Team Build in TFS2010 aan wensen aanpassen

Van laptop tot een gedistribueerde omgeving

Ewald Hofman

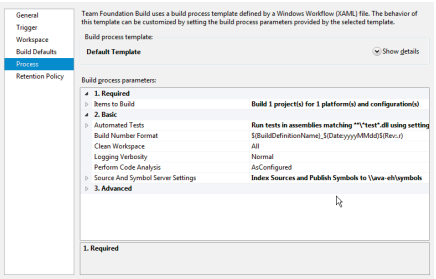
Team Foundation Server 2010 ALM heeft een aantal mooie verbeteringen op het gebied van architectuur, work items, reporting en testen. Ook de Team Build, heeft een facelift ondergaan. Dit artikel beschrijft hoe de nieuwe build werkt en hoe je het kan aanpassen aan je eigen wensen.

Wie Team Build uit TFS 2008 kent, zal zien dat Microsoft veel heeft gewijzigd in de architectuur van Team Build in TFS 2010. De componenten waar je in TFS 2010 mee te maken krijgt zijn: Project Collection, Build Controller en Build Agent.

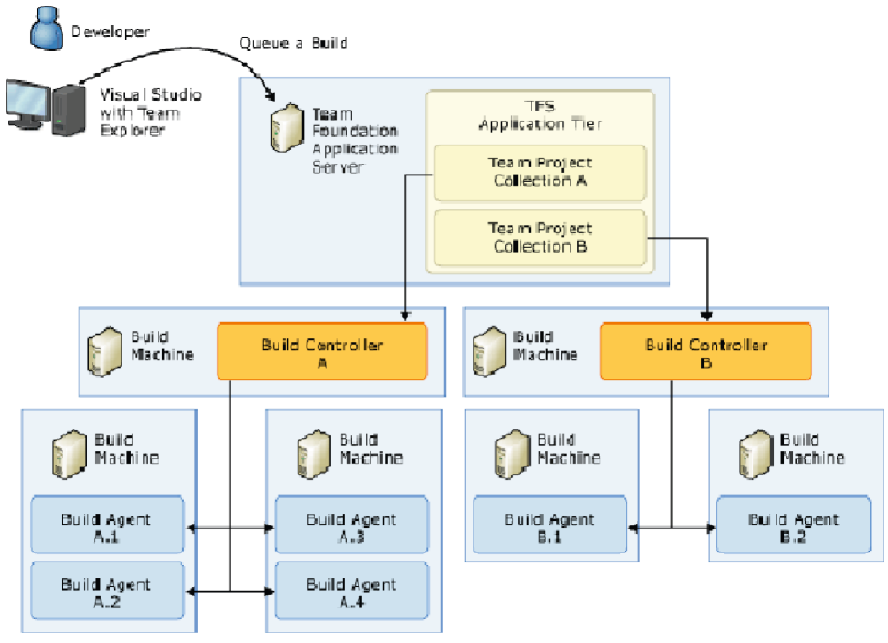
De eerste grote wijziging is dat TFS 2010 nu Project Collections kent, waarmee je meerdere geïsoleerde TFS instanties op één machine kunt hosten. Voor elke Project Collection heb je een Build Controller nodig die het werk toekent aan een of meerdere Build Agents. Een Build Agent is te vergelijken met de Build Agent zoals je die ook in TFS 2008 kent en voert het werk uit, zoals compileren en unit testen. Een groot voordeel van deze opzet is dat de build topologie eenvoudiger in te richten en te beheren is. Daarnaast schaalt het heel goed; van kleine teams die alle componenten op een server hebben tot compleet gedistribueerde omgevingen voor grote ondernemingen (figuur 1).

Workflow Foundation 4.0

Een andere grote wijziging waar je mee te krijgen hebt is de syntax waarmee je een build script definieert. In TFS 2010 maakt Team Build gebruik van Windows Workflow Foundation, waar je met Activities taken sequentieel en parallel kunt uitvoeren. In een build definitie geef je aan welke Windows Workflow Foundation template je wilt gebruiken en vult vervolgens de variabelen die in de template zijn gedefinieerd in (figuur 2).



FIGUUR 2



FIGUUR 1

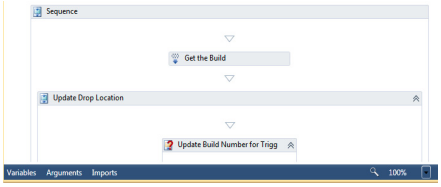
Standaard krijg je drie Build process templates meegeleverd, te weten:

- **Default Template**: de standaard template voor TFS 2010
- **Upgrade Template**: de template voor gemigreerde TFS 2008 builds
- **LabDefaultTemplate**: de template die je kunt gebruiken als je met Lab Management werkt (als je hier meer over wilt weten, hou dan het volgende nummer van het .Net magazine in de gaten)

Als je een nieuwe build aanmaakt met de Default Template, dan voert het de standaard taken uit zoals compileren, valideren met Code Analysis, uitvoeren van testen en het kopiëren naar de Drop location. Indien je wilt dat er nog andere taken worden uitgevoerd, kun je de template aanpassen naar eigen behoefte.

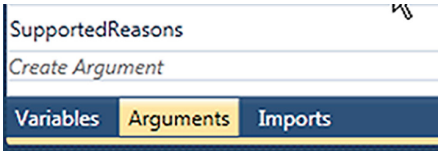
Argumenten en variabelen

De Build Process Template is een .xaml-bestand en kan je vinden in Version Control onder de map BuildProcessTemplates. Als je deze xaml opent in Visual Studio 2010 krijg je een workflow designer als in figuur 3 te zien.



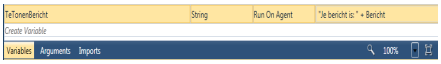
FIGUUR 3

Hierin kan je bijvoorbeeld variabelen en argumenten toevoegen of aanpassen. In figuur 3 zie je onderin de donkerblauwe balk de Variables en Arguments staan. Via een argument moet de tekst 'Je bericht is: XXXX' in de build output te zien zijn. Je maakt een nieuw argument 'Bericht' aan door op de Arguments te klikken en naar beneden te scrollen (figuur 4).



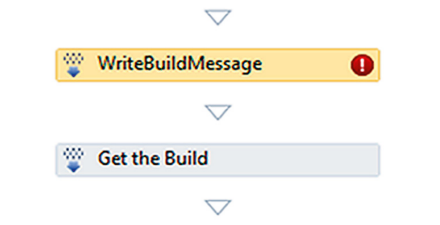
FIGUUR 4

Klik je op de activity genaamd Run on Agent en voeg een nieuwe variabele 'TeTonenBericht' toe, met als Default 'Je bericht is:' + Bericht (figuur 5). In deze variabele heb je nu het argument Bericht gebruikt.



FIGUUR 5

Sleep vervolgens de WriteBuildMessage Activity vanuit de toolbox naar de workflow, boven de 'Get the build' activity (figuur 6)

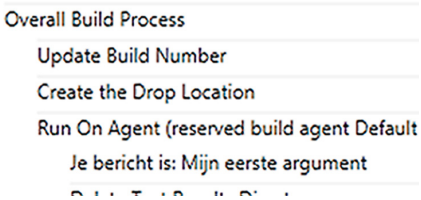


FIGUUR 6

De activiteit toont een rood uitroepteken, omdat je nog niet alle verplichte eigenschappen hebt gevuld. Geef bij de Message property de variabele TeTonenBericht op die we zojuist hebben aangemaakt. Wijzig ook de importance naar High, anders toont de build dit bericht niet. Check je de workflow in om de wijziging in de build definitie te kunnen zien.

Vul bij Bericht de waarde 'Mijn eerste argument in' en start een nieuwe build. Na de build ga je naar de View log en daar zie je de details van de build en je eigen berichtje tussen staan (figuur 7). In de eigenschappen van de Build Process Template kan je het detailnivo van logmeldingen aanpassen. Dit is met name handig als een build fouten geeft en je wilt onderzoeken waar het fout gaat.

> In de eigenschappen van de Build Process Template kun je het detailniveau van logmeldingen aanpassen.



FIGUUR 7

Custom Activities

Je hebt nu een eenvoudige activiteit toegevoegd aan de build. Visual Studio biedt standaard een aantal activiteiten in de toolbox aan, maar die kan je ook zelf maken. We gaan nu zelf je eigen types maken voor de argumenten en hoe je een eigen Activity kunt definiëren om een command uit te kunnen voeren onder een andere user account.

Om een custom activity te kunnen maken, maak je een nieuwe Visual Studio solution met twee Class Library projecten. Het eerste project noem je BuildProcessTemplateHost en gaat het xaml bestand bevatten waaraan je zojuist het bericht hebt toegevoegd. Het tweede project heet CustomActivities en gaat de nieuwe Activity bevatten. Voeg de referenties toe aan de projecten. (MTF is de afkorting voor de namespace Microsoft.TeamFoundation)

BuildProcessTemplateHost

- CustomActivities <het andere project>
- MTF.Build.Client.dll [1]
- MTF.VersionControl.Client.dll [1]
- MTF.WorkItemTracking.Client.dll [1]
- MTF.Build.Workflow.dll [2]
- MTF.TestImpact.BuildIntegration.dll [2]
- MTF.TestImpact.Client.dll [3]
- System.Activities
- System.Drawing
- System.ServiceModel
- System.ServiceModel.Activities
- System.Xaml

CustomActivities

- MTF.Build.Client.dll [1]
- System.Activities

[1] c:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ReferenceAssemblies\v2.0

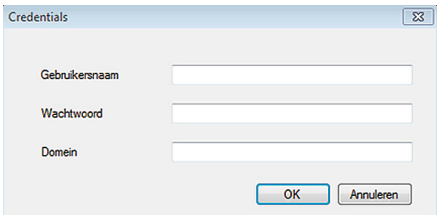
[2] C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies

[3] C:\Windows\assembly\GAC_MSIL\Microsoft.TeamFoundation.TestImpact.Client\10.0.0.0__b03f5f7f11d50a3a

Kopieer de DefaultTemplate.xaml naar de directory van het BuildProcessTemplateHost projecten voeg het toe aan het project. Zet de Build Action van DefaultTemplate.xaml naar XamlAppDef. Je hebt nu het framework klaar waaraan je de custom activities aan kunt toevoegen.

Voeg een Code Activity toe aan je CustomActivities project met de naam 'Impersonate'. Bij het toevoegen van de Activity WriteBuildMessage had je de mogelijkheid om een aantal properties te vullen. Als je de Impersonate Activity toe voegt, krijg je een Text property te zien. De Text property is alleen een placeholder om aan te geven welke properties er zijn en heeft geen waarde tijdens het uitvoeren van de Execute methode. Je kunt met de context.GetValue(<placeholder>) de waarde ophalen voor de property.

Voor het uitvoeren van de Impersonate hebben we een aantal gegevens nodig, namelijk de command die je wilt uitvoeren en de credentials waaronder dat gebeurt. Als je de credentials als drie Text argumenten zou definiëren, dan is het wachtwoord



FIGUUR 8

voor iedereen zichtbaar. Beter kan je hiervoor een nieuw type definiëren en een eigen User Interface maken.
Hiervoor voeg je een nieuwe Class genaamd Credentials toe aan het CustomActivities project met de volgende code:

```
public class Credentials
{
    public string Domain { get; set; }
    public string UserName
    { get; set; }
    public string Password
    { get; set; }
    public override string ToString()
    {
        return UserName;
    }
}
```

Voeg vervolgens een formulier toe genaamd CredentialsForm aan het project wat er ongeveer als in figuur 8 uit ziet, waarbij je voor de Password een MaskedTextBox control gebruikt.

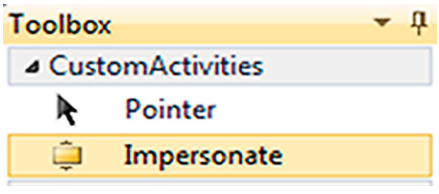
En voeg ook een nieuwe CredentialsEditor class toe. Deze class vertelt de workflow editor wat het moet doen als je een argument van het type Credentials wijzigt. Het komt er op neer dat je erft van de UITypeEditor en de EditValue methode overschrijft. In deze methode definieer je dat je de CredentialsForm wilt gebruiken om de waarde te wijzigen. In de GetEditStyle methode geef je aan dat het formulier als een dialoog moet worden geopend.

```
class CredentialsEditor : UITypeEditor
{

    public override object EditValue(I
    TypeDescriptorContext context,
    IServiceProvider provider, object
    value)
    {
        var editorService = (IWindows-
        FormsEditorService)provider.
        GetService(typeof(IWindows-
        FormsEditorService));

        if (editorService != null)
        {
            var credential = value as
            Credentials;

            using (var dialog = new
            CredentialsForm())
            {
                dialog.UserName =
                credential.UserName;
                dialog.Password =
                credential.Password;
                dialog.Domain =
                credential.Domain;

                if (editorService.
                ShowDialog(dialog) ==
                DialogResult.OK)
                {
                    credential.User-
                    Name = dialog.
                    UserName;
                    credential.Pass-
                    word = dialog.
                    Password;
                    credential.
                    Domain = dialog.
                    Domain;
                }
            }
        }

        return value;
    }

    public override UITypeEditorEdit-
    Style GetEditStyle(ITypeDescriptor
    Context context)
    {
        return UITypeEditorEditStyle.
        Modal;
    }
}
```

Nu kun je dit type gebruiken als argument van de Impersonate Activity. De klasse komt er dan als volgt uit te zien. De Activity gebruikt de Impersonation klasse die ervoor zorgt dat je onder andere credentials code kunt uitvoeren.

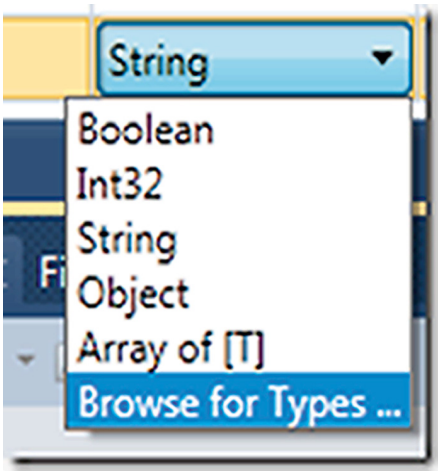
```
public sealed class Impersonate :
CodeActivity
{
    public InArgument<Credentials>
    Credentials { get; set; }
    public InArgument<string> Command
    { get; set; }

    protected override void
    Execute(CodeActivityContext
    context)
    {
        var credentials = context.
        GetValue(this.Credentials);
        var command = context.
        GetValue(this.Command);

        using (new Impersonation
        (credentials))
        {
            Process.Start(command);
        }
    }
}
```

De volgende stap is om de Impersonate activity aan de Build Process Template toe te voegen. Build het CustomActivities project en open daarna de DefaultTemplate.xaml. Na enige tijd zijn alle beschikbare Activities geladen, waaronder de Activity die je zojuist hebt aangemaakt (figuur 9)



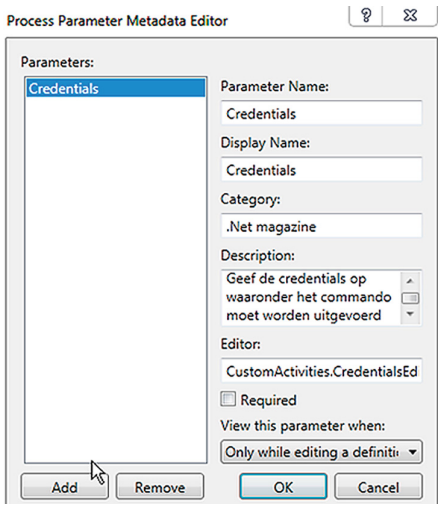
FIGUUR 9

Deze Activity kun je net zoals de WriteBuildMessage naar de workflow slepen. Als je op de Activity gaat staan, dan krijg je in de properties window de command en de credentials argumenten te zijn die je in code hebt definieerd. Je wilt dat de gebruiker van de template zijn eigen gegevens kan invullen . Voeg bij de Arguments een nieuw argument toe van ons eigen Credentials type en geef dit argument door aan de Activity. Dit werkt hetzelfde als je bij de WriteBuildMessage met het argument Bericht hebt gedaan, je geeft alleen een ander type op. Klik op de Argument Type cell en kies voor Browse for Types (figuur 10) en je zoekt de Credentials type op.



FIGUUR 10

Tenslotte vertel je aan de workflow dat het de CredentialsEditor gebruikt, als iemand de waarde van het argument wijzigt. Dat doe je via het Metadata argument , waarmee je het gedrag van alle argumenten kunt bepalen zoals de caption, de groep waartoe het argument toe behoort en de editor die het moet gebruiken. In de veld editor geef je de fullname van de class op



FIGUUR 11

Als je eenmaal door hebt hoe Team Build werkt, zijn de mogelijkheden eindeloos

in het formaat 'CustomActivities.CredentialsEditor,CustomActivities' (figuur 11)

Nu check je de Build Process Template in om de CustomActivities assembly beschikbaar te stellen. Visual Studio kijkt in deze volgorde naar de volgende lokaties voor de Activities:

- De 'Version control Path to custom assemblies' van de buid controller
- De GAC
- De Visual Studio probing paths. Deze paden kun je vinden in de devenv.exe.config en zijn de standaard paden waar alle assemblies staan die tijdens de installatie van Visual Studio zijn neergezet

In ons geval is de Version control path op de build controller het eenvoudigst zodat niet iedereen die de build wilt starten de dll lokaal in de GAC of een van de probing paths hoeft te kopiëren. Wil je de Version control path gebruiken, dien je de assembly toe te voegen aan source control. Open vervolgens het context menu op de Builds node in de Team Explorer en kies je voor 'Manage build controllers'. Kies vervolgens voor de Properties en je krijgt een dialoog te zien met onder andere de optie. Selecteer in dit pad de lokatie waar je de assembly in source control hebt neergezet en accepteer alle wijzigingen.
Daarna kun je een bestaande Build Definition wijzigen of een nieuwe aanmaken. Ga naar de Process tab en klik daar op Show Details. Klik op New en selecteer de DefaultTemplate.xaml die je hebt gewijzigd. Mocht je een foutmelding krijgen dat Visual Studio het Credentials type niet kan laden, start Visual Studio dan opnieuw op. Je ziet nu dat er een Credentials argument is. Vul de Credentials in en je ziet alleen de ingevulde gebruikersnaam, omdat je dat in de ToString() van de Credentials hebt gedefinieerd.

Conclusie

De nieuwe topologie van Team Build zorgt ervoor dat je het kunt installeren op één laptop tot een gedistribueerde omgeving. Gecombineerd met een goed configureerbare en aanpasbare build definitie, is het inzetbaar voor elk type organisatie en voor elk type project.
Als je eenmaal door hebt hoe het werkt, zijn de mogelijkheden eindeloos.

Referenties

http://www.ewaldhofman.nl
http://tfsbuildextensions.codeplex.com

Ewald Hofman, is Solution Developer bij Avanade. Hij is tevens MVP op het gebied van Visual Studio ALM. Hij blogt regelmatig over Team Foundation Server op http://www.ewaldhofman.nl