Planering och programmering av konfigureringsverktyg för mjukvaran Thinking Portfolio

Examensarbete

Informationsteknik

2015

Linus Lindgård

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	5167
Författare:	Linus Lindgård
Arbetets namn:	Planering och programmering av konfigureringsverktyg för mjukvaran Thinking Portfolio
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	AE Business Care Group Oy

Sammandrag:

Thinking Portfolio är ett online-verktyg för strategisk portföljförvaltning av projekt och IT-system. Programmet måste konfigureras enskilt för varje ny kund. Detta examensarbete går igenom planering och programmering av ett verktyg avsett att underlätta konfigureringsarbetet. Målet är att minska på konfigureringstiden, göra konfigureringsarbetet intuitivare och mindre manuellt. Tiden som krävs för konfigurering av en ny portfolio uppskattas bli över 40% kortare med det förbättrade verktyget. Examensarbetets huvudsakliga metod är programvaru-utveckling, men omfattar också dokumentation och tester. Utvecklingsspråk är C#, JavaScript, HTML och CSS. Omgivningen ASP.NET används med Microsoft SQL Server som databas. Verktyget använder sig av biblioteket CodeMirror för färgning av syntax och jQuery UI med insticksmodulen Simone för fönsterhantering.

Nyckelord:	AE Business Care Group Oy, Thinking Portfolio, konfigureringsverktyg, business intelligence, arbetsflödes-optimering, ASP.NET, JavaScript, AJAX
Sidantal:	58
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	5167
Author:	Linus Lindgård
Title:	Planning and programming of a configuration toolkit for the software Thinking Portfolio
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	AE Business Care Group Oy

Abstract:

Thinking Portfolio is an online tool for strategic portfolio management of projects and IT systems. The program must be configured for each new client. This thesis describes the design and programming of a toolkit to support and ease the configuration work. The goals are to reduce the configuration time, make configuration work more intuitive and less manual. The time required for configuration of a new portfolio is estimated to be over 40% shorter with the improved toolkit. The main method of the thesis is software development, but is also documentation and tests. Development languages include C#, JavaScript, HTML and CSS. The environment used is ASP.NET with Microsoft SQL Server as database. The toolkit uses libraries like CodeMirror for syntax highlighting and jQuery UI with the extension Simone for window management.

Keywords:	AE Business Care Group Oy, Thinking Portfolio, configuration toolkit, business intelligence, workflow optimization, ASP.NET, JavaScript, AJAX
Number of pages:	58
Language:	Swedish
Date of acceptance:	

INNEHÅLL

FÖRKORTNINGAR OCH TERMINOLOGI

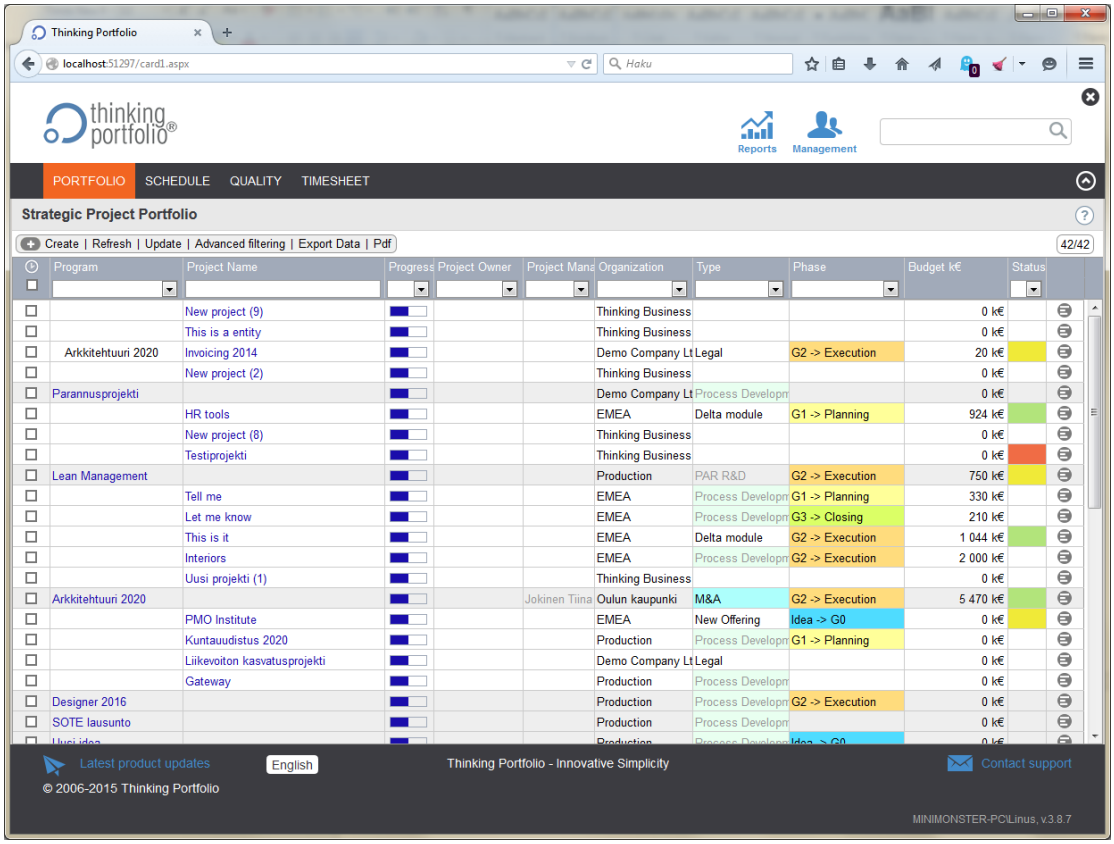
HTML	**H**yper**T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle **S**heets
SQL	**S**tructure **Q**uery **L**anguage
T-SQL	**T**ransact-SQL (Microsoft SQL extension)
JSON	**J**ava**S**cript **O**bject **N**otation
AJAX	**A**synchronous **J**avaScript **A**nd **X**ML
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
POST	Typ av HTTP-förfrågning som används i arbetet för att skicka information mellan servern och webbläsaren
Kort	En sida i Thinking Portfolio
Panel	En logisk uppdelning/sektion på ett kort i Thinking Portfolio
Kontroll	Också kallad kontrollgrupp. En samling data som kan ändras av användaren. Kan ha extra funktionalitet som summering av olika slags avgifter.
Rapport	En visuell eller överblicksgivande samling av redan existerande data i en portfolio.
SmartForm	Ett ASP.NET-webbformulär specialgjort för en specifik kund. Fungerar oftast bakåtkompatibelt med företagets tidigare rapporteringsformat.
Uttryck	*Eng. Expression.* Förkortad version av T-SQL-förfrågningar som används för att skapa dynamiskt innehåll, tex. ett fält som ändras beroende på en samling värden.

1 INLEDNING

Mitt examensarbete var att utveckla ett konfigureringsverktyg för programvaran Thinking Portfolio. Programmet måste konfigureras för varje ny kund och jag har konfigurerat ungefär 12 nya portfolion under mina 2.5 år på företaget. Under den tiden har jag lärt mig hur programmet fungerar och hur man kunde optimera processen. Mitt examensarbete är avsett att fungera som en grund för ett konfigureringsverktyg som senare kunde vidareutvecklas till en heltäckande lösning.

1.1 Översikt av programvaran Thinking Portfolio

Thinking Portfolio är ett online-verktyg för strategisk portföljförvaltning av projekt och it-system. Projekt-portfolion är den populärare versionen och således kommer detta arbete att använda termer relaterade till projekthantering.



Figur 1. Projektöverblick i Thinking Portfolio.

Thinking Portfolio är mycket konfigurerbar/modulär och kunde användas för att få en översikt av vilken som helst information som kan generaliseras och klassificeras (se Figur 1). Exempelvis har det varit tal om att utveckla en idé-portfolio, som kunde användas för evaluering och överblick av vad som senare kunde tas upp som ett projekt.

Thinking Portfolio används för att förenhetliga, effektivera och tillföra transparens i beslutsfattande och ledning (eng. *management*) av projekt. Faktorer som vanligtvis inkluderas är resurs- och tidsanvändning, budget och annat finansrelaterat, fasplanering och riskevaluering.

1.2 Bakgrund

Eftersom våra kunder verkar inom olika brancher och har olika projektprocesser och styrningsmodeller konfigureras programvaran specifikt för varje kund. Via workshopar och olika medel av kommunikation med kunden, skapas en modell av hur portfolion skall byggas upp. Denna modell sätts ihop till en PowerPoint-fil med skisser av alla sidor. Jag går sedan igenom PowerPoint-filen tillsammans med dem som satt ihop den för att få en förståelse för kundens val av uppbyggnad.

Efter det börjar själva konfigurationen som typiskt tar 10-12 dagar beroende på portfolions storlek och omfattning. Hela processen förklaras i kapitel 2.3 Konfigureringsprocessen. Den innebär i korthet kopiering och ändring av konfigurationsdata (SQL, text, html, uttryck) från andra databaser och skapandet av SmartForms, skräddarsydda sidor för kundens behov t.ex. projektplan (eng. *project charter*) eller framstegsrapport (eng. *progress report*).

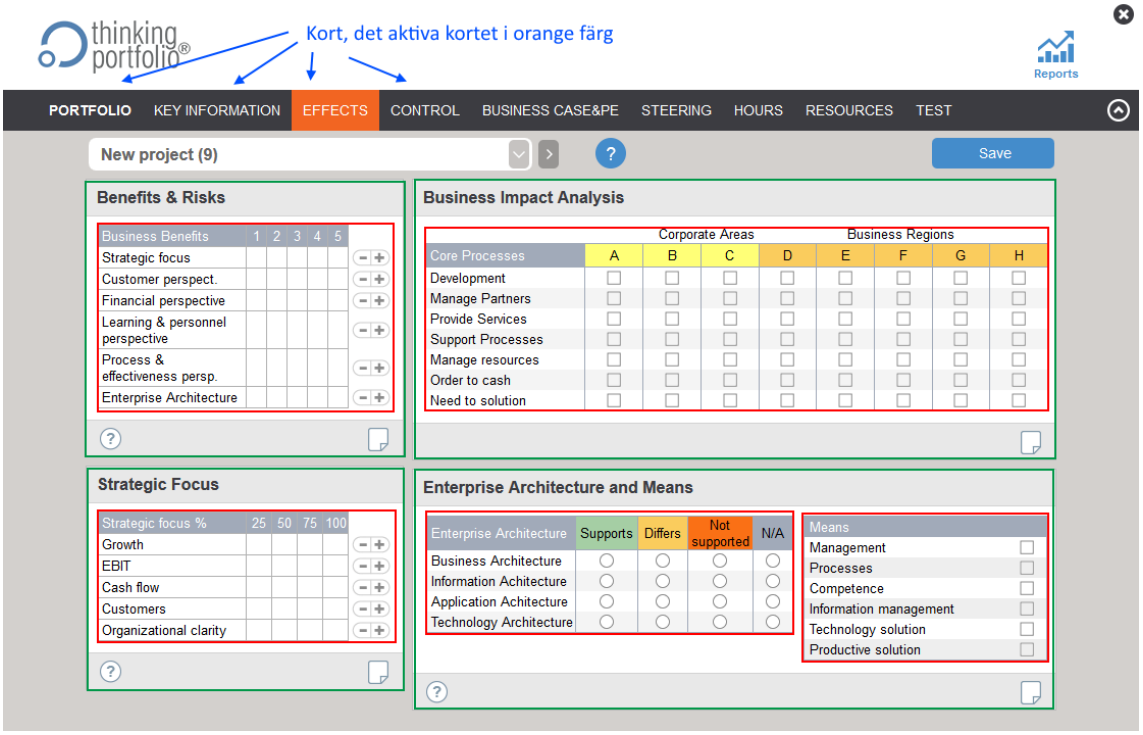
1.3 Målsättning, syfte och metoder

Målet är att minska på konfigureringstiden, göra konfigureringsarbetet intuitivare och mindre manuellt. Detta medför att kostnaden för en s.k. Proof-of-Concept eller demo kunde sänkas, vilket betyder att tröskeln för nya kunder att börja använda produkten

sänks. Det betyder i sin tur att vi kan hantera flera kunder och har mera tid för vidareutveckling och försäljning.

Arbetets huvudsakliga metoder är programvaru-utveckling, dokumentation och tester.

1.4 Avgränsning



Figur 2. En sida i portfolion kallas kort, det aktiva kortet har orange bakgrundsfärg i menyn. Paneler är logiska uppdelningar/sektioner på ett kort och har på illustrationen gröna ramar. Kontroller eller kontrollgrupper har röda ramar och visar information om projektet.

Det är möjligt att optimera konfigurationsprocessen på många sätt, men jag har valt att utveckla ett verktyg som endast hanterar portfolions huvudsakliga struktur och datamodell dvs. kort, paneler och kontroller (se Figur 2).

Figur 3. Rapport dialog med rapportgrupper och exempelrapport.

Portfolions rapporter (se Figur 3) kunde vara ett vidareutvecklingsarbete. För tillfället finns det över 40 olika typer av rapporter och att generalisera ett gemensamt format för redigering av dessa kräver mycket tid, vilket är orsaken till att jag har valt att inte inkludera det i mitt arbete.

SmartForms, som essentiellt är skräddarsydda ASP.NET webbformulär, varierar från kund till kund och kan därför inte generaliseras. Dessa används oftast som en bakåtkompatibilitet (eng. *backwards compatibility*) med företagets tidigare rapporteringsformat, vilket betyder att upplägget varierar kraftigt.

Skapandet av en ny databas kunde också optimeras genom att skapa en modelldatabas och ett program som klonar denna databas och överför konfigurationsdata från den lokala databasen. Detta görs å andra sidan mycket sällan och går att göra med inbyggda funktioner och verktyg på ett snabbt sätt då man blivit van, så det har inte någon hög prioritet.

2 PLANERING

Eftersom jag fick ganska fria händer med examensarbetet gick min planering mest ut på diskussioner med huvudutvecklaren, tillbakablick på tidigare konfigureringar och allmänt filosoferande över vad som kunde göra processen enklare. Först tänkte jag att verktyget skulle vara skilt från Thinking Portfolio, men eftersom alltför mycket kod i så fall borde kopieras över verkade det som onödigt arbete. Till slut insåg jag att jag inte kunde göra ett alltför stort praktiskt arbete, eftersom man faktiskt också måste sätta tid på att skriva

en rapport, varvid jag valde att lite tona ner på idéerna. Jag skisserade olika versioner och delar av användargränssnittet och kom slutligen fram till det som beskrivs i kapitel 2.4 Gränssnitt.

I detta kapitel beskrivs Thinking Portfolios omgivning och vilka tabeller konfigureringsverktyget kommer att hantera. Genomgången av konfigureringsprocessen förklarar hur arbetet brukar genomföras, för att man skall kunna förstå vad verktyget kunde förbättra.

2.1 Plattform och omgivning

Thinking Portfolio använder omgivningen *ASP.NET*, som bygger på ramverket *.NET*. Serverprogramvaran är således *Microsoft Internet Information Services, IIS*.

Som databas används *Microsoft SQL Server*. *Microsoft SQL Server Management Studio* används som användargränssnitt för databaserna tillsammans med instickprogrammet (eng. *plug-in*) *SMSS Tools*.

Applikationskoden är i C# och utvecklingsmiljön (eng. *Integrated Development Environment, IDE*) är *Microsoft Visual Studio Professional 2013*. Utöver C# så används HTML, CSS och JavaScript med olika bibliotek såsom jQuery för att skapa en modern och responsiv webbapplikation.

2.2 Strukturer och tabeller

I detta underkapitel presenteras de tabeller som konfigureringsverktyget kommer att hantera och förklaras hurudan information som kan sparas i tabellerna.

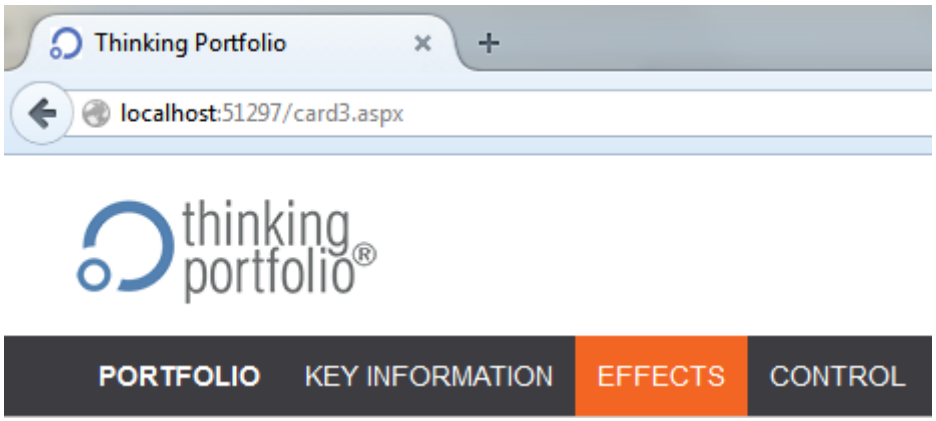
2.2.1 Kort

Tabell 1. Kolumner i Kort-tabellen.

Kolumnnamn	Datatyp
CardId	heltal
Name	teckensträng
URL	teckensträng
CardOrder	heltal

Tabell 2. Exempeldata i Kort-tabellen.

CardId	Name	Url	CardOrder
0	Portfolio	card1.aspx	1
1	Key Information	card2.aspx	2
2	Effects	card3.aspx	3
3	Control	card4.aspx	4



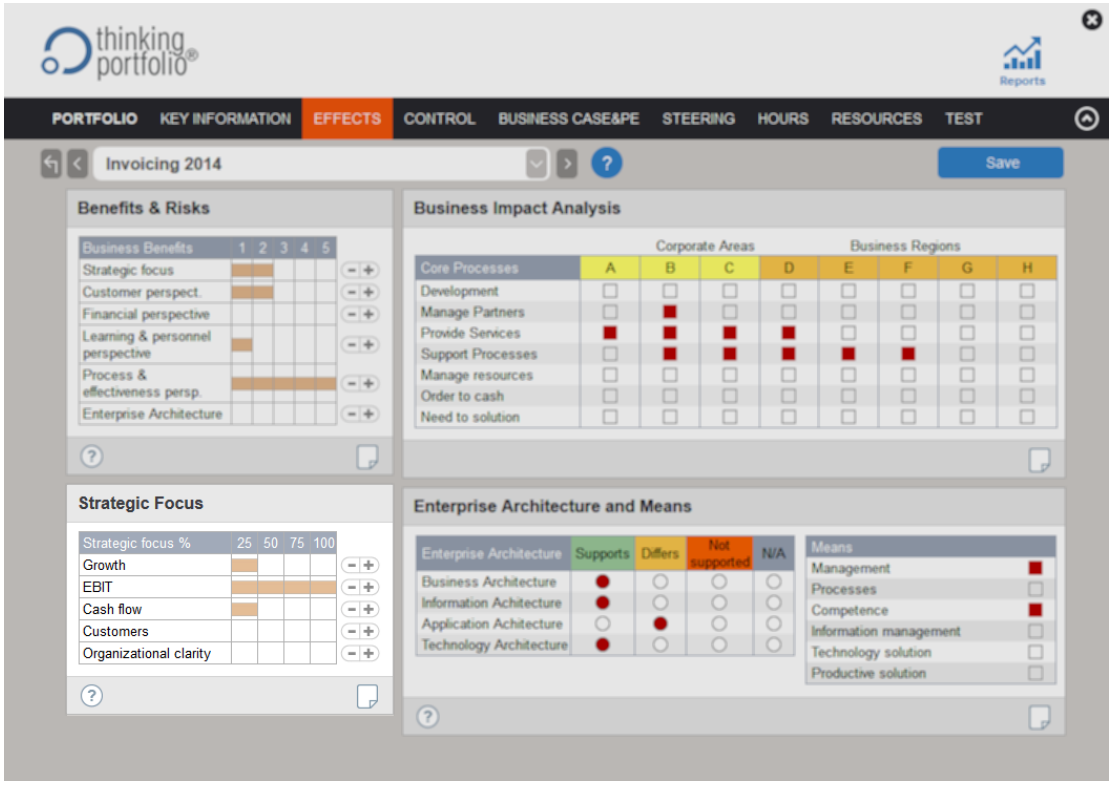
Figur 4. Exempeldata från Tabell 2 som portfoliorepresentation.

Då vi jämför Figur 4 och Tabell 2, ser vi att värdet i kolumnen URL är domän/card3.aspx för kortet Effects. CardId används både i Panel-tabellen och i Kort-tabellen för att peka på vilket kort som panelen eller kontrollen hör till. Detta är relevant pga. användar-rättigheter, som baserar sig på kort, organisation och roll. Med hjälp av CardOrder kan man ändra kortens ordningsföljd efter konfigurering istället för att vara tvungen att ändra CardId i Kort-tabellen, Panel-tabellen och Kontroll-tabellen.

2.2.2 Panel

Tabell 3. Kolumner i Panel-tabellen och exempeldata.

Kolumnnamn	**Datatyp**	**Exempeldata**
ControlId	heltal	31
CardId	heltal	2
Label	teckensträng	Strategic Focus
Position	teckensträng	BottomLeft
Rows	heltal	NULL
Columns	heltal	1
HelpUrl	teckensträng	NULL
Notes	heltal	1
PartialRender	heltal	1
HideInBook	heltal	NULL
Content	teckensträng	<pf:pfBar MaxValue="4" runat="server" ControlId="320"/>
DefaultNote	teckensträng	NULL



Figur 5. Exempeldatan från Tabell 3 som portföljrepresentation.

I Figur 5 ser man den upplysta panelen *Strategic Focus*, som byggs upp av exempeldata i Tabell 3:

- Varje panel har ett ControlId-värde.
- CardId bestämmer på vilket kort som panelen visas, i detta fall 2 som är Effects, se Tabell 2.
- Panelens titel kommer från värdet i kolumnen Label.
- Positionen bestäms av värdet i kolumnen Position. Värdet kan vara TopLeft, TopMiddle, TopRight, TopHalf, BottomLeft, BottomMiddle, BottomRight, BottomHalf eller PopUp.
- Panelens bredd bestäms av värdet i Columns. Värdet kan vara ett heltal mellan 1 och 4. Värdet 4 betyder att panelen fyller hela sidan.
- I panelens högra nedre hörn kan man se ikonen för anteckningar som öppnar ett popup-textfält. Ikonen syns om värdet i kolumnen Notes är 1.
- Om värdet i kolumnen PartialRender är 1 uppdateras en del av innehållet i webbsidan genom AJAX. I exemplet sker detta då man musklickar på + eller – på panelens pfBar-kontroll.

- Content-kolumnen är panelens HTML-innehåll, oftast endast en kontroll, men kan också vara andra HTML-taggar t.ex. table, p, div. I exemplet är Content bara en pfBar-kontroll med 4 möjliga värden.

HelpUrl kan vara en länk till en extern webbsida som beskriver hur panelens innehåll skall användas.

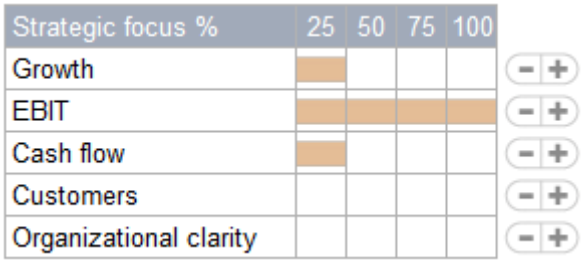
2.2.3 Kontroll

Tabell 4. Kolumner i Kontroll-tabellen.

Kolumnnamn	Datatyp
ControlId	heltal
DataId	heltal
ControlType	teckensträng
DataType	heltal
CardId	heltal
Label	teckensträng
Property_1	teckensträng
Property_2	teckensträng
Property_3	teckensträng
Property_4	teckensträng

Tabell 5. Exempeldata för en kontrollgrupp i Kontroll-tabellen.

ControlId	DataId	ControlType	DataType	CardId	Label	P1	P2	P3	P4
320	3200	pfBar	NULL	2	25;50;75;100	Strategic focus %	NULL	NULL	NULL
320	3205	pfBarItem	2	2	Growth	NULL	NULL	NULL	NULL
320	3210	pfBarItem	2	2	EBIT	NULL	NULL	NULL	NULL
320	3215	pfBarItem	2	2	Cash flow	NULL	NULL	NULL	NULL
320	3220	pfBarItem	2	2	Customers	NULL	NULL	NULL	NULL
320	3225	pfBarItem	2	2	Organizational clarity	NULL	NULL	NULL	NULL



Figur 6. Exempeldata från Tabell 5 som portföljrepresentation.

I Figur 6 visas en pfBar-kontroll för exempeldata från Tabell 5

- Värden i kolumnen ControlId är oftast heltal mellan 100 och 1000. Ett antal identiska värden definierar en kontrollgrupps gemensamma id, med vilken man lägger in kontroller i paneler (se Content i Tabell 3).
- DataId-värdet är unikt för varje rad i Kontroll-tabellen och används då kontrollens data sparas. T.ex. sparas det i exemplet värdet 1 på DataId 3205, 4 på 3210 och 1 på 3215.
- ControlType beskriver typen av kontroll, som i slutändan är C# klasser. I exempeldatan är detta pfBar och dess underkontroller pfBarItem.
- DataType används då kontrollens data sparas. 1 = string, 2 = int, 3 = float, 4 = date.
- CardId-värdet pekar på vilket kort data befinner sig och används för användar-rättigheter som baserar sej på roller.
- Label är titeln på kontrollen och används lite olika beroende på kontrollen.
- Property 1-4 är kolumner med värden för extra konfigureringsdata och kan innehålla allt från text till T-SQL uttryck till CSS.

2.3 Konfigureringsprocessen

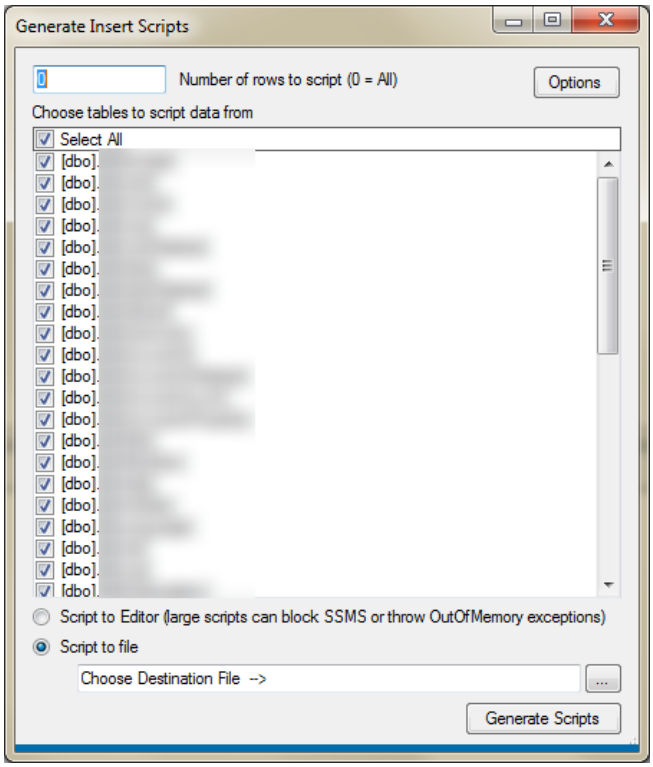
Varje kund har en egen databas i vilken portfolions struktur och data lagras. I en typisk databas finns ca. 35 tabeller, varav 7 beskriver användargränssnittets struktur. I övriga tabeller finns data och olika tilläggsfunktioner. Endast i ett par tabeller kan lagrade värden ändras av kunden via användargränssnittet, men kräver konfigurering på förhand för att fungera.

Konfigureringsprocessen kan indelas i 6 steg, som bekrivs i underkapitlen.

2.3.1 Skapa ny databas

Det första steget är att skapa en ny databas för den nya konfigureringen. Oftast brukar jag söka efter en kund-portfolio som till stor del har likadant upplägg som den nya kunden önskar och använda *Microsoft SQL Server Management Studios* funktionalitet vid namnet *Restore Database* som ger möjligheten att kopiera en existerande databas, både tabeller och data, med bara några steg. Sedan utför jag några SQL-kommandon av typen ”TRUNCATE TABLE TabellNamn” för att ta bort existerande kund-data.

Alternativt kan man använda sig av *New database*, sedan skilt skapa tabeller med *Tasks -> Generate Scripts* och till sist använda *SSMS Tools -> Generate Insert Statements* (se Figur 7) för att välja de tabeller man behöver data från.



Figur 7. SMSS Tools Generate Insert Statements 1. I dialogen kan man välja vilka tabeller som inkluderas.

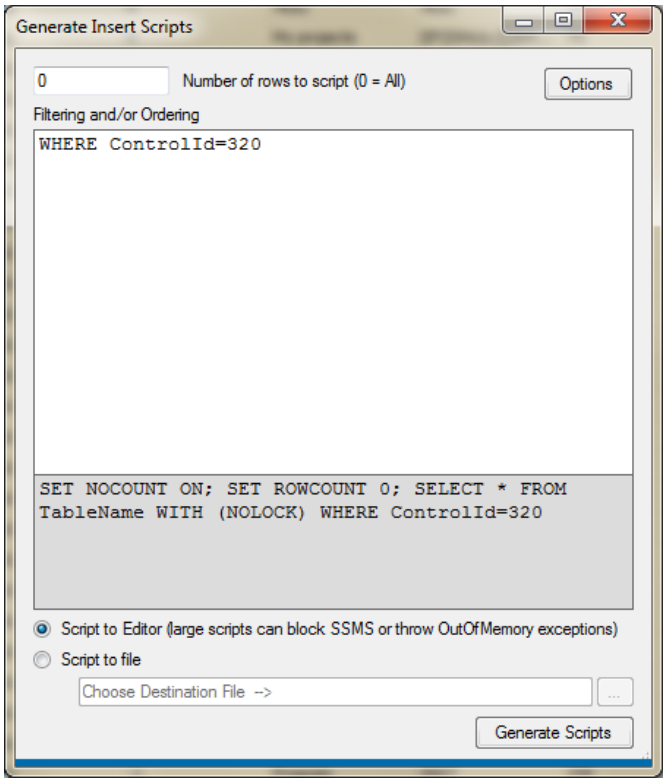
Till sist lägger man till databasens namn samt *Connection string* till master-databasen.

2.3.2 Kort och paneler

Det andra steget är att skapa den större helheten som byggs upp av kort och paneler. Som man ser i Tabell 2 och Tabell 3 behöver man inte fylla i mycket data. Jag brukar helt enkel högerklicka tabellen och välja *Edit All Rows* för att manuellt skriva in namnen på alla kort och paneler. Jag lämnar panelernas Content tom och fyller i dessa efter nästa skede då jag skapat kontrollerna.

2.3.3 Kontroller

Skapandet av kontroller är det skede som oftast tar mest tid. Jag brukar gå igenom tidigare kunders databaser och söka efter kontroller som liknar dem jag behöver. När jag hittat en lämplig kontroll använder jag *SMSS Tools -> Generate Insert Statements*.



Figur 8. SMSS Tools Generate Insert Statements 2.

```
USE [redacted]
SET NOCOUNT ON;
SET XACT_ABORT ON;
GO

BEGIN TRANSACTION;
INSERT INTO [dbo].[redacted]([ControlId], [DataId], [ControlType], [DataType], [CardId], [Label], [Property_1], [Property_2], [Property_3], [Property_4])
VALUES (320, 3200, N'pfBar', NULL, 2, N'25;50;75;100', N'Strategic focus    %', NULL, NULL, NULL),
(320, 3205, N'pfBarItem', 2, 2, N'Growth', NULL, NULL, NULL, NULL),
(320, 3210, N'pfBarItem', 2, 2, N'EBIT', NULL, NULL, NULL, NULL),
(320, 3215, N'pfBarItem', 2, 2, N'Cash flow', NULL, NULL, NULL, NULL),
(320, 3220, N'pfBarItem', 2, 2, N'Customers', NULL, NULL, NULL, NULL),
(320, 3225, N'pfBarItem', 2, 2, N'Organizational clarity', NULL, NULL, NULL, NULL)
COMMIT;
RAISERROR (N'[dbo].[redacted]: Insert Batch: 1.....Done!', 10, 1) WITH NOWAIT;
GO
```

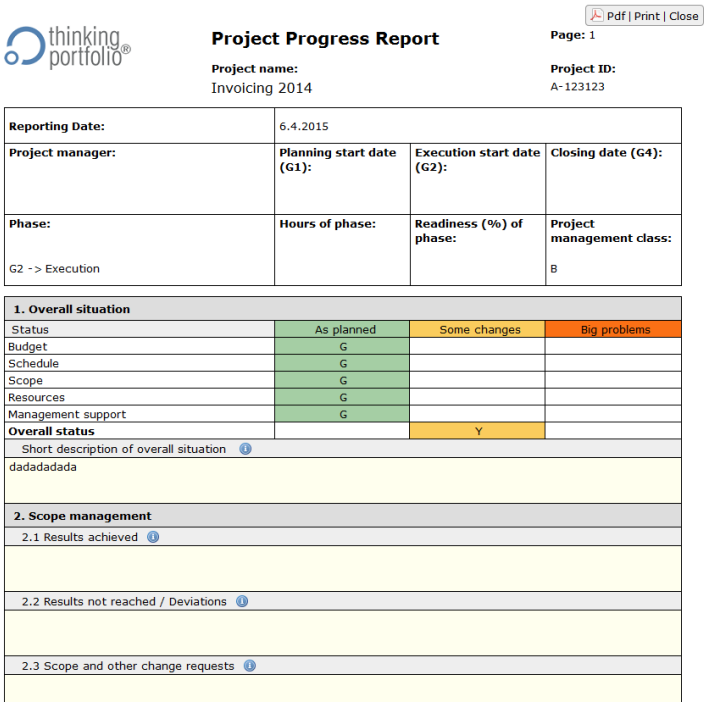
Figur 9. Färdigt genererad kontrolldata från SMSS Tools.

Efter att jag genererat SQL-förfrågningarna som visas i Figur 8 och Figur 9, ändrar jag manuellt data i skriptet, ser till att ControlId och DataId ryms i den nya databasen och exekverar sedan skriptet.

Om ingen liknande kontroll existerar från tidigare, måste man antingen skapa en ny kontroll i C# (om kontrollen är så generisk att den kan användas för någon annan kund) eller göra en snabb lösning genom att kombinera enklare baskontroller i en HTML-tabell.

2.3.4 SmartForms

SmartForm är namnet som används för att beskriva skräddarsydda ASP.NET webbformulär för kunderna. Dessa formulär använder information från portfolion och sätter ihop den till en helhet som kan skrivas ut eller sparas som en pdf-fil.



thinking portfolio®

Project Progress Report

Pdf | Print | Close

Page: 1

Project name:
Invoicing 2014

Project ID:
A-123123

Reporting Date:	6.4.2015		
Project manager:	**Planning start date (G1):**	**Execution start date (G2):**	**Closing date (G4):**
Phase: G2 -> Execution	**Hours of phase:**	**Readiness (%) of phase:**	**Project management class:** B

1. Overall situation			
Status	As planned	Some changes	Big problems
Budget	G		
Schedule	G		
Scope	G		
Resources	G		
Management support	G		
Overall status		Y	

Short description of overall situation

dadadadada

2. Scope management

2.1 Results achieved

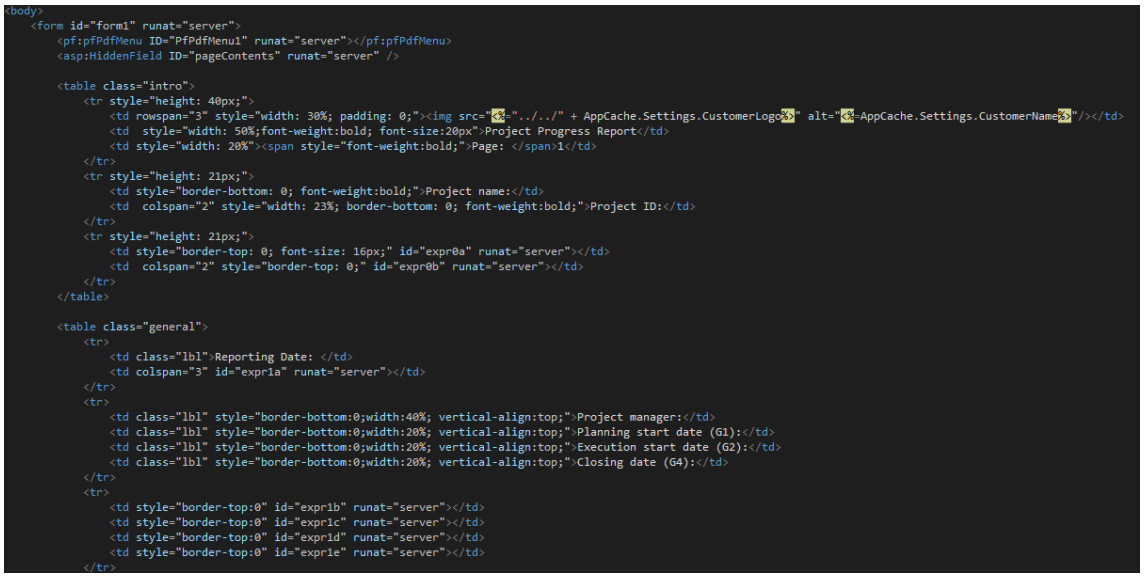
2.2 Results not reached / Deviations

2.3 Scope and other change requests

Figur 10. En demonstrations framstegsrapport.

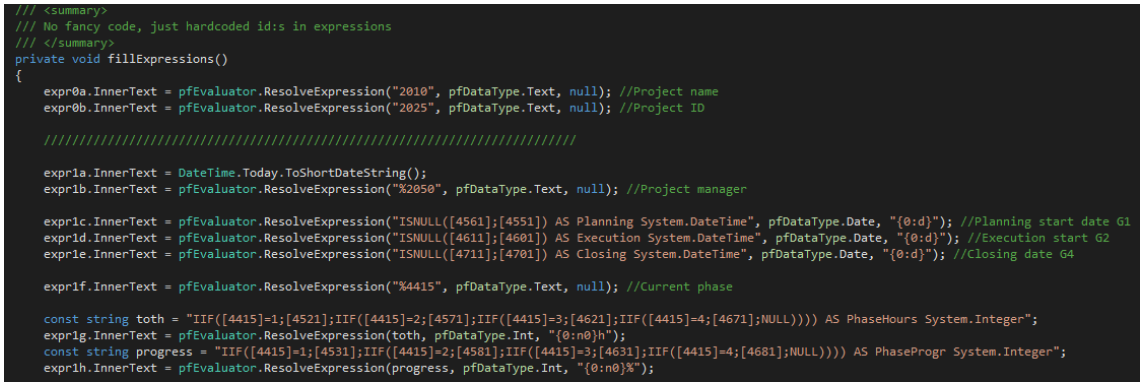
.

Framstegsrapporten i Figur 10 byggs upp av 2 komponenter i koden. I Figur 11 kan vi se den första komponenten av framstegsrapporten, en aspx-sida som är uppbyggd av HTML, CSS och JavaScript.



```
<body>
    <form id="form1" runat="server">
        <pf:pfPdfMenu ID="PfPdfMenu1" runat="server"></pf:pfPdfMenu>
        <asp:HiddenField ID="pageContents" runat="server" />

        <table class="intro">
            <tr style="height: 40px;">
                <td rowspan="3" style="width: 30%; padding: 0;"><img src="<%="../../" + AppCache.Settings.CustomerLogo%>" alt="<%=AppCache.Settings.CustomerName%>"/></td>
                <td  style="width: 50%;font-weight:bold; font-size:20px">Project Progress Report</td>
                <td style="width: 20%"><span style="font-weight:bold;">Page: </span>1</td>
            </tr>
            <tr style="height: 21px;">
                <td style="border-bottom: 0; font-weight:bold;">Project name:</td>
                <td  colspan="2" style="width: 23%; border-bottom: 0; font-weight:bold;">Project ID:</td>
            </tr>
            <tr style="height: 21px;">
                <td style="border-top: 0; font-size: 16px;" id="expr0a" runat="server"></td>
                <td  colspan="2" style="border-top: 0;" id="expr0b" runat="server"></td>
            </tr>
        </table>

        <table class="general">
            <tr>
                <td class="lbl">Reporting Date: </td>
                <td colspan="3" id="expr1a" runat="server"></td>
            </tr>
            <tr>
                <td class="lbl" style="border-bottom:0;width:40%; vertical-align:top;">Project manager:</td>
                <td class="lbl" style="border-bottom:0;width:20%; vertical-align:top;">Planning start date (G1):</td>
                <td class="lbl" style="border-bottom:0;width:20%; vertical-align:top;">Execution start date (G2):</td>
                <td class="lbl" style="border-bottom:0;width:20%; vertical-align:top;">Closing date (G4):</td>
            </tr>
            <tr>
                <td style="border-top:0" id="expr1b" runat="server"></td>
                <td style="border-top:0" id="expr1c" runat="server"></td>
                <td style="border-top:0" id="expr1d" runat="server"></td>
                <td style="border-top:0" id="expr1e" runat="server"></td>
            </tr>
```

Figur 11. En del av framstegsrapportens HTML-kod (demoprogress.aspx).



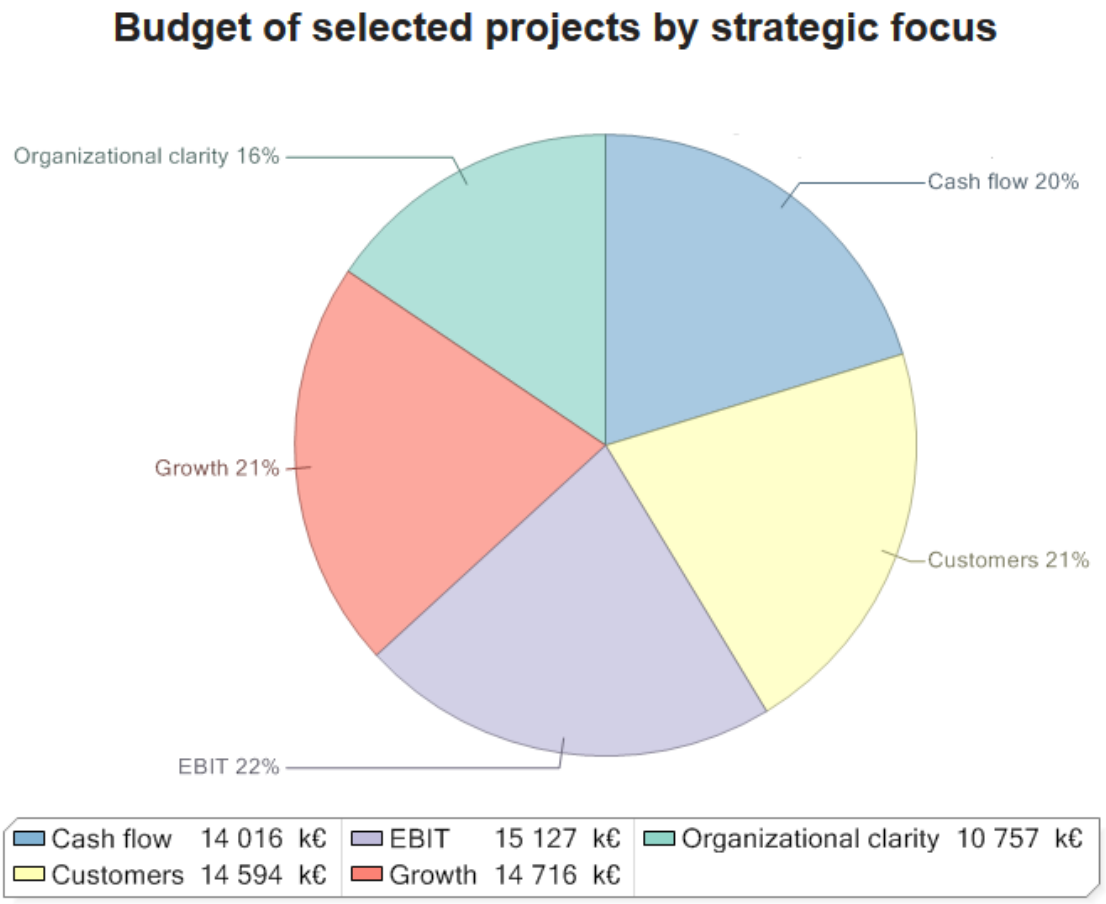
```
/// <summary>
/// No fancy code, just hardcoded id:s in expressions
/// </summary>
private void fillExpressions()
{
    expr0a.InnerText = pfEvaluator.ResolveExpression("2010", pfDataType.Text, null); //Project name
    expr0b.InnerText = pfEvaluator.ResolveExpression("2025", pfDataType.Text, null); //Project ID

    ////////////////////////////////////////////////////////////////////////

    expr1a.InnerText = DateTime.Today.ToShortDateString();
    expr1b.InnerText = pfEvaluator.ResolveExpression("%2050", pfDataType.Text, null); //Project manager

    expr1c.InnerText = pfEvaluator.ResolveExpression("ISNULL([4561];[4551]) AS Planning System.DateTime", pfDataType.Date, "{0:d}"); //Planning start date G1
    expr1d.InnerText = pfEvaluator.ResolveExpression("ISNULL([4611];[4601]) AS Execution System.DateTime", pfDataType.Date, "{0:d}"); //Execution start G2
    expr1e.InnerText = pfEvaluator.ResolveExpression("ISNULL([4711];[4701]) AS Closing System.DateTime", pfDataType.Date, "{0:d}"); //Closing date G4

    expr1f.InnerText = pfEvaluator.ResolveExpression("%4415", pfDataType.Text, null); //Current phase

    const string toth = "IIF([4415]=1;[4521];IIF([4415]=2;[4571];IIF([4415]=3;[4621];IIF([4415]=4;[4671];NULL)))) AS PhaseHours System.Integer";
    expr1g.InnerText = pfEvaluator.ResolveExpression(toth, pfDataType.Int, "{0:n0}h");
    const string progress = "IIF([4415]=1;[4531];IIF([4415]=2;[4581];IIF([4415]=3;[4631];IIF([4415]=4;[4681];NULL)))) AS PhaseProgr System.Integer";
    expr1h.InnerText = pfEvaluator.ResolveExpression(progress, pfDataType.Int, "{0:n0}%");
```

Figur 12. En del av framstegsrapportens C#-kod (demoprogress.aspx.cs).

Den andra komponenten är en aspx.cs-klass (se Figur 12), också kallad *code-behind*, som är C# kod som formaterar innehållet som presenteras i framstegsrapporten.

Innehållet i dessa SmartForms varierar kraftigt men oftast handlar det om en projektplan (eng. *project charter*) eller en framstegsrapport (eng. *progress report*). Jag brukar oftast kopiera en tidigare SmartForm med liknande innehåll och redigera innehållet i den.

2.3.5 Rapporter



Figur 13. En exempelrapport.

Jag har en grundläggande förståelse för hur rapporter fungerar. Rapporterna använder data som finns i portfolion och ritar upp tabeller eller diagram (se Figur 13) som visar en större helhet. Rapporterna görs alltid sist, eftersom alla kontrollers DataId används. Jag skapar oftast inte rapporterna, så jag har inte så mycket att säga om detta.

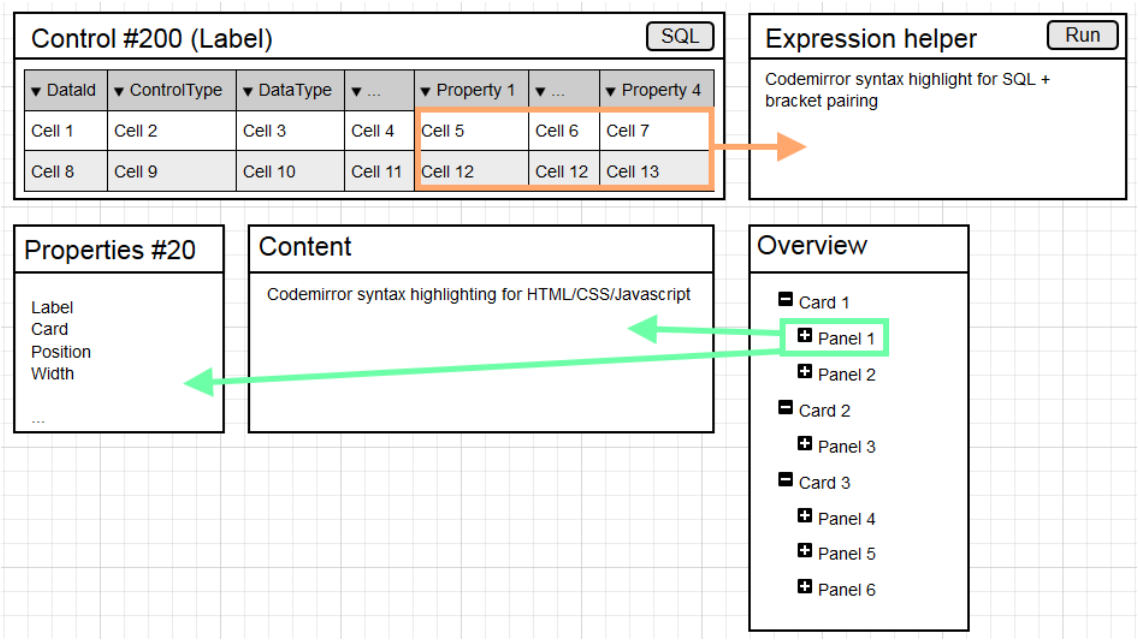
2.3.6 Överföring av databas och versionsbyte

Till sist skapar man en databas på servern och överför informationen mha. att spara SQL-förfrågningarna som ett skript med SMSS Tools (se Figur 7), kopiera över skriptet till servern och exekvera skriptet. Om den nya portfolion har SmartForms måste man också uppdatera mjukvaran Thinking Portfolio, eftersom SmartForms är nya filer.

Versionbytet sker genom att inkrementera versionsnumret, pånytt uppbygga hela lösningen (eng. *Rebuild Solution*) och sedan publicera (eng. *Publish*) nya versionen av Thinking Portfolio. Sedan loggar man in på servern och kopierar över den nya katalogen från den lokala maskinen till servern, kopierar över några filer från förra versionen på servern (bl.a. Web.config) och byter katalog i IIS (*Internet Information Services*). Webbapplikationen omstartas automatiskt, och den nya versionen är direkt tillgänglig för alla.

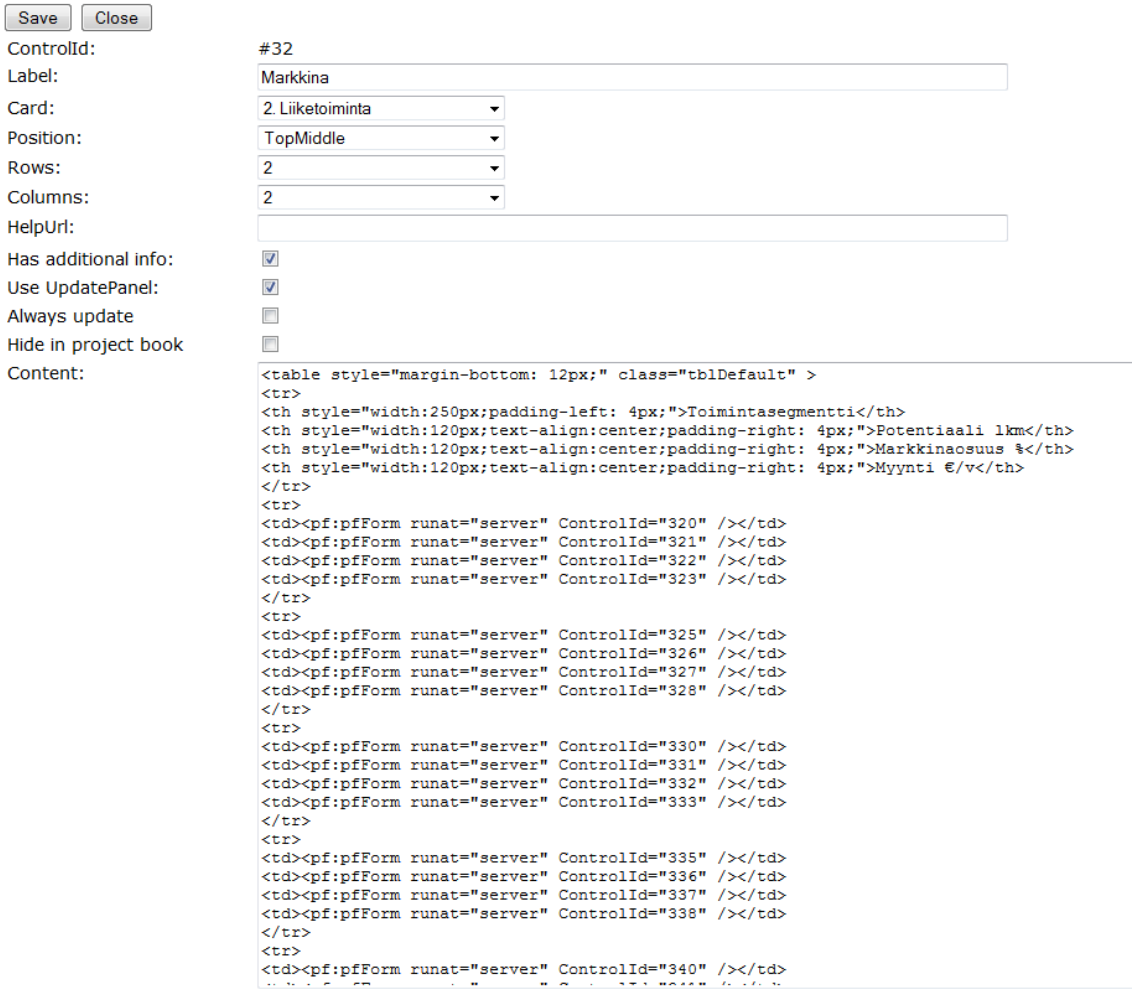
2.4 Gränssnitt

Figur 14 visar den version av gränssnittet som jag först skisserade på papper. Det slutgiltiga gränssnittet blev inte exakt likadant, nya idéer och begränsningar uppstod då jag programmerade.



Figur 14. Beskrivning hur gränssnittet skall fungera. De två översta fönstren (Control och Expression helper) är för kontroller medan Properties och Content är för panelerna. Overview visar en överblick över portfolion och borde fungera som navigerings-fönster.

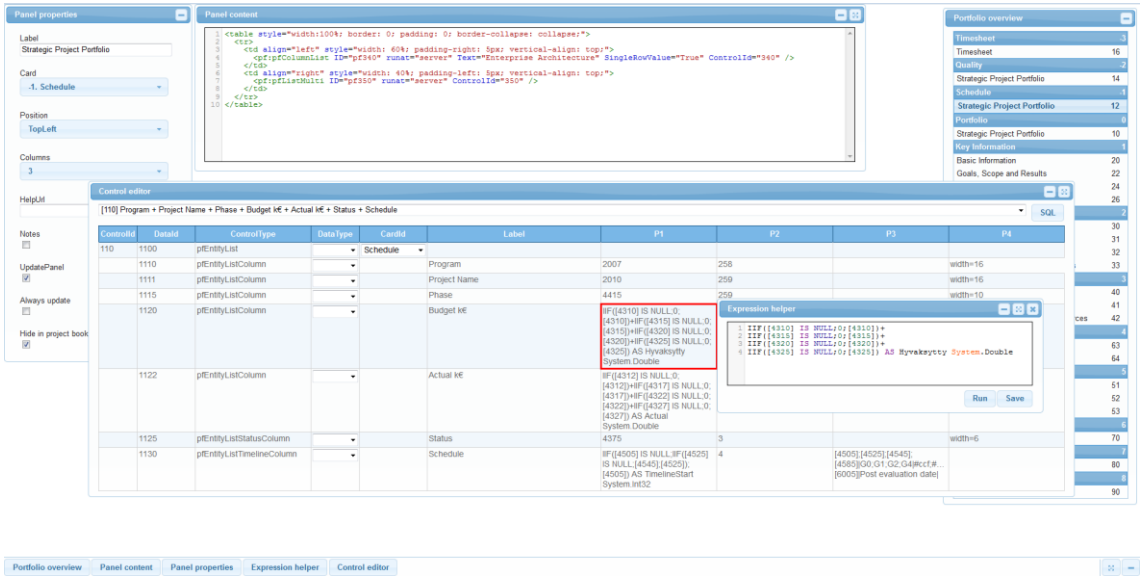
Den ursprungliga tanken var att man skulle kunna klicka på en kontrolls HTML-tagg i Content för att öppna den i fönstret Control. Problemet med denna lösning är att man inte kommer åt alla tillgängliga kontroller på detta sätt utan endast de kontroller som redan är synliga i panelerna. Jag löste problemet med att ha en rullgardingsmeny i Control fönstret med alla kontroller som finns på den valda panelens kort.



Figur 15. panel.aspx användes som bas för arbetet.

Som bas för examensarbetet modifierade jag en webbsida som används för redigering av innehållet i en panel (se Figur 15). Denna webbsida var det enda konfigureringsverktyg som var inbyggt i programmet redan tidigare. Det var viktigt att samma funktionalitet som fanns på denna webbsida också skulle finnas kvar i detta arbete.

Gränssnittet måste vara praktiskt och ge en täckande överblick av konfigureringen. Detta betyder dock inte att vem som helst utan teknisk eller programspecifik kunskap skulle kunna använda detta konfigureringsverktyg. Det viktigaste är att underlätta arbetet samt att komplettera existerande verktyg.

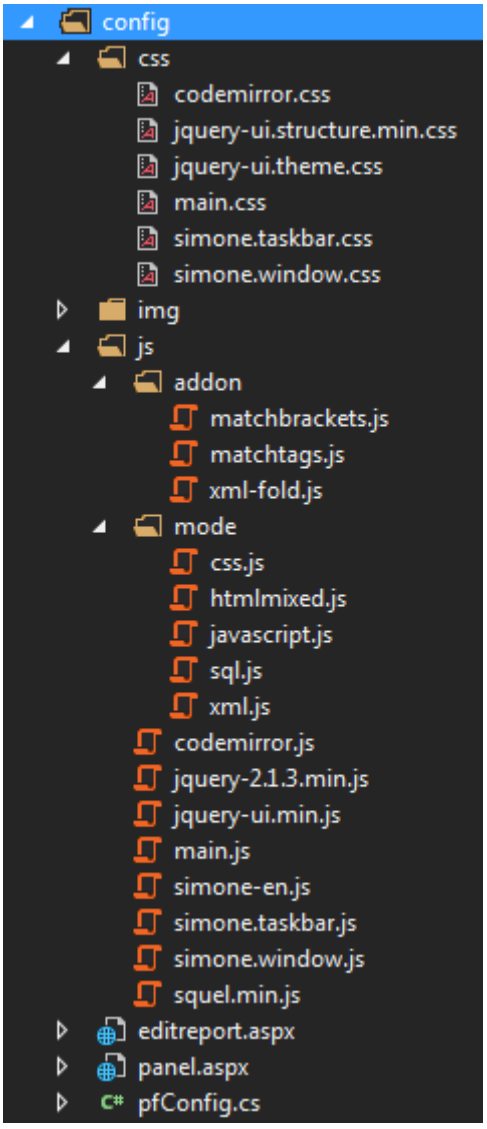


Figur 16. Det färdiga gränssnittet i sin helhet.

I Figur 16 ser man hela gränssnittet för verktyget. Beskrivningar på varje enskiljd del och dess uppbyggnad finns i kapitel 3.2 Front-end.

3 PROGRAMMERING

Detta kapitel går igenom vilka bibliotek som används och kommenterar koden för konfigureringsverktyget. Koden har delats in i front-end (Javscript, HTML, CSS) och back-end (C#, SQL).



Figur 17. Filstrukturen för arbetets kod. Som man ser så är största delen JavaScript.

Som Figur 17 visar är största delen av filerna bibliotek. Den egna koden finns i filerna main.js, main.css, panel.aspx och pfConfig.cs.

3.1 Val av bibliotek

Om jag hade valt att programmera allting från grunden skulle arbetet ha tagit för mycket tid att genomföra. Jag valde att kombinera ett flertal JavaScript-bibliotek för att snabbt skapa en fungerande helhet. I underkapitlen går jag igenom vilka bibliotek jag använt och varför.

3.1.1 CodeMirror

I (CodeMirror 2015) beskrivs biblioteket CodeMirror på följande sätt:

> CodeMirror is a code-editor component that can be embedded in Web pages. The core library provides only the editor component, no accompanying buttons, auto-completion, or other IDE functionality. It does provide a rich API on top of which such functionality can be straightforwardly implemented. See the addons included in the distribution, and the list of externally hosted addons, for reusable implementations of extra features.
>
> CodeMirror works with language-specific modes. Modes are JavaScript programs that help color (and optionally indent) text written in a given language. The distribution comes with a number of modes (see the mode/ directory), and it isn't hard to write new ones for other languages.

CodeMirror är således ett JavaScript-bibliotek som skapar en webbkontroll likt en textarea och färgar innehållets syntax (*eng. Syntax highlight*).

Biblioteket CodeMirror används i *Panel content* (se 3.2.2 Panel content and properties) som använder HTML och CSS samt *Expression helper* (se 3.2.4 Expression helper) som använder T-SQL.

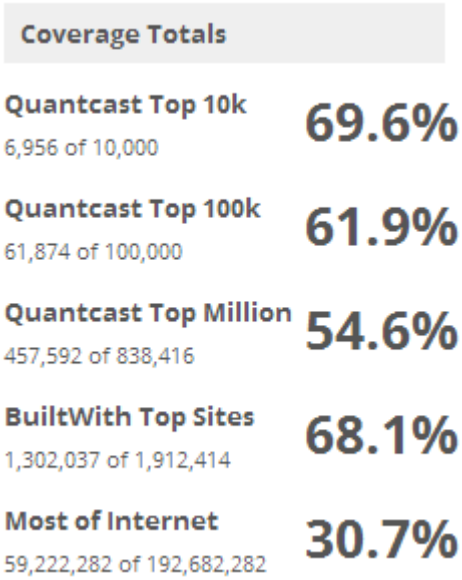
```
1 IIF(ISNULL([2040];0)=@CurrentUser OR
2     ISNULL([2045];0)=@CurrentUser OR
3     ISNULL([2050];0)=@CurrentUser OR
4     ISNULL([4005];0)=@CurrentUser OR
5     ISNULL([4045];0)=@CurrentUser OR
6     ISNULL([4085];0)=@CurrentUser OR
7     ISNULL([4125];0)=@CurrentUser OR
8     ISNULL([4165];0)=@CurrentUser OR
9     ISNULL([4205];0)=@CurrentUser;'My projects';NULL) AS MyProjects System.String
```

Figur 18. CodeMirror i Expression helper.

Färgning av syntax hjälper på många sätt (se Figur 18). Det blir lättare att se vilka HTML-taggar hör ihop, var citationstecken fattas, osv.

3.1.2 jQuery och jQuery UI

jQuery är ett JavaScript-ramverk som har revolutionerat sättet att hantera en webbsidas DOM (Document Object Model) och är nuförtiden en väsentlig del av de flesta webbsidor (se Figur 19).



Figur 19. jQuery-användning Januari 2014 (The State of jQuery 2014).

Jag valde att använda jQuery som jag är van med för att kunna göra utvecklingsarbetet snabbare. Ramverket ger också en viss säkerhet i JavaScript som annars är ett mycket löst specificerat programmeringsspråk, t.ex. inga variabeltyper.

Jag valde att använda användargränssnittet jQuery UI för att med Simone (se 3.1.3 Simone), som är en utvidgning (*eng. extension*) av jQuery UI, enkelt skapa fönster och dialoger för verktygen.

Både jQuery och jQuery UI ingick redan tidigare i Thinking Portfolio. En del andra bibliotek krävde dock nyare versioner av jQuery.

3.1.3 Simone

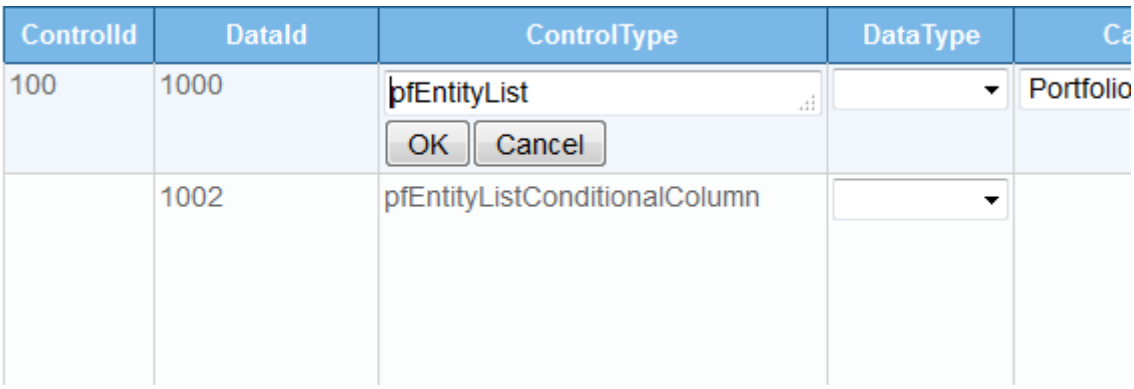
Biblioteket Simone beskrivs på följande sätt i (Simone 2015):

> Simone is a JavaScript window manager consisting of two widgets: taskbar and window, providing a desktop-like experience. It's especially useful for single-page applications. It's built on top of the jQuery and jQuery UI.

Simone är således en fönsterhanteringsutvidgning av jQuery UI, som i sin tur bygger på jQuery. Jag sökte en tid efter en lämplig fönsterhanterare, men de flesta saknade maximeringsfunktionalitet som är väsentlig för *Panel content* (se 3.2.2 Panel content and properties) och *Expression helper* (se 3.2.4 Expression helper) pga. innehållets potentiala storlek. I Simone ingick allt jag behövde och mycket extra, t.ex. helskärmsläge. Simones vidareutveckling pågår ännu.

3.1.4 Jeditable

jQuery-insticksmodulen (*eng. plugin*) Jeditable används för redigering av fält i HTML-tabellen i *Control editor* (se Figur 20 och 3.2.3 Control editor).



Figur 20. Jeditable i Control editor.

Beskrivning på Jeditable i (Jeditable 2015):

> Hi! My name is Jeditable and I am inplace editor plugin for jQuery. With few lines of JavaScript code I allow you to click and edit the content of different html elements. I am based on Dylan Verheul's editable. For those in hurry download latest source or minified. For bleeding edge version check GitHub.

Jeditable ingick redan tidigare i Thinking Portfolio och användes för en översättningstabell.

3.1.5 Squel.js

Beskrivning av Squel.js i (Squel.js 2015):

> Squel helps you quickly and easily build SQL query strings through an object oriented API. There are two main benefits to using Squel to build your queries:
>
> **It requires less effort** - instead of having to use brittle methods like string concatenation and substitution you get a nice clean API to do the hard work for you.
>
> **It lets you build flexibly** - Sometimes you don't yet know what your final query ought to look like until you've checked information from various other parts of your program. Representing the query as an object which can be manipulated allows you to be flexible about how you build it.

En funktionalitet i Control editor är att utmata SQL-förfrågningar för kontrolldata likt SSMS Tools (se Figur 7). Squel.js var en enkel lösning som underlättade generering av SQL-förfrågningar från JavaScript-objekt.

3.2 Front-end

Utvecklingen av konfigureringsverktygets front-end, i HTML/CSS och JavaScript, utgjorde den största delen av mitt examensarbete. All JavaScript-kod finns i filen main.js som innehåller några funktioner och en klass med namnet Config (se Figur 21).

```
"use strict";
var config;

squel.useFlavour('mssql');

function getUrlParameter(sParam)...

function refreshOpener()...


$(document).keydown(function (event)...);

function Config(d)...


$(document).ready(function () {
    $.ajax({
        type: "POST",
        url: window.location.pathname.split('/').pop() + "/ConfigInit",
        data: "{panelId: " + getUrlParameter("id") + "}",
        dataType: "json",
        contentType: "application/json; charset=utf-8",
        success: function (response) {
            var data = JSON.parse(response.d);
            console.log(data);
            config = new Config(data);
        },
        error: function (xhr, status, error) { alert(JSON.parse(xhr.responseText).Message); }
    });


});
```

Figur 21. Överblick över main.js (773 rader).

Klassen *Config* innehåller så gott som all funktionalitet. Ett *Config-objekt*, som skapas med direktivet *new* i JavaScript, initialiseras efter ett AJAX-anrop av funktionen *ConfigInit* som returnerar en JSON-teckensträng. Denna JSON-teckensträng är ett serialiserat C#-objekt i klassen *InitData* (se 3.3 Back-end). Detta objekt konverteras till ett JavaScript-objekt med metoden *JSON.parse* och ges som parameter till *Config-objektets* konstruktor. *InitData* innehåller en tabell (*eng. array*) på alla kort, en tabell på alla paneler och en tabell på alla kontrollgrupper. Detta globala objekt kallas *data* i klassen *Config*.

3.2.1 Portfolio overview

Figur 22. Portfolio overview, kort (blå bakgrund) och tillhörande paneler (med vit bakgrund). Respektive id visas till höger.

Portfolio overview används för att ge en överblick av alla kort och kortens paneler i portfoliokonfigureringen (se Figur 22). Man kan musklicka på en paneltitel för att öppna dess innehåll (*Panel content*) och få fram dess egenskaper (*Panel properties*). Om man ändrar något värde i de andra tillhörande fönstren märks panelen med ett *-tecken efter titeln för att visa att värdet är ändrat och osparat.

```
this.RefreshOverview = function () {
    var ot = $("#OverviewTree");
    ot.html('');
    for (var i = 0; i < data.Cards.length; i++) {
        ot.append('<li class="card ui-widget-header" data-cardid="' + data.Cards[i].CardId.toString() + '">'
    }
    for (var i = 0; i < data.Panels.length; i++) {
        var parent = ot.find("[data-cardid='" + data.Panels[i].CardId.toString() + "']");
        var isDirty = '';
        if (data.Panels[i].IsDirty) isDirty = ' *';
        $('<li class="panel" data-panelid="' + data.Panels[i].PanelId.toString() + '"><p>' + data.Panels[i].
        if (data.Panels[i].PanelId == this.panelId) $("[data-panelid='" + this.panelId.toString() + "']").add
    }

    $("#OverviewTree").menu({
        items: "> :not(.ui-widget-header)"
    }).menu("refresh");
};
```

```
this.RefreshOverview();

$(".panel").on("click", function () {
    config.ChangePanel($(this).data("panelid"));
});
```

Figur 23. Funktion för att skapa/uppdatera överblicksmenyn, data är InitData från ConfigInit.

Såsom det framkommer i Figur 23 från benämningen #OverviewTree var det urprungligen tänkt att skapa en såkallad trädvy (*eng. tree view*), men efter några försök gav jag upp och nöjde mig med en *menu*.

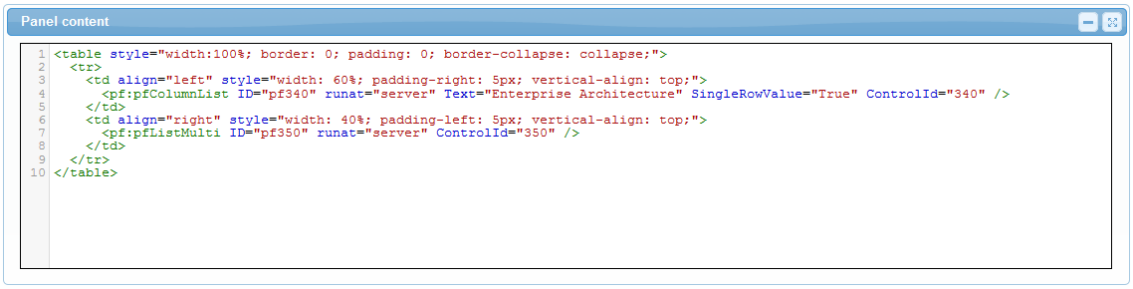
Efter att metoden *RefreshOverview* har anropats inväntar alla paneler en musklick-händelse (*eng. click event*) som anropar metoden *ChangePanel*. Eftersom direktivet *this* är bundet till ifrågavaranade objekt, som representerar panel-elementet i överblicksmenyn, måste den globala referensen till *Config-objektet* användas (se andra raden i Figur 21).

```
this.ChangePanel = function (panelId) {
    var n = -1;
    for (var i = 0; i < data.Panels.length; i++) {
        if (data.Panels[i].PanelId == panelId) {
            n = i;
            break;
        }
    }
    if (n != -1) {
        var panel = data.Panels[n];

        this.panelChanging = true;

        //Update overview
        $("#panelid").val(panel.PanelId);
        $(".selected").removeClass("selected");
        $("[data-panelid='" + panel.PanelId.toString() + "']").addClass("selected");

        console.log(panel);

        //Update values
        this.content.getDoc().setValue(panel.Content);
        $("#Label").val(panel.Label);
        $("#Card").val(panel.CardId).selectmenu("refresh");
        $("#Position").val(panel.Position).selectmenu("refresh");
        $("#Columns").val(panel.Columns).selectmenu("refresh");
        $("#HelpUrl").val(panel.HelpUrl);
        $("#Notes").prop("checked", panel.Notes);
        $("#PartialRender").prop("checked", panel.PartialRender1);
        $("#DependentUpdatePanel").prop("checked", panel.PartialRender2);
        $("#HideInBook").prop("checked", panel.HideInBook);

        this.FillControlList(panel.CardId);


        this.panelChanging = false;
    }
};
```

Figur 24. Metoden ChangePanel.

Metoden *ChangePanel* (se Figur 24) söker först efter panelens index i tabellen *Panels* (*eng. array*) och ändrar sedan den valda panelen i *Portfolio overview* samt alla värden i *Panel content* och *Panel properties*.

3.2.2 Panel content and properties



Figur 25. Panel content.

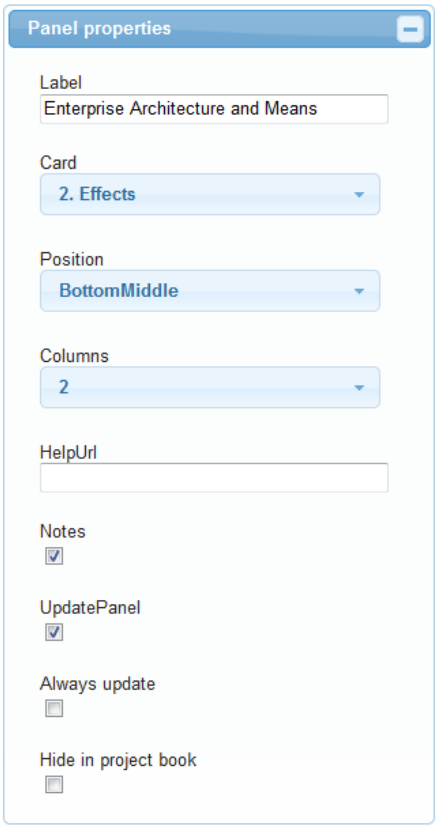
Såsom Figur 25 visar använder *Panel content* CodeMirror för att färga syntaxen för panelens innehåll, som består av HTML och CSS.

```
this.content = CodeMirror.fromTextArea(document.getElementById("Content"), {
    mode: "text/html",
    lineNumbers: true,
    matchTags: { bothTags: true },
    matchBrackets: true,
    extraKeys: { "Ctrl-J": "toMatchingTag" },
    viewportMargin: Infinity
});
this.content.setSize(this.width * 0.63, 500);
this.content.on("change", function(cm, change) {
    config.setContent(cm.getDoc().getValue());
    config.setDirty(true);
});
```

Figur 26. Initialisering av Panel content.

CodeMirror initialiseras med metoden *fromTextArea* vars parametrar är HTML-elementet i fråga och en inställningstabell (se Figur 26). Inställningstabellen innehåller t.ex. vilket läge CodeMirror ska köras i (mode: ”text/html”) och att all text ska renderas på en gång (viewportMargin: Infinity).

Efter att CodeMirror har initialiserats ändras storleken i enlighet med skärmens bredd och därefter inväntar CodeMirror en ändrings-händelse då värdet överförs till objektet *data.Panels[x].Content*, där *x* betecknar ett index värde i tabellen *Panels*.



Figur 26. Panel properties.

Panel properties är en samling textfält, rullgardinsmenyer och kryssrutor (se Figur 26). Alla element inväntar en ändringshändelse (se Figur 27) och uppdaterar bakomliggande objekt på samma sätt som objektet *data.Panels[x].Content* har uppdaterats. *Card, Postion* och *Columns* initialiseras först som *selectmenu*, en rullgardinsmeny som kommer med jQuery UI.

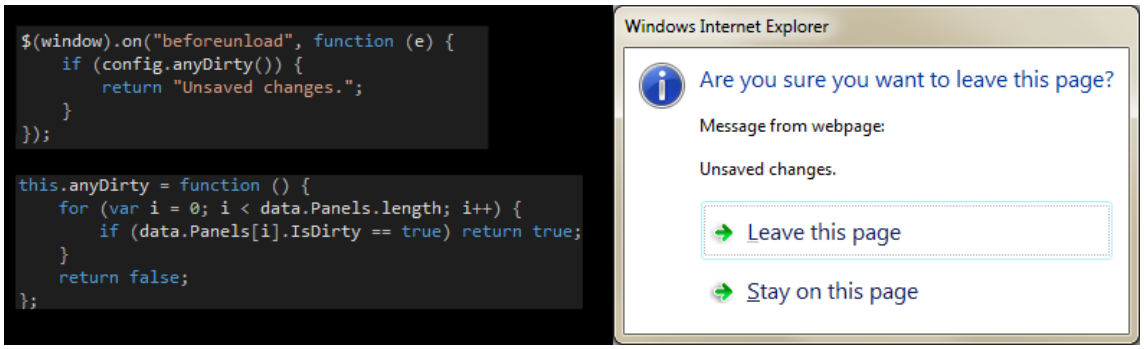
```
$("#Label").on("change", function () {
    config.setLabel($(this).val());
    config.setDirty(true);
});

$("#Card").selectmenu({
    change: function () {
        config.setCard($(this).val());
        config.setDirty(true);
    }
});
```

Figur 27. Två exempel på ändringshändelser för Panel properties.

```
this.setDirty = function (value, pid) {
    if (this.panelChanging) return;
    if (pid === undefined) {
        this.GetCurrentPanel().IsDirty = value;
    } else {
        this.GetPanel(pid).IsDirty = value;
    }
    this.updateDirty();
};

this.setContent = function (value)...;
this.setLabel = function (value) {
    this.GetCurrentPanel().Label = value;
};
this.setCard = function (value)...;
this.setPosition = function (value)...;
this.setColumns = function (value)...;
this.setHelpUrl = function (value)...;
this.setNotes = function (value)...;
this.setPartialRender1 = function (value)...;
this.setPartialRender2 = function (value)...;
this.setHideInBook = function (value)...;

this.GetPanel = function (panelId) {
    for (var i = 0; i < data.Panels.length; i++) {
        if (data.Panels[i].PanelId == panelId)
            return data.Panels[i];
    }
};

this.GetCurrentPanel = function () {
    var p = $("#panelid");
    if (p.length != 0) {
        var n = parseInt(p.val(), 10);
        if (!isNaN(n)) {
            return this.GetPanel(n);
        }
    }
};
```

Figur 28. Getter- och setter-metoder för paneler.

Metoden *GetCurrentPanel* anropar *GetPanel* med den valda panelens PanelId som parameter (se Figur 28). Den valda panelens PanelId sparas i ett gömt element som updateras i metoden *ChangePanel* (se Figur 24). Metoden *setDirty* används för att markera paneler som är ändrade och osparade. Detta indikeras sedan i *Portfolio overview* med ett *-tecken och hindrar sidan att stängas utan att visa en bekräftelse-dialog (se Figur 29).



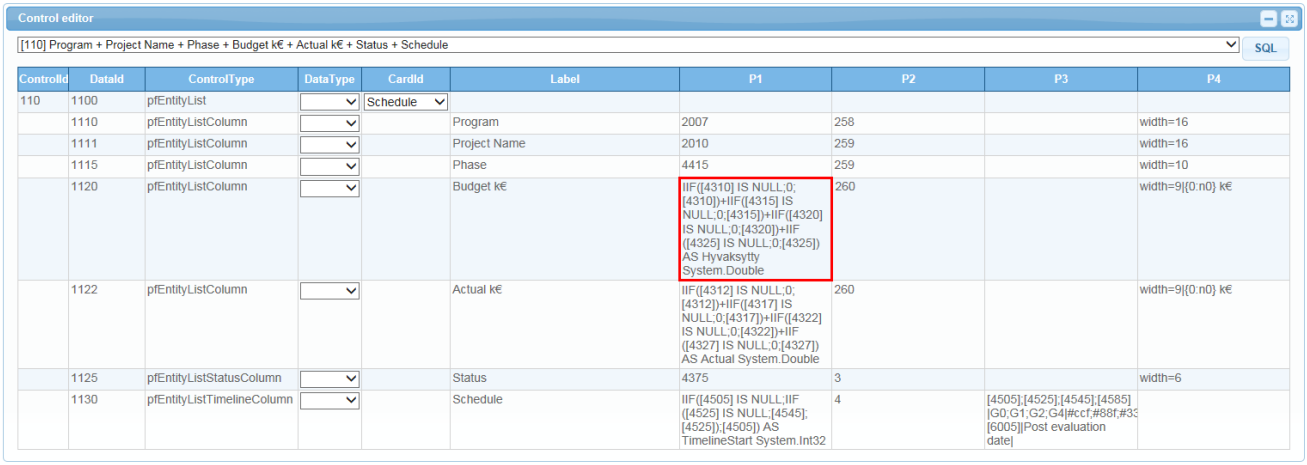
Figur 29. Bekräftelsedialog som syns om det finns osparade paneler och sidan stängs.

Då verktygets användare trycker ned tagenterna Ctrl + S eller musklickar på Save-knappen skickas en JSON-representation av den valda panelen över AJAX till funktionen *ConfigSavePanel* (se Figur 30). Anropet besvaras med den PanelId som framgångsrikt uppdaterats så att användargränssnittet kan uppdateras med metoden *setDirty(false, id)* som tar bort *-tecknet från *Portfolio overview*. Funktionen *refreshOpener* omladdar portfolio-tabben så att man kan se ändringarna.

```
$(document).keydown(function (event) {
    if (!(String.fromCharCode(event.which).toLowerCase() == 's' && event.ctrlKey)) return true;
    event.preventDefault();
    $("#btnSave").click();
});
$("#btnSave").button().on("click", function (e) {
    $.ajax({
        type: "POST",
        url: window.location.pathname.split('/').pop() + "/ConfigSavePanel",
        data: "{ panel: " + JSON.stringify(config.GetCurrentPanel()) + "}",
        dataType: "json",
        contentType: "application/json; charset=utf-8",
        success: function (response) {
            if (response.d) {
                var id = parseInt(response.d, 10);
                if (id > 0) {
                    config.setDirty(false, id);
                }
            }
            refreshOpener();
        },
        error: function (err) { alert(err.responseText); }
    });
    e.preventDefault();
});
```

Figur 30. AJAX-anrop till ConfigSavePanel.

3.2.3 Control editor



Figur 31. ”Control editor”-fönstret består av kontroll-listan, SQL-knappen och en HTML-tabell som innehåller kontrolldata.

I Figur 31 visas *Control editor* som kan användas för att redigera och kopiera kontroller. Överst i fönstret finns en rullgardinsmeny, kontroll-listan, som innehåller namn och ControlId för alla kontroller som har samma CardId som valda panel i *Portfolio overview*. Genom att välja en kontroll i rullgardinsmenyn hämtas och öppnas dess konfigureringsdata. Till höger om rullgardinsmenyn finns en SQL-knapp med vilken man kan generera SQL-förfrågningar av typen INSERT för den valda kontrollen. Största delen av fönstret består av en HTML-tabell som representerar den valda kontrollens konfigureringsdata.

```
$("#ControlList").on("change", function (e) {
    if ($(this).val() != -1) {
        var controlId = $(this).val();
        $.ajax({
            type: "POST",
            url: window.location.pathname.split('/').pop() + "/ConfigGetControlGroup",
            data: "{ controlId: " + JSON.stringify(controlId) + "}",
            dataType: "json",
            contentType: "application/json; charset=utf-8",
            success: function (response) {
                config.ChangeControlGroup(JSON.parse(response.d).Controls)
            },
            error: function (err) { alert(err.responseText); }
        });
    } else {
        //Clear control table
        config.ChangeControlGroup();
    }
});
```

Figur 32. Ändringshändelse för kontroll-listan.

Genom att välja en kontroll i kontroll-listan skickas ett AJAX-anrop till funktionen *ConfigGetControlGroup* med ControlId som parameter (se Figur 32). Anropet besvaras med ett *ControlGroupData*-objekt som innehåller en tabell med *ControlData*-objekt som metoden *ChangeControlGroup* får som parameter.

```
this.ChangeControlGroup = function (controls) {
    var table = $("#ControlTable").find("tbody");
    table.find("tr:not(.header)").remove();
    if (controls === undefined) {
        config.currentControls = null;
        return;
    }
    config.currentControls = controls;
    for (var i = 0; i < controls.length; i++) {
        var row = $("<tr>");

        if (i == 0)
            row.append($('<td>').addClass("ControlId").append(controls[i].ControlId));
        else
            row.append($('<td>').addClass("readonly"));

        row.append($('<td>').addClass("DataId").append(controls[i].DataId));
        row.append($('<td>').addClass("ControlType").append(controls[i].ControlType));

        var dt = config.MakeDataTypeSelect(controls[i].DataType);
        dt.data("ControlId", controls[i].ControlId);
        dt.data("DataId", controls[i].DataId);
        dt.on("change", function () {
            config.updateControlField($(this), $(this).val());
        });
        row.append($('<td>').addClass("readonly").append(dt));

        if (i == 0) {
            var ci = config.MakeCardIdSelect(controls[i].CardId);
            ci.data("ControlId", controls[i].ControlId);
            ci.data("DataId", controls[i].DataId);
            ci.on("change", function () {
                config.updateControlField($(this), $(this).val());
            });
            row.append($('<td>').addClass("readonly").append(ci));
        } else {
            row.append($('<td>').addClass("readonly"));
        }

        row.append($('<td>').addClass("Label").append(controls[i].Label));
        row.append($('<td>').addClass("Property_1").append(controls[i].Property_1));
        row.append($('<td>').addClass("Property_2").append(controls[i].Property_2));
        row.append($('<td>').addClass("Property_3").append(controls[i].Property_3));
        row.append($('<td>').addClass("Property_4").append(controls[i].Property_4));

        row.find("td").data("ControlId", controls[i].ControlId);
        row.find("td").data("DataId", controls[i].DataId);
        table.append(row);
    }
```

Figur 33. Metoden ChangeControlGroup del 1.

I metoden *ChangeControlGroup* skapas en HTML-tabell av konfigureringdata i *Control editor* (se Figur 33). Detta sker genom att gå igenom tabellen *control* med en *for-loop* och skapa td-element med kolumnnamn som klass för varje fält. Fältet ControlId och

rullgardinsmenyn CardId tilläggs på första raden, *if(i == 0)*, eftersom värdet borde vara samma för alla rader i kontrollgruppen.

```
this.MakeDataTypeSelect = function (value) {
    var s = $("<select/>").addClass("DataType");
    s.append($("<option value='0'></option>"));
    s.append($("<option value='1'>String</option>"));
    s.append($("<option value='2'>Integer</option>"));
    s.append($("<option value='3'>Float</option>"));
    s.append($("<option value='4'>Date</option>"));
    s.val(value);
    return s;
};
```

```
this.MakeCardIdSelect = function (value) {
    var s = $("<select/>").addClass("CardId");
    for (var i = 0; i < data.Cards.length; i++) {
        s.append($("<option value='" + data.Cards[i].CardId + "'>" + data.Cards[i].Name + "</option>"));
    }
    s.val(value);
    return s;
};
```

Figur 34. Metoder för att skapa rullgardinsmenyer för kolumnerna DataType och CardId i HTML-tabellen.

För kolumnerna CardId och DataType skapas rullgardingsmenyer med metoderna *MakeDataTypeSelect* och *MakeCardIdSelect* (se Figur 34).

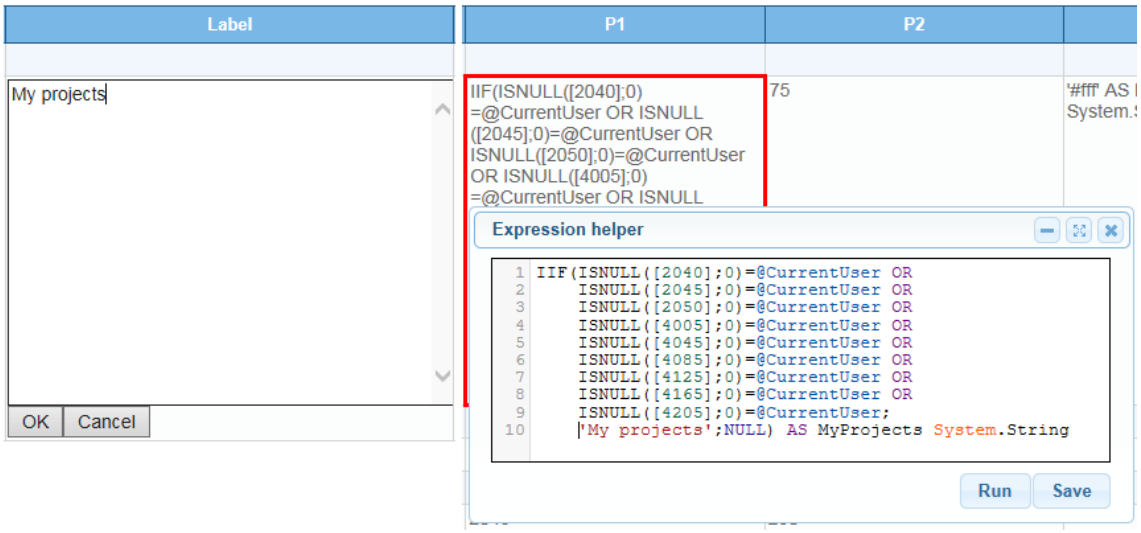
Resten av HTML-tabellen fylls med värdena från tabellen *control* mha. metoden *jQuery.append* som lägger in HTML mellan elementets begynnelse- och sluttagg. Alla fält associeras med tillhörande ControlId och DataId mha. metoden *jQuery.data* (se sista raderna i Figur 33).

```
    table.find('tr:visible:even').removeClass('alternate');
    table.find('tr:visible:odd').addClass('alternate');

    table.find(".Property_1, .Property_2, .Property_3, .Property_4").on("click", function (e) {
        config.editProperty($(this));
    });

    table.find("td").not(".readonly, .Property_1, .Property_2, .Property_3, .Property_4").editable(function (value, settings)
    {
        config.updateControlField($(this), value);
        return value;
    },
    {
        type: 'textarea',
        cancel: 'Cancel',
        submit: 'OK',
        placeholder: '',
        event: 'click'
    });
};
```

Figur 35. Metoden ChangeControlGroup del 2.

Efter att alla rader lagts till färgas varannan rad ljusblå mha. en CSS-klass med namnet *alternate* (se Figur 35). En musklick-händelse för Property 1-4 anropar metoden *editProperty* som öppnar innehållet i *Expression helper* (eftersom innehållet ofta är T-SQL-uttryck). Det valda fältet får en röd ram för att visa att det är öppnat (se Figur 36). Övriga fält använder sig av *Jeditable* som initialiseras med metoden *editable* och har en händelse-funktion samt en inställningstabell som parametrar. I inställningstabellen

definieras att ett textarea-element skall användas då en musklicks-händelse sker. Genom att musklicka på ett fält i tabellen kan man redigera innehållet rakt i tabellen.



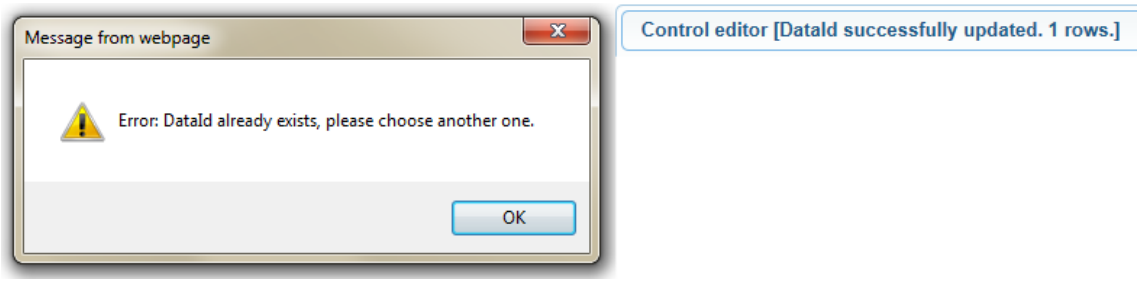
Figur 36. Fälten P1-P4 öppnas i Expression helper medan resten av fälten redigeras rakt i tabellen mha. Jeditable.

Då man musklickar på Save-knappen i *Expression helper* eller OK-knappen under textarean som *Jeditable* skapar anropas metoden *updateControlField*. Metoden *updateControlField* anropar *ConfigUpdateControlItem* mha. AJAX.

```
this.updateControlField = function (field, value) {
    $.ajax({
        type: "POST",
        url: window.location.pathname.split('/').pop() + "/ConfigUpdateControlItem",
        data: "{ controlId: " + field.data("ControlId") + ", dataId: " + field.data("DataId") + ", property: " + JSON.stringify(field.attr("class")) + ", value: " + JSON.stringify(value) +  "}",
        dataType: "json",
        contentType: "application/json; charset=utf-8",
        success: function (response) {
            var updateInfo = JSON.parse(response.d);
            config.updateControl(updateInfo);
        },
        error: function (err) { alert(err.responseText); }
    });
};
```

Figur 37. Metoden updateControlField.

Såsom Figur 37 visar anropas *ConfigUpdateControlItem* med parametrarna ControlId, DataId, fältets namn och fältets värde. Back-endkoden uppdaterar kontrollen och skickar tillbaka ett *UpdateInfo-objekt* som innehåller information om vad som har uppdaterats och om något fel uppstått. Detta *UpdateInfo-objekt* angivs som parameter till metoden *updateControl* som uppdaterar objekttabellen *config.currentControl* och användargränssnittet.



Figur 38. Exempel på felmeddelande och statusmeddelande i ”Control editor”-fönstret.

Om något går fel, t.ex. en ControlId eller DataId kollision, visar verktyget en JavaScript-påminnelse (se Figur 38). Om allting går rätt till uppdateras rubriken i *”Control editor”*-fönstet.

```
//One click triggers the ControlId dialog, double click takes you directly to the SQL
$("#btnSQL").button().on("click", function (e) {
    if (config.currentControls === null) return;
    config.sqlClicks++;

    if (config.sqlClicks === 1) {

        config.sqlTimer = setTimeout(function () {

            $('<div title="ControlId"><textarea style="width: 170px; height: 20px;">' + config.currentControls[0].ControlId + '</textarea></div>').dialog({
                resizable: false,
                modal: true,
                width: 200,
                height: 130,
                open: function () {
                    $(this).find("textarea").select();
                },
                buttons: {
                    "Generate": function () {
                        $(this).dialog("close");
                        config.GenerateSQL($(this).find("textarea").val());
                    }
                }
            });

            config.sqlClicks = 0;

        }, 700);

    } else {

        clearTimeout(config.sqlTimer);

        config.GenerateSQL();

        config.sqlClicks = 0;
    }
    e.preventDefault();
}).on("dblclick", function(e) { e.preventDefault() });
```

Figur 39. Musklicks-händelse för SQL-knappen. Ett musklick öppnar ControlId-dialogen som låter användaren ändra ControlId (och automatiskt DataId) för en SQL-förfrågning som genereras av kontrolldatan. Dubbelmusklick genererar direkt en SQL-förfrågning med samma ControlId som den valda kontrollen har.

Såsom visas i Figur 39 används en JavaScript-timer och en variabel som räknar musklick för att åstadkomma två skilda händelser för musklick samt dubbelmusklick på samma knapp.

Då användaren musklickar på SQL-knappen (se Figur 31) inkrementeras räknarvariabeln *sqlClicks*. Om *sqlClicks* är 1 (då användaren musklickat första gången på knappen) anropas funktionen *setTimeout* med en anonym funktion och ett heltalsvärdet som parametrar (se Figur 39). Heltalsvärdet, i detta fall 700 millisekunder, anger en fördröjning innan den anonyma funktionen exekveras. Funktionen *setTimeout* returnerar ett ID som sparas i variabeln *sqlTimer*.

I detta skede kan två saker hända:

1. **Användaren väntar 700 millisekunder.** Den anonyma funktionen exekveras, ControlId-dialogen (se Figur 40) skapas med metoden *jQuery.dialog* och variabeln *sqlClicks* nollas. Dialogen innehåller ett textarea-element med den valda kontrollens ControlId och en Generate-knapp. Då användaren musklickar på Generate-knappen stängs ControlId-dialogen och metoden *GenerateSQL* anropas.
2. **Användaren musklickar på SQL-knappen pånytt inom 700 millisekunder.** Variabeln *sqlClicks* inkrementeras och jämförs med värdet 1. *If-satsen* är osann och *else-blocket* exekveras. Funktionen *clearTimeout* anropas med ID-variabeln *sqlTimer* och timern avbryts (ControlId-dialogen visas aldrig). Metoden *GenerateSQL* anropas och variabeln *sqlClicks* nollas.

Figur 40. ControlId-dialog. Val av ControlId före generering av SQL.

Figur 41. Färdigt genererat skript SQL INSERT av vald kontroll.

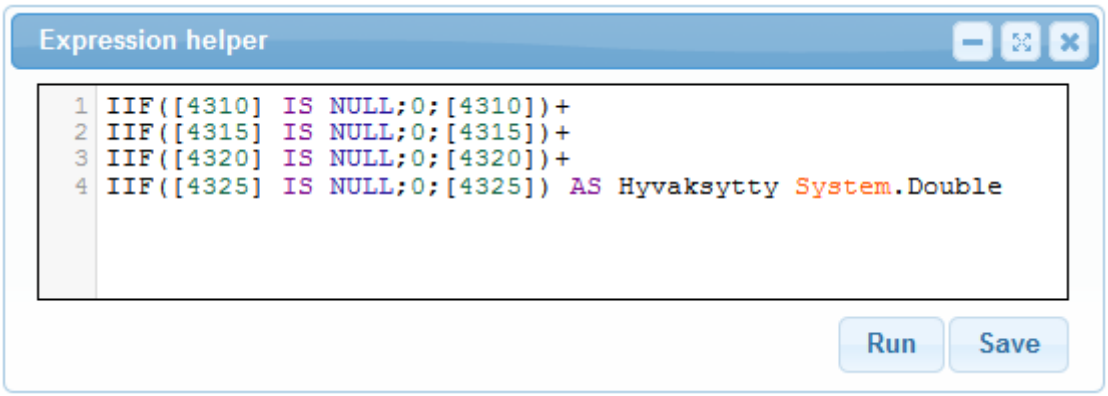
I Figur 41 visas en färdigt genererad SQL-förfrågning i dialogen *SQL dump* som blir synlig då metoden *GenereateSQL* anropats, dvs. efter dubbelmusklick på SQL-knappen i *Control editor* eller efter musklick på Generate-knappen i ControlId-dialogen. Orsaken till detta är att man inte alltid behöver ändra på ControlId och med denna funktionalitet undviker man ett extra steg. Väntetiden på 0,7 sekunder (700 millisekunder) är någonting som kan ändras om det efter mera testning känns trögt.

```
this.GenerateSQL = function (id) {
    var sqldata = $.extend(true, [], config.currentControls);
    var oldId = config.currentControls[0].ControlId;
    var newId = parseInt(id, 10);

    if (id !== undefined) {
        for (var i = 0; i < sqldata.length; i++) {
            sqldata[i].ControlId = newId;
            sqldata[i].DataId = (sqldata[i].DataId - (oldId * 10)) + (newId * 10);
        }
    }

    for (var i = 0; i < sqldata.length; i++) {
        if (sqldata[i].DataType == 0) sqldata[i].DataType = null;
        if (sqldata[i].Property_1 !== null) sqldata[i].Property_1 = sqldata[i].Property_1.replace(/'/g, "''");
        if (sqldata[i].Property_2 !== null) sqldata[i].Property_2 = sqldata[i].Property_2.replace(/'/g, "''");
        if (sqldata[i].Property_3 !== null) sqldata[i].Property_3 = sqldata[i].Property_3.replace(/'/g, "''");
        if (sqldata[i].Property_4 !== null) sqldata[i].Property_4 = sqldata[i].Property_4.replace(/'/g, "''");
    }
    var output = squel.insert().into(          ).setFieldsRows(sqldata).toString().replace(/\), /g, "),\n").replace("VALUES ", "VALUES\n");
    $('<div title="SQL dump"><textarea style="width: 1160px; height: 580px">' + output + '</textarea></div>').dialog({
        resizable: false,
        modal: true,
        width: 1200,
        height: 700,
        open: function () {
            $(this).find("textarea").select();
        }
    });
}
```

Figur 42. Metoden GenerateSQL som använder sig av Squel.js.

I metoden *GenerateSQL* (se Figur 42) kopieras tabellen *currentControls* mha. metoden *jQuery.extend* till *sqldata*, eftersom alla variabler i JavaScript bara kopieras enligt hänvisning (*eng. by reference*) och inte själva värdet (*eng. by value*). Om användaren gett ett nytt ControlId överskrivs det och DataId omräknas. Alla enkla citationstecken i Property 1-4 ersätts med två enkla citationstecken för att unvika felmeddelandet *Error: 105 Unclosed quotation mark after the character string '*. Detta beror på att alla teckensträngar i en INSERT-förfrågning är omgivna av enkla citationstecken. Squel.js används för att skapa en INSERT-förfrågning direkt från tabellen *sqldata*. Den resulterande teckensträngen formateras mha. *regular expression* genom att lägga till radbyten *\n* efter varje avslutande parentes *)* och teckensträngen *VALUES*. Resultatet visas i dialogen *SQL dump* (se Figur 41).

3.2.4 Expression helper



Figur 43. Expression helper.

Expression helper används för redigering av fälten Property 1-4 i *Control editor*. Precis som i *Panel content* används CodeMirror, men i SQL-läge (se Figur 44). Uttryck (*eng. Expressions*) är korta T-SQL-förfrågningar som används för dynamiskt innehåll i kontroller, jämförelse av olika värden, uträkning av färg eller summering osv. CodeMirror läget är *x-mssql* som står för Microsoft SQL alltså T-SQL (Transact-SQL). Inställningen *matchBrackets: true* gäller också hakparenteser [ ] vilket är viktigt i T-SQL-uttryck.

```
this.expression = CodeMirror.fromTextArea(document.getElementById("Expression"), {
    mode: "text/x-mssql",
    lineNumbers: true,
    matchBrackets: true,
    viewportMargin: Infinity
});
this.expression.setSize(470, 200);
this.expression.on("change", function (cm, change) {
    config.expressionDirty = true;
});
```

Figur 44. CodeMirror-initialisering i Expression helper.

```
$("#btnExpDebug").button().on("click", function (e) {
    $.ajax({
        type: "POST",
        url: window.location.pathname.split('/').pop() + "/ConfigDebugExpression",
        data: "{ input: " + JSON.stringify(config.expression.getDoc().getValue()) + "}",
        dataType: "json",
        contentType: "application/json; charset=utf-8",
        success: function (response) {
            alert(response.d);
        },
        error: function (err) { alert(err.responseText); }
    });
    e.preventDefault();
});

$("#btnExpSave").button().on("click", function (e) {
    config.propertyField.html(config.expression.getDoc().getValue());
    config.updateControlField(config.propertyField, config.propertyField.html());
    config.clearExpression();
    config.expressionWindow.window("minimize");
    e.preventDefault();
});
```

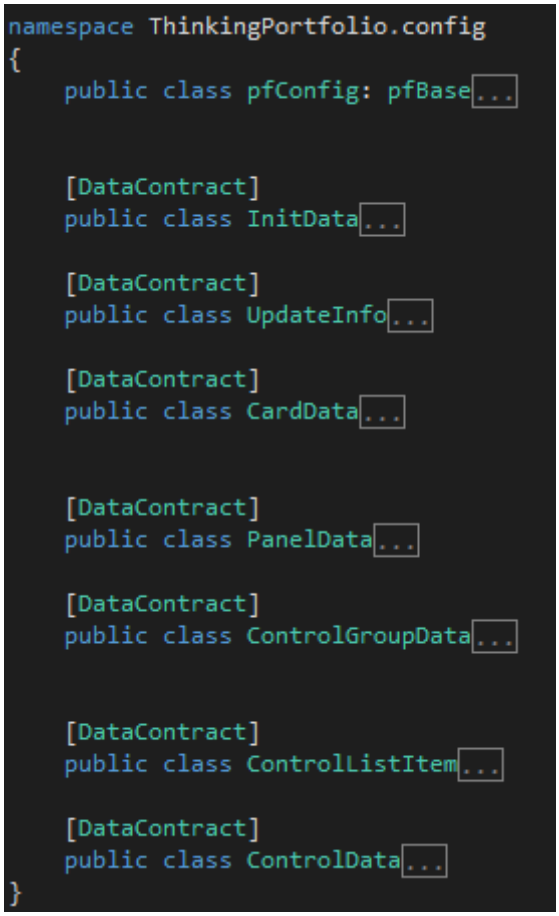
Figur 45. Musklickhändelser för knapparna i Expression helper.

Då användaren musklickar på knappen Run (btnExpDebug) anropas *ConfigDebugExpression* som tar emot teckensträngen i textfältet *expression* (se Figur 45). Resultatet visas i en JavaScript-påminnelse.

Knappen Save (btnExpSave) uppdaterar det ursprungliga Property-fältet i *Control editor* och anropar metoden *updateControlField* som sköter om uppdateringen av värdet i databasen. Till slut borttas innehållet i textfältet *expression* och *Expression helper* minimeras.

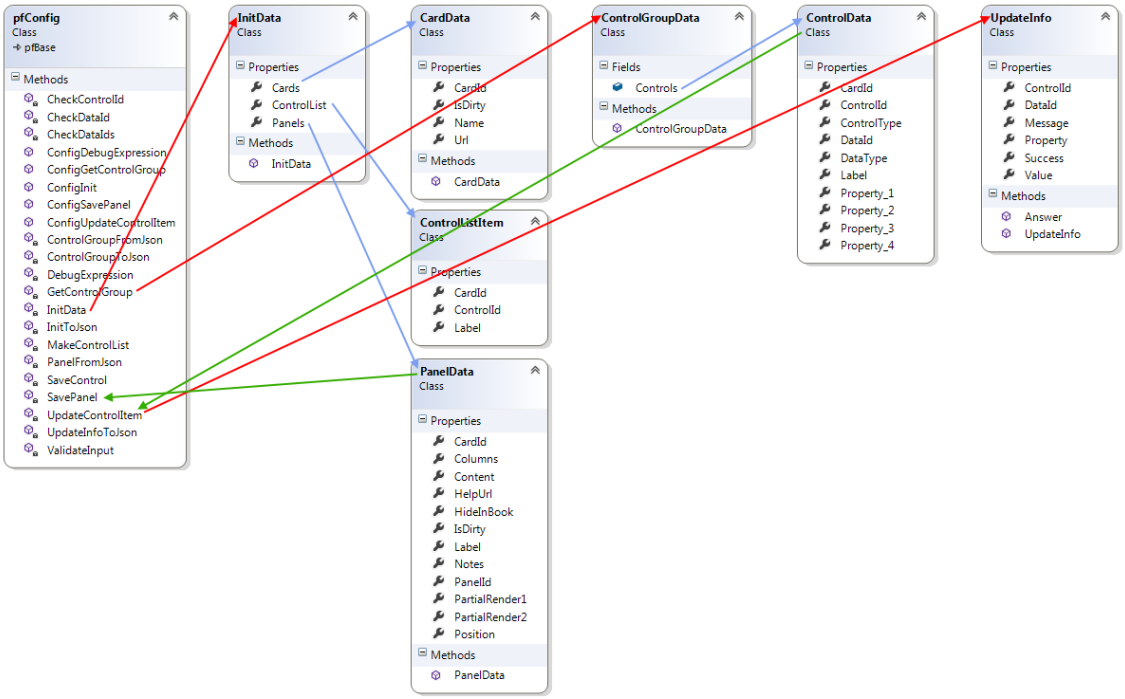
3.3 Back-end

Back-endkoden finns i en fil med namnet pfConfig.cs. Dessutom finns några kodrader i filen basepage.cs för att göra AJAX-anropen möjliga. Den huvudsakliga klassen pfConfig baserar sig på pfBase, en Thinking Portfolio basklass som innehåller sessionsvariabler mm.



Figur 46. pfConfig.cs överblick, 782 rader.

Såsom visas i Figur 46 innehåller pfConfig.cs flera klasser med taggen *DataContract*. Dessa klasser används för kommunikation med front-end. Huvudklassen pfConfig innehåller metoder som hämtar data och serialiserar objekt i *DataContract-klasser* till JSON-strängar. Serialiseringen använder .NET-ramverkets inbyggda klass *DataContractJsonSerializer*.



Figur 47. Klassdiagram på alla back-end klasser. De blåa pilarna pekar på vilken klass tabellens objekt är, t.ex. InitData.Cards är en tabell med CardData objekt. De röda pilarna går från metoden till returvärdets klass och de gröna pilarna går från parameterns klass till metoden, t.ex. metoden pfConfig.UpdateControlItem() tar emot ett ControlData objekt och returnerar ett UpdateInfo objekt.

I Figur 47 visas back-endkodens struktur. Alla metoder finns i klassen pfConfig. Metoderna som har prefixet *Config* är tillgängliga för anrop i front-endkoden. Omslagsmetoderna (*eng. wrapper methods*) börjar med att kolla att användaren är en såkallad *superuser*, så att endast vår firmas anställda har dataåtkomst (se Figur 48).

```
//Public methods, ¡HAS TO BEGIN WITH ADMIN CHECK!
public static string ConfigInit(int panelId)
{
    if (!AppCache.CurrentUser.IsSuperuser) throw new HttpException(401, "Illegal access");
    return InitToJson(InitData(panelId));
}
```

Figur 48. Metoden pfConfig.ConfigInit med säkerhetskontroll.

Såsom visas i Figur 48 anropar metoden *pfConfig.ConfigInit InitData* med samma parameter. Metoden *InitData()* returnerar ett *InitData-objekt* som sedan överförs till metoden *InitToJson* som serialiserar objektet och returnerar en JSON-teckensträng.

För att alla metoderna med prefixet *Config* skall vara tillgängliga för anrop från front-enden genom AJAX (HTTP-förfrågning av typen POST) finns det 5 omslagsmetoder i basepage.cs (se Figur 49). Dessa metoder har taggen WebMethod och måste vara statiska, vilket som betyder att alla metoder som anropas också måste vara statiska.

```
//Config communication
[WebMethod]
public static string ConfigInit(int panelId) { return pfConfig.ConfigInit(panelId); }

[WebMethod]
public static int ConfigSavePanel(PanelData panel) { return pfConfig.ConfigSavePanel(panel); }

[WebMethod]
public static string ConfigDebugExpression(string input) { return pfConfig.ConfigDebugExpression(input); }

[WebMethod]
public static string ConfigGetControlGroup(int controlId) { return pfConfig.ConfigGetControlGroup(controlId); }

[WebMethod]
public static string ConfigUpdateControlItem(int controlId, int dataId, string property, string value) { return
```

Figur 49. Metoder i basepage.cs med taggen WebMethod för att göra AJAX-anrop möjliga.

ConfigInit	Hämtar initialiseringsdata till konfigureringsverktyget som består av 3 tabeller: *Cards*, *ControlList* och *Panels*.
ConfigSavePanel	Sparar innehållet i och egenskaperna för en panel.
ConfigDebugExpression	Returnerar resultatet av ett T-SQL-uttryck från *Expression helper* på det valda projektet.
ConfigGetControlGroup	Returnerar en kontrollgrupp (kontroll) baserad på parametern controlId.
ConfigUpdateControlItem	Uppdaterar ett värde i Kontroll-tabellen (se 2.2.3 Kontroll).

Alla dessa metoder exekverar SQL-förfrågningar som läser från (SELECT) eller uppdaterar (UPDATE) rader i tabeller. I detta skede av konfigureringsverktygets utveckling finns inga metoder som lägger till (INSERT) eller tar bort (DELETE) rader i tabellerna.

```
private static InitData InitData(int panelId)
{
    var result = new InitData();
    using (var db = new pfDbConnection(AppCache.ConnectionString))
    {
        var command = db.CreateCommand("SELECT CardId, Name, Url FROM [redacted]");
        db.ExecuteReader(command);
        while (db.Reader.Read())
        {
            result.Cards.Add(new CardData
            {
                CardId = db.Reader.IsDBNull(0) ? 0 : db.Reader.GetInt32(0),
                Name = db.Reader.IsDBNull(1) ? null : db.Reader.GetString(1),
                Url = db.Reader.IsDBNull(2) ? null : db.Reader.GetString(2)
            });
        }
```

Figur 50. Exempel på inläsning av data från Kort-tabellen i metoden pfConfig.InitData.

I Figur 50 visas en del av metoden InitData som först ansluter sig till databasen mha. den inbyggda hjälpklassen *pfDbConnection*. En SELECT-förfrågning skapas och exekveras, koden går igenom alla rader och inläser resultaten i ett *CardData-objekt* som sparas i en tabell i *result-objektet*.

```
private static string UpdateControlItem(int controlId, int dataId, string property, string value)
{
    var ui = new UpdateInfo(controlId, dataId, property, value);
    int result;
    switch (property)
    {
        case "ControlId":
            int newControlId;
            if (int.TryParse(value, out newControlId))
            {
                //Check collision
                if (CheckControlId(newControlId))
                {
                    ui.Answer(false, "ControlId already exists, please choose another one.");
                    break;
                }
                using (var db = new pfDbConnection(AppCache.ConnectionString))
                {
                    var command = db.CreateCommand("UPDATE [redacted] SET ControlId=@NewId WHERE ControlId=@ControlId");
                    command.Parameters.Add(db.CreateParameter("@ControlId", controlId));
                    command.Parameters.Add(db.CreateParameter("@NewId", newControlId));
                    result = db.ExecuteNonQuery(command);
                }
                if (result > 0)
                {
                    ui.Answer(true, "ControlId successfully updated. " + result + " rows.");
                }
                else
                {
                    ui.Answer(false, "Unknown error, no rows updated.");
                }
            }
            else
            {
                ui.Answer(false, "Not valid ControlId, please enter integer value.");
            }
            break;
```

Figur 51. Exempel på datauppdatering i Kontroll-tabellen med metoden pfConfig.UpdateControlItem.

I Figur 51 visas uppdatering av värdet ControlId med metoden *UpdateControlItem*. Teckensträngen *value* konverteras till ett heltalsvärde och metoden *CheckControlId* anropas för att kolla att inga kollisioner sker. Om inga fel uppstår skapas en UPDATE-förfrågning vars parametrar (@ControlId och @NewId) ersätts med rätta värden innan förfrågningen exekveras. Till slut returneras *UpdateInfo-objektet ui* med datamedlemmarna Success och Message vilka tidigare fått sina värden från anropet till metoden *Answer()*. Dessa värden visas i *Control editor* (se Figur 38).

```
private static string UpdateInfoToJson(UpdateInfo data)
{
    string result = null;
    if (data == null)
        return result;

    using (var ms = new MemoryStream())
    {
        var serializer = new DataContractJsonSerializer(typeof(UpdateInfo));
        serializer.WriteObject(ms, data);

        using (var sr = new StreamReader(ms))
        {
            ms.Position = 0;
            result = sr.ReadToEnd();
        }
    }
    return result;
}
```

Figur 52. Exempel på JSON-serialisering med DataContractJsonSerializer.

Alla metoder som besvarar AJAX-anropet med ett objekt dvs. *ConfigInit*, *ConfigGetControlGroup* och *ConfigUpdateControlItem* måste serialisera objektet till en JSON-teckensträng (se Figur 52).

4 RESULTAT

För att kunna jämföra tidsanvändningen utan och med konfigureringsverktyget konfigurerade jag en IT-systemportfolio. Portfolions storlek är ungefär samma som en IT-systemportfolio jag gjort sommaren 2014. För att göra saken lättare och klarare kallar jag den nya portfolion B, som gjorts med det nya konfigureringsverktyget. Den gamla portfolion kallar jag A och har tidigare gjorts utan konfigureringsverktyget.

Det sammanlagda timantalet för A var 96, medan B hittills bara krävt 29 timmar. Det låter som om konfigureringverktyget skulle vara otroligt bra, men man måste ta en del aspekter i beaktande:

- B är endast i pilotfas dvs. testning-skede. För att vara 100% färdig för användning uppskattar jag att portfolion ännu behöver 12 timmar arbete. För att räkna med det värsta kan man lägga till 8 timmar för bugg-fixar och små ändringar som kunden önskar. Detta skulle betyda 49 timmar för B.
- B baserar sig på A-portfolions datamodell, vilket betyder att en del av jobbet redan var gjort. A baserades såklart på en IT-system portfoliomodell som jag gjort några månader innan. För att göra det rättvist kan vi ta bort 8 timmar från A för en SmartForm som gjordes under timmarna för A, men som också används av B. Vi tar också bort 4 timmar som jag arbetade på färgscheman för A. Detta betyder att A är 84 timmar.
- Erfarenhet. Det är värt att nämna att jag fått nästan 1 år mera erfarenhet mellan konfigureringen av dessa portfolion, vilket betyder att jag gör färre misstag och vet hur jag gör saker på snabbare sätt.

Tabell 6. Tidsanvändning och skillnad före och efter konfigureringsverktyget

	A (före)	**B (efter)**	**Skillnad**
Oförändrade timmar	96	29	67
Justerade timmar	84	49	35

Som man ser i Tabell 6 så går konfigureringen mycket snabbare än tidigare. Man kan argumentera hur stor del konfigureringsverktyget bidrar med och hur mycket omständigheterna och ökad erfarenhet påverkar resultaten. Om man endast beaktar de

oförändrade timmarna i Tabell 6 skulle det betyda att konfigureringsprocessen gjorts ungefär 70% effektivare. Detta är dock enligt mig ett orealistiskt resultat och de justerade timmarna är närmare sanningen med ungefär 40%.

5 SLUTSATSER OCH VIDAREUTVECKLING

Konfigureringsverktygets fördelar:

- Snabbare att flytta kontroller och innehåll mellan paneler. Navigering mellan paneler utan omflyttning mellan kort i portfolion eller omladdning av webbsidan.
- Lätt att märka om kontroller har felaktig CardId och enkelt att åtgärda med *en* rullgardinsmeny istället för t.ex. ändring av 12 värden.
- Tillåter redigering av kontroller utan sökning efter rätta rader i tabellen i *Microsoft SQL Server Manager* (och utan extra steg för att logga in på servern om databasen är i produktion). Kollisionskontroll av ControlId och DataId är också en stor fördel och betyder att man gör färre misstag under konfigureringsarbetet.
- Kopiering av kontroller som genererar SQL-förfrågningar. Man behöver inte längre komma ihåg en ControlId, använda *SMSS Tools* med en ”SELECT .. WHERE”-förfrågning, kopiera den genererade SQL-förfrågningen och ändra alla ControlId och DataId. Allt detta görs istället automatiskt genom att välja kontroll från kontroll-listan i *Control editor*, musklicka på SQL-knappen och skriva in vilken ControlId den nya kontrollen borde ha.
- Lättare att redigera och formatera HTML och CSS för panelinnehåll och T-SQL-uttryck i kontrolldata pga. färgad syntax och indentering.
- Fungerar som en plattform för vidareutveckling av olika verktyg för konfgurering.

Exempel på funktionalitet som kunde utvecklas:

- Möjligheten att kunna lägga till och ta bort kort, paneler och kontroller. Detta skulle betyda att man mycket sällan skulle behöva externa program, såsom *Microsoft SQL Server Manager*, vilket skulle bidra till ett bättre arbetsflöde.
- Snabbtangenter för olika funktioner såsom minimering, maximering eller hopp mellan fönster. Personligen uppskattar jag snabbtangenter i alla program, allting blir smidigare ju färre musklick och musrörelser behövs.
- Ikoner för alla fönster för att visuellt göra verktyget lättare för ögat.
- *pfEntityListColumn* Property 2 *bitmask helper*. Kontrollen *pfEntityListColumn* som används för överblick av alla projekt i en portfolio har en samling inställningar som representeras som en bitmask. Det skulle vara lättare att se en

samling kryssrutor med namn istället för att räkna ut att 216 betyder 128 + 64 + 16 + 8 och sedan kolla vad dessa heltal betyder i koden.

- Kontrollvisualisering. Det skulle vara klarare att direkt kunna koppla en ControlId till en bild eller ikon av kontrollen, så att man inte tappar bort sig i verktyget och i misstag ändrar orätt kontroll.
- Kontroll-mallar. Detta skulle kräva en eller två nya tabeller, men skulle möjliggöra att man med tiden kan samla alla olika kontroller och bara plocka de man behöver från en plats istället för att gå igenom flera databaser i jakt efter rätt kontroll.
- Kontrollfiltrering. Just nu filtreras kontroll-listan i *Control editor* automatiskt med den valda panelens CardId. Det skulle vara lättare att hitta rätt kontroll om man t.ex. kunde välja att filtrera per panel eller kort (som det är just nu) och ingen filtrering.
- Expression-mallar. T-SQL-uttryck används i många kontroller för dynamiskt innehåll och det skulle vara bra om man hade mallar för vanliga uttryckssatser såsom summering av ett antal värden eller jämförelse av ett värde med andra värden osv.

Konfigureringsverktyget är en bra början och har många fördelar för konfigureringsarbetet. Dess största fördel är att det är en stadig grund för vidareutveckling.

KÄLLOR

Thinking Portfolio Project Portfolio Whitepaper Q1/2015
Tillgänglig: http://www.thinkingportfolio.com/wp-content/uploads/2015/02/Thinking-Portfolio-Project-Portfolio-Whitepaper-Q1-2015-english.pdf, Hämtad: 24.3.2015

CodeMirror: User manual and reference guide
Tillgänglig: https://codemirror.net/doc/manual.html, Hämtad: 25.3.2015

Simone documentation
Tillgänglig: http://cezarykluczynski.github.io/simone/docs/index.html, Hämtad 3.4.2015

Jeditable – Edit In Place Plugin For jQuery
Tillgänglig: http://www.appelsiini.net/projects/jeditable, Hämtad 30.3.2015

Squel.js – SQL query string builder for JavaScript
Tillgänglig: http://hiddentao.github.io/squel/index.html, Hämtad 26.4.2015

Methvin, Dave. 2014. The State of jQuery 2014
Tillgänglig: http://blog.jquery.com/2014/01/13/the-state-of-jquery-2014/, Hämtad 29.4.2015