# Utilisation d’expressions lambda en 10 étapes

## Sources

* Utilisation d’expressions lambda en 10 étapes (<http://blog.pagesd.info/post/2011/10/18/utilisation-expressions-lambda-en-10-etapes>)
* Introduction au délégués en C# par Fabien Guillot (<http://fguillot.developpez.com/cours/dotnet/introduction-delegates-csharp/>)

## Point de départ : 3 fonctions avec du code répétitif ( !DRY)

### PrevisionFacturationSrv.cs

PrevSrv.UpdateEndOfContract(PrevFinContrat);

PrevSrv.UpdateEndOfMonth(PrevFinMois);

PrevSrv.UpdatePlacement(PrevPlacement, PonderationPlacement);

### PrevisionSrv.cs

/// <summary>

/// Màj de la prévision de facturation fin de contrat pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

public void UpdateEndOfContract(IList<PrevisionFacturation> list)

{

foreach (var pf in list)

{

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

**Prevision.FinContrat = pf.Montant < 0 ? 0 : Math.Round(pf.Montant, 2);**

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Màj de la prévision de facturation fin de mois pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

public void UpdateEndOfMonth(IList<PrevisionFacturation> list)

{

foreach (var pf in list)

{

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

**Prevision.FinMois = pf.Montant < 0 ? 0 : Math.Round(pf.Montant, 2);**

**Prevision.NbContrats = pf.Nombre;**

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Màj de la prévision de facturation placement pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="ponderation">Taux de pondération pour la prise en compte du placement

(style 10, 10, 25, 100)</param>

public void UpdatePlacement(IList<PrevisionFacturation> list, int ponderation)

{

foreach (var pf in list)

{

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

**Prevision.Placement = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);**

this.CreateOrUpdate(Prevision);

}

}

## 1° étape : Mise en commun du code (ancien style)

### PrevisionFacturationSrv.cs

PrevSrv.UpdatePrevision(PrevFinContrat, "FinContrat", 0);

PrevSrv.UpdatePrevision(PrevFinMois, "FinMois", 0);

PrevSrv.UpdatePrevision(PrevPlacement, "Placement", PonderationPlacement);

### PrevisionSrv.cs

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="type">Type prévision à traiter (FinContrat / FinMois / Placement)</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (0 en général

et 0 à 100 pour Placement)</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, string type, int ponderation)

{

foreach (var pf in list)

{

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

switch (type)

{

case "FinContrat":

**Prevision.FinContrat = pf.Montant < 0 ? 0 : Math.Round(pf.Montant, 2);**

break;

case "FinMois":

**Prevision.FinMois = pf.Montant < 0 ? 0 : Math.Round(pf.Montant, 2);**

**Prevision.NbContrats = pf.Nombre;**

break;

case "Placement":

**Prevision.Placement = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);**

break;

}

this.CreateOrUpdate(Prevision);

}

}

## 2° étape : Isolation des traitements spécifiques dans des fonctions séparées

### PrevisionFacturationSrv.cs

PrevSrv.UpdatePrevision(PrevFinContrat, "FinContrat", 100);

PrevSrv.UpdatePrevision(PrevFinMois, "FinMois", 100);

PrevSrv.UpdatePrevision(PrevPlacement, "Placement", PonderationPlacement);

### PrevisionSrv.cs

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="type">Type prévision à traiter (FinContrat / FinMois / Placement)</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, string type, int ponderation)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

switch (type)

{

case "FinContrat":

**UpdateEndOfContract(ref Prevision, pf.Montant, 0);**

break;

case "FinMois":

**UpdateEndOfMonth(ref Prevision, pf.Montant, pf.Nombre);**

break;

case "Placement":

**UpdatePlacement(ref Prevision, pf.Montant, 0);**

break;

}

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Isole les affectations spéciques à la màj de la prévision de facture fin de mois

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public void UpdateEndOfContract(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinContrat = montant;

}

/// <summary>

/// Isole les affectations spéciques à la màj de la prévision de facture fin de contrat

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public void UpdateEndOfMonth(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinMois = montant;

prevision.NbContrats = nombre;

}

/// <summary>

/// Isole les affectations spéciques à la màj de la prévision de facture placement

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public void UpdatePlacement(ref Prevision prevision, decimal montant, int nombre)

{

prevision.Placement = montant;

}

## 3° étape : Mise en place des délégués (enfin !)

### PrevisionFacturationSrv.cs

**PrevSrv.UpdateAssign assign;**

**assign = new PrevSrv.UpdateAssign(PrevSrv.UpdateEndOfContract);**

PrevSrv.UpdatePrevision(PrevFinContrat, 100, **assign**);

assign = new PrevSrv.UpdateAssign(PrevSrv.UpdateEndOfMonth);

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

assign = new PrevSrv.UpdateAssign(PrevSrv.UpdatePlacement);

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, assign);

### PrevisionSrv.cs

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

/// <param name="assign">Méthode pour réaliser le traitement spécifique</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, int ponderation, UpdateAssign assign)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

**assign(ref Prevision, pf.Montant, pf.Nombre);**

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Signature des fonctions d'affectation spécifiques

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

**public delegate void UpdateAssign(ref Prevision prevision, decimal montant, int nombre);**

public void UpdateEndOfContract(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinContrat = montant;

}

public void UpdateEndOfMonth(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinMois = montant;

prevision.NbContrats = nombre;

}

public void UpdatePlacement(ref Prevision prevision, decimal montant, int nombre)

{

prevision.Placement = montant;

}8

## 4° étape : Simplification des délégués grâce à l’inférence de type

### PrevisionFacturationSrv.cs

PrevSrv.UpdatePrevision(PrevFinContrat, 100, **PrevSrv.UpdateEndOfContract**);

PrevSrv.UpdatePrevision(PrevFinMois, 100, **PrevSrv.UpdateEndOfMonth**);

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, **PrevSrv.UpdatePlacement**);

### PrevisionSrv.cs : inchangé

## 5° étape : Arrivée des méthodes anonymes (avec .NET 2)

### PrevisionFacturationSrv.cs

PrevSrv.UpdateAssign assign;

assign = new PrevSrv.UpdateAssign(delegate(ref Prevision prevision, decimal montant, int nombre)

{

**prevision.FinContrat = montant;**

});

PrevSrv.UpdatePrevision(PrevFinContrat, 100, assign);

assign = new PrevSrv.UpdateAssign(delegate(ref Prevision prevision, decimal montant, int nombre)

{

**prevision.FinMois = montant;**

**prevision.NbContrats = nombre;**

});

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

assign = new PrevSrv.UpdateAssign(delegate(ref Prevision prevision, decimal montant, int nombre)

{

**prevision.Placement = montant;**

});

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, assign);

### PrevisionSrv.cs

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

/// <param name="assign">Méthode pour réaliser le traitement spécifique</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, int ponderation, UpdateAssign assign)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

assign(ref Prevision, pf.Montant, pf.Nombre);

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Signature des fonctions d'affectation spécifiques

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public delegate void UpdateAssign(ref Prevision prevision, decimal montant, int nombre);

**// FINI !!!!!!!!!!!!!!!!!!!!**

## 6° étape : Simplification des délégués anonymes grâce à l’inférence de type

### PrevisionFacturationSrv.cs

PrevSrv.UpdateAssign assign;

assign = delegate(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinContrat = montant;

};

PrevSrv.UpdatePrevision(PrevFinContrat, 100, assign);

assign = delegate(ref Prevision prevision, decimal montant, int nombre)

{

prevision.FinMois = montant;

prevision.NbContrats = nombre;

};

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

assign = delegate(ref Prevision prevision, decimal montant, int nombre)

{

prevision.Placement = montant;

};

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, assign);

### PrevisionSrv.cs : inchangé

## 7° étape : Retour sur le code : « ref » ne sert à rien (merci Nicolas)

### PrevisionFacturationSrv.cs (une fois le mot clé 'ref' supprimé)

PrevSrv.UpdateAssign assign;

assign = delegate(Prevision prevision, decimal montant, int nombre)

{

prevision.FinContrat = montant;

};

PrevSrv.UpdatePrevision(PrevFinContrat, 100, assign);

assign = delegate(Prevision prevision, decimal montant, int nombre)

{

prevision.FinMois = montant;

prevision.NbContrats = nombre;

};

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

assign = delegate(Prevision prevision, decimal montant, int nombre)

{

prevision.Placement = montant;

};

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, assign);

### PrevisionSrv.cs (une fois le mot clé 'ref' supprimé)

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

/// <param name="assign">Méthode pour réaliser le traitement spécifique</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, int ponderation, UpdateAssign assign)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

assign(Prevision, pf.Montant, pf.Nombre);

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Signature des fonctions d'affectation spécifiques

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public delegate void UpdateAssign(Prevision prevision, decimal montant, int nombre);

## 8° étape : Arrivée des expressions lambdas (avec .NET 3.5)

### PrevisionFacturationSrv.cs

* Suppression du mot clé « delegate » avant la liste des paramètres
* Ajout de l’opérateur « => » après la liste des paramètres

PrevSrv.UpdateAssign assign;

assign = (Prevision prevision, decimal montant, int nombre) =>

{

prevision.FinContrat = montant;

};

PrevSrv.UpdatePrevision(PrevFinContrat, 100, assign);

assign = (Prevision prevision, decimal montant, int nombre) =>

{

prevision.FinMois = montant;

prevision.NbContrats = nombre;

};

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

assign = (Prevision prevision, decimal montant, int nombre) =>

{

prevision.Placement = montant;

};

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, assign);

### PrevisionSrv.cs : inchangé

## 9° étape : Simplification des expressions lambdas

### PrevisionFacturationSrv.cs (pas à pas)

PrevSrv.UpdateAssign assign;

assign = (prevision, montant, nombre) =>

{

prevision.FinContrat = montant;

};

PrevSrv.UpdatePrevision(PrevFinContrat, 100, assign);

assign = (p, m, n) =>

{

p.FinMois = m;

p.NbContrats = n;

};

PrevSrv.UpdatePrevision(PrevFinMois, 100, assign);

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, (p, m, n) => { p.Placement = m; });

### PrevisionFacturationSrv.cs (définitif)

PrevSrv.UpdatePrevision(PrevFinContrat, 100, (p, m, n) => { p.FinContrat = m; });

PrevSrv.UpdatePrevision(PrevFinMois, 100, (p, m, n) => { p.FinMois = m; p.NbContrats = n; });

PrevSrv.UpdatePrevision(PrevPlacement, PonderationPlacement, (p, m, n) => { p.Placement = m; });

### PrevisionSrv.cs : inchangé

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

/// <param name="assign">Méthode pour réaliser le traitement spécifique</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, int ponderation, UpdateAssign assign)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

assign(Prevision, pf.Montant, pf.Nombre);

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Signature des fonctions d'affectation spécifiques

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public delegate void UpdateAssign(Prevision prevision, decimal montant, int nombre);

## 10° étape : Utilisation de paramètres optionnels

### PrevisionFacturationSrv.cs

PrevSrv.UpdatePrevision(PrevFinContrat, (p, m, n) => { p.FinContrat = m; });

PrevSrv.UpdatePrevision(PrevFinMois, (p, m, n) => { p.FinMois = m; p.NbContrats = n; });

PrevSrv.UpdatePrevision(PrevPlacement, (p, m, n) => { p.Placement = m; }, PonderationPlacement);

### PrevisionSrv.cs

/// <summary>

/// Mise à jour de la prévision de facturation pour un ensemble de Siren / Société

/// </summary>

/// <param name="list">Liste des couples Siren / Société et des montants à traiter</param>

/// <param name="assign">Méthode pour réaliser le traitement spécifique</param>

/// <param name="ponderation">Taux de pondération pour le type de prévision (100 en général

et 0 à 100 pour Placement)</param>

public void UpdatePrevision(IList<PrevisionFacturation> list, UpdateAssign assign, **int ponderation = 100**)

{

foreach (var pf in list)

{

pf.Montant = pf.Montant < 0 ? 0 : Math.Round(pf.Montant \* ponderation / 100, 2);

var Prevision = this.GetBySirenAndSociety(pf.Siren, pf.Societe);

assign(Prevision, pf.Montant, pf.Nombre);

this.CreateOrUpdate(Prevision);

}

}

/// <summary>

/// Signature des fonctions d'affectation spécifiques

/// </summary>

/// <param name="prevision">Objet Prevision à mettre à jour</param>

/// <param name="montant">Montant de la prévision</param>

/// <param name="nombre">Nombre de contrats traités (ssi fin de mois)</param>

public delegate void UpdateAssign(Prevision prevision, decimal montant, int nombre);