Hola Alexis.
Et responc a la teva pregunta sobre validació de dades.

Per tenir validació de dades, comprovar-les o restringir-les, en temps real al introduir valors o quan es vol manipular el formulari complert com quan es vol enregistrar, es comproven les dades en diferents nivells.

Nivell de controls a la vista.
Una primera restricció es utilitzar les propietats de cada control que permeten indicar per exemple un màxim numero de caràcters com es veu al passwordbox de la vista d'Identificació. Una contrasenya no pot ser de mes de 12 caràcters.

<PasswordBox Name="Clau **MaxLength="12"** />

Nivell comportaments "Behaviours" de controls a la vista.
Jo no els he fet servir però es poden definir unes classes auxiliars amb comportaments que s'enllacen als controls i restringeixen per exemple quins caràcters son admesos o converteixen les dades introduïdes a unes altres.

Nivell de Propietats enllaçades al ViewModel
Mostraré trossos de codi corresponents al MVVM de la interfície de creació de Tècnics al meu sistema.

Quan es te desacoblada la lògica del disseny com es fa amb el patró MVVM tenim el codi XAML amb la descripció de la vista *TecnicView.xaml*, el code behind mínim associat a la vista *TecnicView.cs* i la lògica al ViewModel *TecnicViewModel.cs*

Habitualment el ViewModel ha d'implementar la interfície ***INotifyPropertyChanged*** per tal que els controls de la vista i les propietats del ViewModel es comuniquin mútuament els canvis a les dades enllaçades.
Mes informació al següent enllaç.
http://msdn.microsoft.com/en-us/library/ms743695(v=vs.100).aspx

També s'ha de ser explícit al fer l'enllaç, *binding*, de les propietats del control amb les del ViewModel com es veu al següent exemple

<TextBox Name="Nom" **Text="{Binding Nom, Mode=TwoWay, ValidatesOnDataErrors=True}"** />

Però fent servir el framework per MVVM ***Caliburn Micro*** tota aquesta feina queda facilitada. Les classes bases per implementar ViewModel ja implementen ***INotifyPropertyChanged*** i el *binding* es fa automàticament si els noms del control a la vista i la propietat al ViewModel coincideixen. El mateix *binding* automàtic passa amb els controls enllaçats a comandaments.

L'exemple anterior quedaria:

<TextBox Name="Nom" />

Quan ja es tenen enllaçats el View i el ViewModel per tal de que es puguin validar les dades i es reflecteixi en temps real al View, el ViewModel ha d'implementar la interfície ***IDataErrorInfo.***

Això permet comprovar la validesa de les dades i mantenir un diccionari amb les propietats i els seus missatges d'error. Quan les dades no son vàlides al control apareix un rectangle vermell vorejant-lo.

Jo he afegit addicionalment una propietat general al ViewModel anomenada ***FormErrors*** que em serveix per mostrar o ocultar un rectangle vermell amb el missatge d'error específic.

Nivell d'activació de commandaments

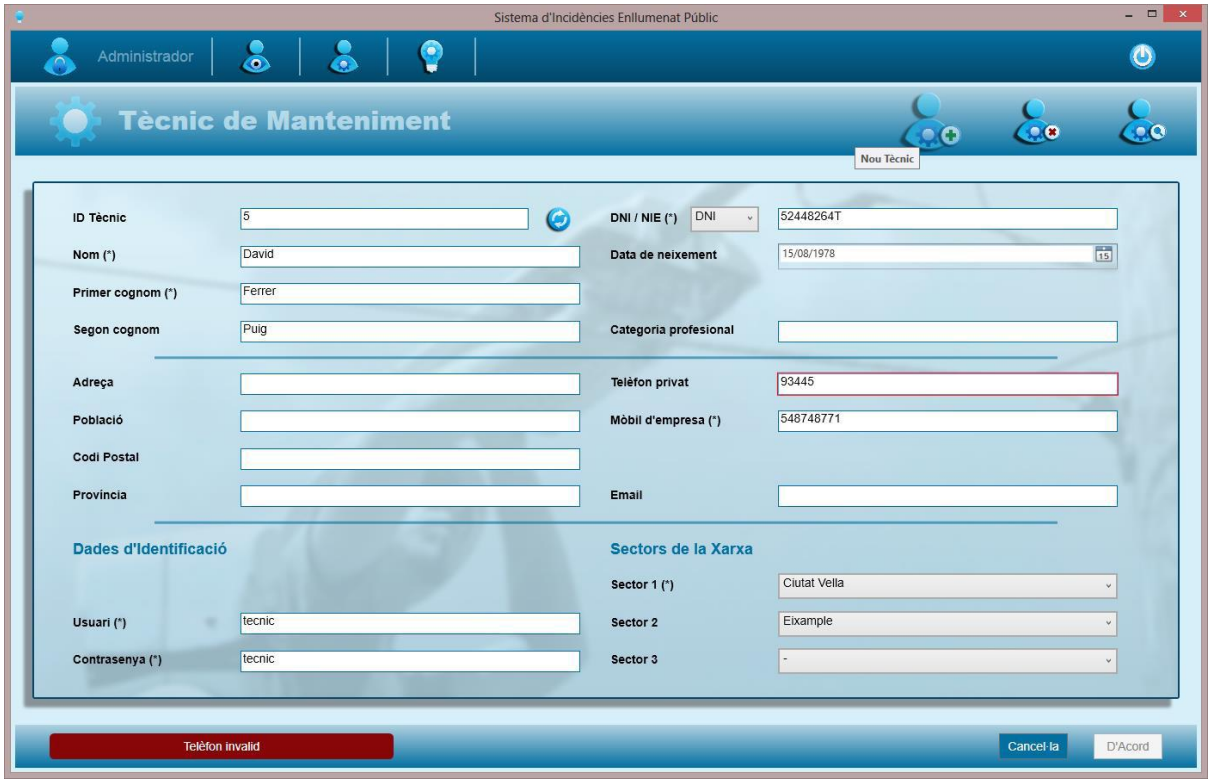
Caliburn Micro proporciona un altre eina molt potent que consisteix en un mètode vinculat a dada comandament que es vol executar. Aquest mètode ***CanNomCommandament*** comprova una expressió booleana que serveix per activar o no el comandament ***NomCommandament*** enllaçat al control corresponent a la Vista. Aquest enllaç com he comentant, es fa automàticament amb Caliburn si es diuen de la mateixa manera.

```
public bool CanSave
{
        get
        {
            bool isValid = !String.IsNullOrEmpty(Nom);
            return isValid;
          }
}

public bool Save
{
…
}
```

EXEMPLE:

A continuació es proporcionen parts importants del codi que implementen la vista del formulari per introduir i validar les dades d'un tècnic al sistema. Les parts mes significatives estan comentades i remarcades.



TecnicView.xaml

<TextBox Name="Nom" />
<Border Name="NotificacioError" Visibility="{Binding Path=FormErrors}" Style="{StaticResource ErrorsBorder}" >

<Button Name="Save" Content="D'Acord" Style="{StaticResource BlueButton}" Width="80" />

TecnicView.cs

```
namespace TFC.PublicLighting.PresentationWPF.Tecnic
{
   public partial class TecnicView : UserControl
   {
      public TecnicView()
      {
         InitializeComponent();
      }
   }
}
```

TecnicViewModel.cs

```
using Caliburn.Micro;
…

public class TecnicViewModel : IDataErrorInfo
   {
      private string _formErrors;
      private string _lastError

     // MODEL de dades del ViewModel
      private UsuariDTO _tecnicModel;

…

# region Error
      public string FormErrors
      {
         get { return _formErrors; }
         set
         {
            _formErrors = value;
            NotifyOfPropertyChange(() => FormErrors);
         }
      }

      public string LastError
      {
         get { return _lastError; }
         set
         {
            _lastError = value;
            NotifyOfPropertyChange(() => LastError);
         }
      }
# endregion

# region Properties
// Model amb les dades que s'enllacen a través del ViewModel amb la vista
public UsuariDTO TecnicModel
      {
         get { return _tecnicModel; }
         set
         {
            _tecnicModel = value;
            NotifyOfPropertyChange(() => TecnicModel);
         }
      }
```

```
// Propietat Nom que s'ha de validar al formulari i es obligatoria
public string Nom
      {
         get { return TecnicModel.Nom; }
         set
         {
            TecnicModel.Nom = value;
          // Notifica que hi han canvas a la propietat i la vista s'actualitza.
           NotifyOfPropertyChange(() => Nom);
          // Notifica al metode CanSave que hi han canvas a una propietat involucrada a la validació.
           NotifyOfPropertyChange(() => CanSave);

          // Fa la validació del contingut del nou valor cada vegada que s'introduiex un carácter.
          if (string.IsNullOrEmpty(value))
          {
              // Modifica el missatge d'error
             LastError = "Es requereix el Nom";
             // Afegeix l'error al diccionari de la implementació de IDataErrorInfo
             _dataErrors["Nom"] = LastError;
          }
          else
          {
                    // Si no hi han errors esborra l'entrada del diccionari d'errors.
                    if (_dataErrors.ContainsKey("Nom"))
                _dataErrors.Remove("Nom");
          }
         }
      }

# endregion


# region Commands

      // Comprova que el comandament Save es pot activar segons les regles de validació.
      // En aquest exemple només es comprova una propietat de tot el formulari.
      public bool CanSave
      {
         get
         {
           bool isValid = !String.IsNullOrEmpty(Nom);

            return isValid;
         }
      }
```

```
// Si CanSave es True els controls enllaçats a Save estàn actius  i  el commandament es pot executar.
public void Save()
      {
         if (TecnicModel.Id == 0)
         {
            if (client.AddTecnic(TecnicModel, CacheMain.CurrentUserId))
            {
               MessageBox.Show("Tècnic enregistrat correctament");
               New();
            }
            else
            {
               MessageBox.Show("Error enregistrant Tècnic");
            }
         }
         else if (TecnicModel.Id > 0)
         {
            if (client.UpdateTecnic(TecnicModel, CacheMain.CurrentUserId))
            {
               MessageBox.Show("Tècnic actualitzat correctament");
               New();
            }
            else
            {
               MessageBox.Show("Error actualitzant Tècnic");
            }
         }
         NotifyOfPropertyChange(LastError);
         NotifyOfPropertyChange(FormErrors);
      }

# endregion

// Implementació del membres de IDataErrorInfo
#region IDataErrorInfo Members

      private string _dataError = string.Empty;

      public string Error
      {
         get { return _dataError; }
      }

// Diccionari amb les propietats amb errors com a clau i un missatge com a valor.
protected Dictionary<string, string> _dataErrors = new Dictionary<string, string>();

      public string this[string columnName]
```

```
{
   get
   {
      //Activa o desactiva el missatge general d'errors
      if (_dataErrors.Count != 0)
      {
         FormErrors = "Visible";
         NotifyOfPropertyChange(() => FormErrors);
      }
      else
      {
         FormErrors = "Collapsed";
        NotifyOfPropertyChange(() => FormErrors);
      }

      if (_dataErrors.ContainsKey(columnName))
      {
         return _dataErrors[columnName];
      }

      else
         return null;
   }
}

#endregion
```