    📄 METHOD ABSTRACT MTimer;
```

```

FUNCTION_BLOCK ABSTRACT CTimer IMPLEMENTS ITF_ObjTimer
VAR
    m_xIn      : BOOL;
    m_tPT      : TIME ;
    m_xQ       : BOOL;
    m_tET      : TIME ;
END_VAR
```



## ❑ ABSTRACT

- Comme les **trois types de temporisateurs** proposent tous la même **interface**, on peut définir une **classe abstraite CTimer** dont le rôle est de **fournir un concept** pour le développement d'objets temporisateur.

```

🔑 INTERFACE ITF_ObjTimer (* contrat pour la Classe CTimer *)
    PROPERTY xIN : BOOL (* entrée IN : accesseurs Get et Set *)
    PROPERTY tPT : TIME (* Preset Time : accesseurs Get et Set *)
    PROPERTY xQ : BOOL (* sortie Q : accesseur Get *)
    PROPERTY tET : TIME (* Elapsed Time : accesseur Get *)
    METHOD ABSTRACT MTimer;
```

```

FUNCTION_BLOCK ABSTRACT CTimer IMPLEMENTS ITF_ObjTimer
```

```

VAR
```

```
    m_xIn      : BOOL;
```

```
    m_tPT      : TIME ;
```

```
    m_xQ       : BOOL;
```

```
    m_tET      : TIME ;
```

```

END_VAR
```

```

THIS^.MTimer();
```

## □ ABSTRACT

 **CTimer.FB\_Init** (\* constructeur pour la Classe CTimer \*)

**METHOD FB\_Init** : **BOOL**

**VAR\_INPUT**

    bInitRetains               : **BOOL**;

    bInCopyCode             : **BOOL**;

**END\_VAR**

.....

.....

.....

.....

.....

 **CTimer**.....

.....

 **CTimer**.....

.....

 **CTimer**.....

.....

 **CTimer**.....

.....

 **CTimer**.....

.....

 **CTimer**.....

.....

## ❏ ABSTRACT

 **CTimer.FB\_Init** (\* constructeur pour la Classe CTimer \*)

### METHOD *FB\_Init* : *BOOL*

## VAR\_INPUT

**blnitRetains** : **BOOL**;**blnCopyCode** : **BOOL**;

**END\_VAR**

```
THIS^.m_xIN      := FALSE ;
```

```
THIS^.m_tPT      := T#1s ;
```

```
THIS^.m_xQ      := FALSE ;
```

```
THIS^.m_tET      := T#0s ;
```

```
FB_Init := TRUE;
```

 **CTimer**.....

 **CTimer.....**

 **CTimer**.....

 **CTimer.....**

## CTimer.....

## CTimer.....

## □ ABSTRACT

 **CTimer.FB\_Init** (\* constructeur pour la Classe CTimer \*)

**METHOD FB\_Init** : **BOOL**

**VAR\_INPUT**

    bInitRetains               : **BOOL**;

    bInCopyCode               : **BOOL**;

**END\_VAR**

**THIS**^.m\_xIN                := **FALSE** ;

**THIS**^.m\_tPT                := T#1s ;

**THIS**^.m\_xQ                 := **FALSE** ;

**THIS**^.m\_tET                := T#0s ;

**FB\_Init** := **TRUE**;

 **CTimer.xIN.Get**

    xIN := **THIS**^.m\_xIN ;

 **CTimer.tPT.Get**

    tPT := **THIS**^.m\_tPT ;

 **CTimer.xQ.Get**

    xQ := **THIS**^.m\_xQ ;

 **CTimer.tET.Get**

    tET := **THIS**^.m\_tEt;

 **CTimer.xIN.Set**

**THIS**^.m\_xIN := xIN ;

 **CTimer.tPT.Set**

**THIS**^.m\_tPT := tPT;



```
FUNCTION_BLOCK CTON EXTENDS CTimer
VAR
    InstTON : standard.TON ;
END_VAR
```

```
THIS^.MTimer();
```

```

MTimer ()
    InstTON (          IN      := ..... ,
                    PT      := ..... ,
                    Q       := ..... ,
                    ET      := ..... );
```

```
FUNCTION_BLOCK CTOF EXTENDS CTimer
VAR
    InstTOF : standard.TOF ;
END_VAR
```

```
THIS^.MTimer();
```

```

MTimer ()
    InstTOF (          IN      := ..... ,
                    PT      := ..... ,
                    Q       := ..... ,
                    ET      := ..... );
```

```
FUNCTION_BLOCK CTon EXTENDS CTimer
VAR
    InstTON : standard.TON ;
END_VAR
```

```
THIS^.MTimer();
```

 **MTimer ()**

```
    InstTON (          IN      := THIS^.m_xIN,
                      PT      := THIS^.m_tPT,
                      Q       := THIS^.m_xQ,
                      ET      := THIS^.m_tE );
```

```
FUNCTION_BLOCK CTOF EXTENDS CTimer
VAR
    InstTOF : standard.TOF ;
END_VAR
```

```
THIS^.MTimer();
```

 **MTimer ()**

```
    InstTOF (          IN      := .....,
                      PT      := .....,
                      Q       := .....,
                      ET      := ..... );
```

```
FUNCTION_BLOCK CTON EXTENDS CTimer
VAR
    InstTON : standard.TON ;
END_VAR
```

```
THIS^.MTimer();
```

 **MTimer ()**

```
    InstTON (          IN      := THIS^.m_xIN,
                      PT      := THIS^.m_tPT,
                      Q       := THIS^.m_xQ,
                      ET      := THIS^.m_tE );
```

```
FUNCTION_BLOCK CTOF EXTENDS CTimer
VAR
    InstTOF : standard.TOF;
END_VAR
```

```
THIS^.MTimer();
```

 **MTimer ()**

```
    InstTOF (          IN      := THIS^.m_xIN,
                      PT      := THIS^.m_tPT,
                      Q       := THIS^.m_xQ,
                      ET      := THIS^.m_tE );
```