**Eventy, EventHandler**

# Normovadlo Dobes – jak vyhodit event a poslouchat ho v nesouvisici tride pomoci registrace v Castlu

<https://tfs.mediaresearch.cz/AdIntel/AdIntel/Ad%20Intel%20Developers/_git/AdIntel/commit/2d65aaa9d9550858ff36e8da6d641a9dfe864965>

**Slo o to, aby reclamationViewModel refreshnul plausibilitu v celem kodovadle (zmeni se brush backgroundu).**

**V Kodovadle existuji komponenty ktere pozaduji MessageReload. Pridame k nim i reclamationViewModel. Tady vyjmenujeme, ktere komponenty chceme mit v kolekci componentsRequiringMessageReload.**

Component.For<ICodingMode>().ImplementedBy<ProductionCodingModeViewModel>().Named(ProductionCodingModeInstallerComponents.ProductionCodingModeViewModel)

.DependsOn(

Dependency.OnComponentCollection("componentsRequiringMessageReload", ProductionCodingModeInstallerComponents.VideoDuplicityHunter, ReclamationInstallerComponents.ReclamationViewModel),

**ReclamationViewModel musi implementovat rozhrani IRequiresMessageReload:**

public class ReclamationViewModel : ScreenWithIcon, ICodingComponent, IAcceptsActualMessageChanged, IDisableAdviser, IVisible, ISupportProxy, IRequiresMessageReload

**Na rozhrani pridal EventHandler**

public interface IRequiresMessageReload

{

bool RequiresReload { get; }

void UpdateLoadingArgs(ref ActualMessageIdChangedEventArgs args);

event EventHandler ReloadRequested;

}

**V ReclamationViewModelu musel toto rozhrani implementovat (I metodu kterou nepotrebuje. Tu nechal prazdnou) V miste, kde chce notifikovat zavola metodu OnReloadRequested, ve ktere vyhodi event ReloadRequested.**

public void UpdateLoadingArgs(ref ActualMessageIdChangedEventArgs args)

{

}

protected void OnReloadRequested()

{

ReloadRequested?.Invoke(this, EventArgs.Empty);

}

public event EventHandler ReloadRequested;

**V konstruktoru ProductionCodingModeViewModelu (VM ktery muze reloadnout message) dostane diky registraci od Castlu kolekci komponent, ktere muzou pozadovat reload message.**

**Jejich event ReloadRequested zaregistruje k metode ktera event obslouzi**

ComponentsRequiringMessageReload = componentsRequiringMessageReload.ToList();

ComponentsRequiringMessageReload.ForEach(c => c.ReloadRequested += HandleReloadRequested);

private void HandleReloadRequested(object sender, EventArgs e)

{

if (m\_lastMessageEventArgs == null)

{

return;

}

ITask changeMessageTask = new DelegateTask<ActualMessageIdChangedEventArgs>("ChangeMessage", ChangeMessage, m\_lastMessageEventArgs);

m\_taskQueue.EnqueueTask(changeMessageTask);

}

}

}

**All**

# Popis eventu:

**Trida ktera chce event vyhodit ho musi definovat :**

public event EventHandler OnFilterCanExecuteChanged;

**Na vyhazovani eventu si vytvorime metodu:**

private void FireOnFilterCanExecuteChanged()

{

OnFilterCanExecuteChanged?.Invoke(this, null);

}

**Trida ktera chce event poslouchat (pravdepodobne jiny ViewModel) se musi zaregistrovat jako posluchac:**

MotivletFilters.OnFilterCanExecuteChanged += OnFilterCanExecuteChanged;

**Trida ktera chce event poslouchat musi mit metodu ktera se provede kdyz sem dorazi event:**

private void OnFilterCanExecuteChanged(object sender, EventArgs e)

{

NewValuesFilters.ButtonGroupCanExecute = MotivletFilters.Filters.Any(d => d.SelectedValues.Any()) && NewValuesFilters.Filters.Any(d => d.SelectedValues.Any()) && ChangingInProgress == false;

}

**Pri likvidaci teto tridy se musi posluchac odregistrovat :**

#region IDispose Implementation

private bool m\_disposed;

public virtual void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!m\_disposed)

{

if (disposing)

{

}

MotivletFilters.OnFilterCanExecuteChanged -= OnFilterCanExecuteChanged;

}

m\_disposed = true;

}

#endregion

EventHandler <genericky>

**Slouzi na predani objektu jako EventArgs**

**Chci predat napr boolean**

**Trida ktera vyhazuje ma genericky event**

public event EventHandler<bool> ChangingInProgressChanged;

**a vyhazovaci metodu s parametrem:**

private void FireChangingInProgressChanged(bool parameter)

{

ChangingInProgressChanged?.Invoke(this, parameter);

}

**Trida ktera chce byt posluchacem se registruje**

DataViewModel.ChangingInProgressChanged += ChangingInProgress;

**A ma metodu s parametem ktera se provede kdyz sem event dorazi:**

private void ChangingInProgress(object sender, bool e)

{

Filters.ChangingInProgress = e;

}

**Odregistrace pri likvidaci tridy:**

#region IDispose Implementation

private bool m\_disposed;

public virtual void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!m\_disposed)

{

if (disposing)

{

}

DataViewModel.ChangingInProgressChanged -= ChangingInProgress;

}

m\_disposed = true;

}

#endregion