INGENJÖRSHÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

DATAORIENTERADE KOMPONENTER FÖR ADOBE FLASH

Anton Frennevi

EXAMENSARBETE 2007

DATATEKNIK

Postadress:
Box 1026
551 11 Jönköping

Besöksadress:
Gjuterigatan 5

Telefon:
036-10 10 00 (vx)

INGENJÖRSHÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

DATAORIENTERADE KOMPONENTER FÖR ADOBE FLASH

DATA-ORIENTED COMPONENTS FOR ADOBE FLASH

Anton Frennevi

Detta examensarbete är utfört vid Ingenjörshögskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Magnus Schoultz

Omfattning: 10 poäng

Datum:

Arkiveringsnummer:

Postadress:
Box 1026
551 11 Jönköping

Besöksadress:
Gjuterigatan 5

Telefon:
036-10 10 00 (vx)

Abstract

Flash is a development environment for creating interactive presentations and games for the Internet and CD-media. The program creates files of the SWF format which supports vector and raster graphics, the script language ActionScript and bi-linear streaming of sound and video.

A database is a collection of data organized after a systematic scheme. A database can be consulted to answer questions such as selection. Databases give many possibilities; a game can store statistics etcetera.

A file of the SWF format cannot consult a database directly. For consultation a middle layer is required, a so called middleware. The creation of a middleware is uncomplicated but time consuming and extensive work.

In Flash components are available for use. Reuse of previously defined objects is to be preferred and by doing so development time can be saved. Components allow for reuse and are easily utilized.

The purpose of this diploma report is therefore to develop a handful of components to facilitate the development of database bound Flash media.

The work has been practically oriented. By collecting facts and thereafter by setting collected facts to practical use, three separate components for Flash have been developed.

During this exam I’ve found numerous methods for making database consultation possible through SWF files. Furthermore I’ve learned a great deal as to how components work, how they’re constructed and how they can be used.

Sammanfattning

Flash är en utvecklingsmiljö i vilken bland annat interaktiva presentationer och spel kan utvecklas för Internet och CD-media. Programmet skapar filer av SWF-formatet vilket stöder vektor- och rastergrafik, skriptspråket ActionScript och dubbelriktad strömning av ljud och video.

En databas är en samling data lagrad efter ett systematiskt schema. En databas kan således konsulteras för att besvara frågor så som selektion. Databaser ger flera möjligheter; spel kan lagra statistik etcetera.

En fil av SWF-formatet kan inte direkt konsultera en databas. För konsultation krävs ett mellanliggande lager, en så kallad mellanhand. Att skapa en mellanhand är okomplicerat dock ett tidskrävande och omfattande arbete.

I Flash finns komponenter tillgängliga att använda. Återanvändning av tidigare definierade objekt är att föredra. På så vis kan utvecklingstid sparas. Komponenter tillåter återanvändning och är lätta att använda.

Syftet med detta examensarbete var således att utveckla en handfull komponenter för att underlätta arbetet vid utveckling av databasbunden Flash-media.

Arbetet har varit praktiskt orienterat. Genom faktainsamling och därefter praktisk tillämning av insamlad information har tre skilda komponenter för Flash utvecklats.

Jag har under arbetets gång funnit flera olika metoder för att i SWF-filer möjliggöra databaskonsultation. Därtill har jag lärt mig mycket om hur komponenter fungerar, hur de är uppbyggda samt hur de kan användas.

Nyckelord

Macromedia, Adobe, Flash, Flash Remoting, komponent, databas, kommunikation

Innehållsförteckning

1 Inledning

Denna rapport är ett led i min utbildning till dataingenjör vid Tekniska Högskolan i Jönköping. Rapporten är en del av det examensarbete på tio poäng vilket utförts under utbildningens sista del.

1.1.1 Företagssammanslagning

December 2005 kom *Macromedia* och *Adobe* att bli ett och samma företag. (1) I denna rapport har jag, till följd av uppköpet, valt att använda varumärket Adobe.

1.2 Bakgrund

Flash är en utvecklingsmiljö för att utveckla interaktiva applikationer, presentationer och inte minst spel. Programmet används i stor utsträckning för webbutveckling. (2)

I programmet kan multimedia av *SWF-formatet* skapas. Formatet stöder vektor- och rastergrafik, ett skriptspråk vid namn *ActionScript* samt dubbelriktad strömning av ljud och video. (2)

Filer av SWF-formatet saknar direkta medel för kommunikation med databaser. Det finns ett flertal möjliga lösningar till detta kortkommande; samtliga innefattar ett mellanliggande lager.

Detta mellanliggande serverlager kan refereras till som en *mellanhand* och kan programmeras i en rad olika språk så som *ColdFusion* och *C#*. (3)

1.2.1 Komponenter

Det finns i Flash en handfull *komponenter* för användaren att bruka. Dessa syftar till att underlätta utvecklingsarbetet genom återanvändning av skalbar ActionScript-kod. Komponenterna är utformade på så sätt att de enkelt kan infogas i ett projekt och konfigureras så att de sömlöst blir en del av den applikation i vilken de inskjutits.

Exempel på komponenter vilka medföljer programmet är rullningslisten, kalendern, multimediakontrollen och textrutan. Dessa är exempel på objekt vilka det vore önskvärt att återanvända i flera separata projekt. Komponenter möjliggör detta. (4)

1.3 Syfte och mål

Syftet med detta examensarbete är att utveckla tre databasdrivna komponenter för utvecklingsmiljön Flash. Komponenterna syftar till att underlätta arbetet vid utveckling av databasorienterad Flash-media.

Tre skilda komponenter skall utvecklas:

1. En komponent vilken kan selektera, redigera, radera och infoga data i en databas.
2. En komponent speciellt utformad för användarkontroll mot en databas. Komponenten skall kunna användas för att implementera användarrestriktioner i SWF-filer.
3. En komponent vilken kan presentera data bunden till en specifik rad i en databas, med kopplingsmöjlighet till komponentklassen i listans första punkt.

Komponenterna ska vara förhållandevis enkla att använda och möjliga att konfigurera om för mer specifika ändamål. Samtliga skall vara enkla att installera och ska automatiskt integreras i programgränssnittet.

Då jag sedan drygt ett år tillbaka är egen företagare ämnar jag förlägga mitt examensarbete inom min egen företagsverksamhet. Jag har genom företaget ett behov av tidigare nämna komponenter, vilka betydligt skulle minska arbetsbördan vid utveckling.

Det finns idag, till min kännedom, inga färdiga komponenter likt de specificerade ovan. Metoden för att kringgå avsaknaden av direkt databaskommunikation, använd inom detta examensarbete, är väletablerad. En generell skalbar tillämpning i komponentform saknas dock.

Jag vill poängtera att jag sedan tidigare aldrig använt mig av eller utvecklat komponenter i Flash. I och med detta syftar examensarbetet även till att lära mig mer om hur komponenter fungerar, hur de är uppbyggda samt hur de kan användas.

Det övergripande målet är tre kompletta komponenter vilka lätt kan installeras och konfigurers efter de ändamål för vilka de utvecklats.

1.4 Avgränsningar

Detta examensarbete syftar inte till att utveckla nya kommunikations-möjligheter mellan en SWF-fil och en databas. Utvecklade komponenter bygger på en redan etablerad metod, med vissa mindre tekniska skillnader. Arbetet pekar på en påtaglig begränsning i SWF-formatet och presenterar därtill en av flera möjliga lösningar.

Rapporten sträcker sig längre än utvecklingsmiljön Flash. Två av de komponenter, som presenteras i denna rapport, bygger även på serverfunktionalitet. Berörda funktioner kommer dock inte beskrivas i detalj då fokus ligger på Flash.

Jag kommer inte att behandla funktioner av mindre betydelse för utvecklade komponenter. För de funktioner vilka har större betydelse kommer redovisningar att lämnas. Viktigt att poängtera är att denna rapport ser till komponentutveckling i dess helhet.

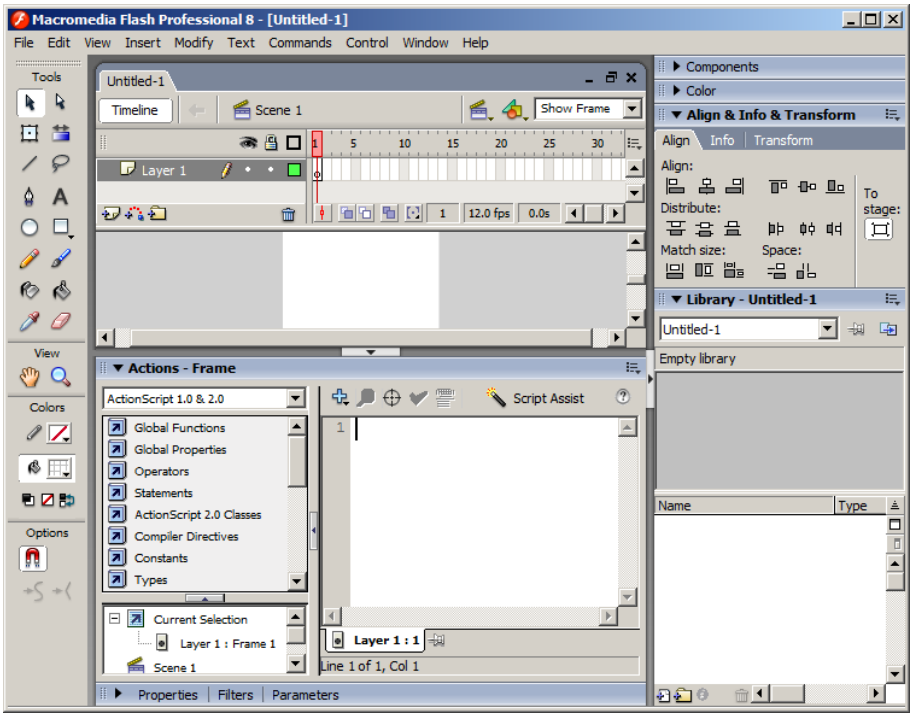
Önskas detaljerade beskrivningar av samtliga komponenters metoder och parametrar finns sådan information att läsa i komponentdokumentationen. Åtkomst till dessa dokument förutsätter att komponenterna installeras.

1.5 Disposition

Inledande avsnitt i denna rapport kommer att ge en mer detaljerad beskrivning av Flash. Därefter kommer använt tillvägagångssättet att redogöras för. Senare kommer utvecklade komponenter att presenteras. Avslutningsvis kommer en diskussion föras rörande möjlig vidareutveckling.

2 Teoretisk bakgrund

2.1 Adobe Flash



Figur 1. Programgränssnittet i Flash 8 Professional

Flash är ett animationsverktyg som idag utvecklas av Adobe. Programmet används bland annat för att skapa interaktiva presentationer och spel för Internet och CD-media.

Charlie Jackson, Jonathan Gay och Michelle Welsh startade januari 1993 ett mindre mjukvaruföretag vid namn *FutureWare*. Företagets första produkt var *SmartSketch*; ett ritprogram vars kärna var enkelhet. Idén var ett ritprogram lika enkelt att använda som papper och penna. Programmet vann dock mycket liten marknad trots dess innovativa utformning.

Internet växte snabbt under 1990-talet och företaget såg att möjlighet fanns att ta strid med *Shockwave*, vilket utvecklades av Macromedia. Två år senare, 1995, vidareutvecklades således SmartSketch till ett bildrutebaserat animationsverktyg. SmartSketch släpptes på nytt som *FutureSplash Animator*. Programmet köptes upp av Macromedia i december 1996 och släpptes senare under programnamnet Flash.

Nu, 11 år senare, har elva versioner av Flash släppts:

- Flash 1, november 1996
- Flash 2, juni 1997
- Flash 3, maj 1998
- Flash 4, juni 1999

- Flash 5, augusti 2000
- Flash MX, mars 2002
- Flash MX 2004, september 2003
- Flash MX 2004 Professional, september 2003
- Flash 8 Basic, september 2005
- Flash 8 Professional, september 2005
- Flash CS3 Professional, april 2007

Flash 9, vilket går under kodnamn *Blaze*, är för närvarande under utveckling. En betaversion av programmet finns dock att ladda hem och prova fritt. (2)

Figur 1 visar programgränssnittet i Flash 8 Professional. Till vänster i programgränssnittet finns en verktygspanel innehållande sexton verktyg vilka kan användas för att rita upp och modifiera grafiska objekt.

Gränssnittet mitt kallas för en *scen* och respresenterar den aktuella kanvasen. Det är på denna yta grafiska element kan ritas upp. Ovanför scenen finner vi scenens *tidslinje*. Tidslinjen tillåter för förändringar till scenens element över tid. Tidslinjen används således för animation. Animation kan ske manuellt eller genom programmering. Programmering sker med hjälp av ActionScript-kod vilken skrivs in i programmets *kodpanel*. Kodpanelen är placerad strax under scenen.

I programgränssnittets högra nedersta hörn finner vi *biblioteket*. Biblioteket innehåller lagrade element vilka enkelt kan dras in till scenen och därefter användas för animation. Dessa paneler är de vilka används mest vid utveckling i Flash.

2.1.1 ActionScript

Juni 2006 lanserades *Flash Player 9*. Med denna lansering kom även den senaste versionen av skriptspråket ActionScript; version 3.0. Flash Player 9 bygger på en ny *ActionScript Virtual Machine* där en *Just In Time*-kompilator översätter ActionScript-kod till maskinkod för maximal exekveringshastighet. (2)

ActionScript 3.0 skiljer sig helt ifrån sina föregångare. Den senaste versionen av skriptspråket ligger dock utanför ramarna för detta examensarbete.

ActionScript 2.0 är ett *objektorienterat* skriptspråk löst specificerat efter *ECMAScript 4*. ECMAScript är ett skriptspråk standardiserat av *Ecma International* specificerat i *ECMA-262*. Vanligtvis kallas språket *JavaScript* eller *JScript* efter dess två huvudsakliga tillämpningar.

Objektorienterad programmering är ett programmeringsparadigm där objekt används för att formge en applikation. Objektorienterad programmering kan ses som skapandet av ett objektkollektiv där varje enskilt objekt är kapabelt att ta emot, hantera och skicka data. Varje objekt kan ses som en enskild byggsten med en distinkt roll i applikationskollektivet. (5) Objektorienterad programmering utgör grunden för utformningen av komponenter i Flash.

2.1.2 SWF-formatet

Vid utveckling produceras filer av *SWF-formatet* (vilket inte är en akronym för *Shockwave Flash* (6)). SWF-formatet är ett vektorbaserat och därigenom skalbart filformat. Formatet kan hantera ljud, bild och skriptspråket ActionScript. (2)

Vid webbpublicering kräver filformatet att ett specifikt insticksprogram finns installerat på de klienter, vilka önskar spela upp det interaktiva materialet. Insticksprogrammet, *Flash Player*, fungerar som ett webbläsartillägg, men kan även användas som en projektor för att utanför webbläsaren ta del av material producerat i Flash. (2)

2.1.2.1 Begränsning till SWF-formatet

Som nämnts tidigare har SWF-formatet en betydande begränsning. Denna begränsning syftar detta examensarbete till att lösa. I rapportavsnittet *Syfte och mål* står att läsa att tre databasdrivna komponenter skall skapas.

Flash-media är av statisk natur; data kan inte lagras varaktigt då media körs lokalt genom Flash Player. ActionScript exekveras alltså lokalt hos de klienter vilka önskar ta del av det interaktiva materialet. (3)

2.2 Microsoft ASP.NET

ASP.NET är en uppsättning teknologier för utveckling av webbapplikationer från Microsoft. ASP.NET är en vidareutveckling av *Active Server Pages* (ASP) och del av .NET-plattformen. .NET-plattformen tillhandahåller robusta programmeringsbibliotek och verktyg användbara vid utveckling. (7)

2.2.1 C#

.NET innefattar det nya programmeringsspråket C# (uttalas C Sharp). Språket använder en syntax lik den av C och C++ och är på många sätt likt Java från Sun. C# är ett generellt språk väl anpassat för de allra flesta typer av applikationer. Språket är enkelt att använda och möjliggör snabb programmering av applikationer på hög nivå. Samtidigt som det även lämpar sig väl för programmering på låg- och systemnivå. (7)

2.2.1.1 Microsoft Visual Web Developer 2005

Microsoft Visual Web Developer 2005 är ett verktyg i vilket webbsidor och webbtjänster med fördel kan utvecklas. Programmet stödjer användandet av ASP.NET och tillåter för lokal exekvering av projekt genom användandet av en utvecklingsserver. Projekt kan skrivas i Visual Basic eller C#. (8) För detta examensarbete användes det senare.

Programmet finns att hämta hem och använda gratis.

2.3 SQL

Structured Query Language (SQL) är ett gränssnittsspråk för arbete mot relationsdatabassystem. (5) SQL finns i flera olika dialekter beroende på vilket databassystem som används. Språket utvecklades av IBM under 1970-talet och är idag en ISO- och ANSI-standard.

SQL är det språk vilket används av framtagna komponenter.

3 Genomförande

Examensarbetet inleddes med en omfattande informationssökning. Jag vände mig till en början främst till Internet och min privata bokhylla för den information jag var i behov av. Allteftersom arbete fortlöpte och problem påstöttes samlades ytterligare information för att hjälpa mig över påträffade hinder.

Till en början var mitt källkritiska ställningstagande rent praktiskt. De exempel jag ställdes inför prövades för att se huruvida informationen kunde anses riktig eller inte. Vid ett senare skede har jag även värderat information på en teoretisk nivå. När riktig information samlats kunde jag utefter teorin tillämpad i dessa sedan sålla bland källor med tveksamma exempel.

Den bokdatabas som finns tillgänglig för studerande vid Tekniska Högskolan användes också. Databasen visade sig mycket användbar och ett antal titlar, vilka berör detta examensarbetes område, har genom databasen tagits del av och senare refererats till.

Vad gäller informationssökning på Internet visade det sig mycket svårt att hitta idag aktuell information. Mycket av den information som fanns att hitta rörde äldre versioner av Flash. Tillvägagångssättet vid komponentprogrammering har med senare versioner av Flash förändrats radikalt. I och med detta visade sig mycket insamlad information oanvändbar.

Min främsta informationsskälla har varit programmets egna hjälpfiler och *LiveDocs*.

3.1.1 LiveDocs

Programdokumentationen för Flash finns tillgänglig på Internet, publicerad på en samlingsplats kallad LiveDocs. LiveDocs är kort en interaktiv dokumentationsdatabas där användare kan ge kommentarer till dokumentationen; en plattform för utbyte. (9)

3.2 Mjukvara

För att utföra detta examensarbete har fyra program använts; Flash, Extension Manager, Dreamweaver och Visual Web Developer.

Extension Manager är ett program vilket installeras tillsammans med Flash och används för att hantera utbyggnader och komponenter för flera programtitlar från Adobe; däribland Flash och Dreamweaver.

Visual Web Developer är som bekant ett program för webbutveckling. Dreamweaver är av samma programtyp och har använts då en utbyggnad finns till mjukvaran som tillåter för enklare dokumentation. Mer information rörande detta följer i nästa rapportavsnitt.

Alla program som använts finns att ladda hem och använda under en begränsad tid. Visual Web Developer är dock inte tidsbundet utan kan användas oförhindrat inom programmets licensavtal.

3.3 Komponentprogrammering

Den information vilken använts som underlag vid utveckling har samlats här. Informationen ger ett teoretiskt underlag när källkoden studeras; se bilagorna 2-7.

3.3.1 Användningsområden

Användandet av komponenter möjliggör separering av design- och programmeringsprocessen. Tidigare författad kod kan återanvändas i hög utsträckning och komponenter skrivna av andra utvecklare kan enkelt implementeras i egna applikationer. Kort innebär komponenter en lättnad av arbetsbördan vid utveckling.

Komponenter är mycket enkla att använda och importering och konfigurerig kan göras visuellt eller genom programkod.

Klassarv, vilket kommer att tas upp i ett senare stycke, innebär mindre filstorlekar för större projekt och möjliggör för ändringar att flöda genom en applikation. En förändring i en överordnad klass löper genom den hierarkiska strukturen vilket innebär att alla underordnade klasser till superklassen i fråga förändras därefter. Ytterligare fördelar till detta programmeringsparadigm tas upp senare i denna rapport.

(9)

3.3.2 Komponentklassen

En komponent består huvudsakligen av ett komponentobjekt och en komponentklass. Komponentobjektet och komponentklassen skapas separat för att sedan sammanfogas till en komplett komponent. Filerna kompileras sedan till en SWC-fil. (9) (4)

Komponentklassen utgörs av ett enkelt textdokument innehållande en uppsättning ActionScript. ActionScript är namnet på det skriptspråk vilket används i Flash. Skriptspråket har en för många familjär syntax då dess formatering och struktur har sitt ursprung i samma språkdefinition så som den av JavaScript och JScript. (9) (10)

Följande är ett exempel på hur en komponentklass kan skrivas. Observera att detta endast är ett exempel gällande komponenter vilka ärver från UIComponent. Mer om klassarv senare.

```
[Event("eventName")]

[IconFile("MyIcon.FileFormat")]
```

```
import mx.core.UIComponent;
class MyComponent extends UIComponent {

  function init():Void {
    super.init();
  }

  function createChildren():Void {
    size();
    invalidate();
  }

  function MyComponent() {
  }

  function draw():Void {
    super.draw();
  }

  function size():Void {
    super.size();
    invalidate();
  }

}
```

För korrekt tillämpning måste komponentklasser baserade på UIComponent innehålla de ovan deklarerade metoderna. (9)

3.3.2.1 Klassarv

Komponentklassen måste följa en fördefinierad struktur. Strukturen varierar något beroende på klassens överordnade komponentklass, så kallad *superklass*. Den underordnade komponentklassen kan i sin tur refereras till som *subklass*. När komponentklassen deklareras kan klassen placeras i en hierarkisk struktur och kan på så sätt ärva attribut och egenskaper från en eller flera redan deklarerade komponentklasser. Genom arv kan metoder definierade i en överordnad klass tillgängliggöras utan ytterligare programmering. (9) (11)

Användandet av super- och subklasser är ett objektorienterat koncept känt som *abstraktion*. (12) Förenklat innebär abstraktion att varje klass deklareras så generellt som möjligt och ger fördelar som:

- Flexibel kod
- Hanterbar och skalbar kod
- Kod kan omdefinieras i subklasser

Bland möjliga superklasser finns två basklasser; *UIObject* och *UIComponent*.

UIObject utgör första byggblock för alla grafiska objekt och möjliggör funktionalitet så som stilredigering, händelsehantering och storleksändrig genom skalning.

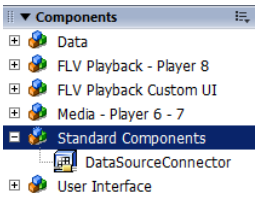
```
import mx.core.UIObject;
class MyComponent extends UIObject {
}
```

UIComponent utgör grunden för samtliga fördefinierade komponenter och möjliggör funktionalitet så som skalning, hantering av mus- och tangentbordshändelser samt upprättandet av tabbscheman. UIComponent är i realiteten en subklass till UIObject.

```
import mx.core.UIComponent;
class MyComponet extends UIComponent {
}
```

(4)

3.3.2.2 Komponentikoner



Figur 2. Komponentpanelen

Samtliga utvecklade komponenter representeras av en mindre ikon vilka grafiskt integrerats med utvecklingsmiljöns användargränssnitt, se *Figur 2*. När en komponent installerats finns den tillgänglig för användning genom programmets gränssnitt. Menytabbar för ändring av bland annat komponentvariabler genereras automatisk varför komponenter är enkla att installera och konfigurera.

Inom ramarna för detta examensarbete har fyra ikoner ritats. Komponentikoner med liknande funktionalitet har använts som utgångspunkt, för att på så sätt utnyttja redan etablerad funktionsförståelse.

Att länka samman en komponent och tillhörande komponentikon är enkelt. Kopplingen är en så kallad *metadatatagg*. Metadatataggarna beskriver komponentens huvudsakliga utformning och utöver ikonlänkning används dessa taggar bland annat för att deklarera komponenthändelser, skrivbara variabler och huruvida komponenten kan bindas till andra objekt eller inte. En komponentikon länkas enkelt genom att ange den relativa sökvägen till den ikon man önskar använda, innan deklarationen av komponentklassen.

```
[IconFile("MyIcon.FileFormat")]
```

(9)

3.3.2.3 Händelser

Även händelsedeklaration sker genom metadatataggar. Det finns tillfällen man önskar att ett specifikt block kod exekveras till följd av att en specifik händelse inträffar. För att koppla samman en händelse och extern kod krävs först en händelsedeklaration. En händelse deklareras som följer:

```
[Event("EventName")]
```

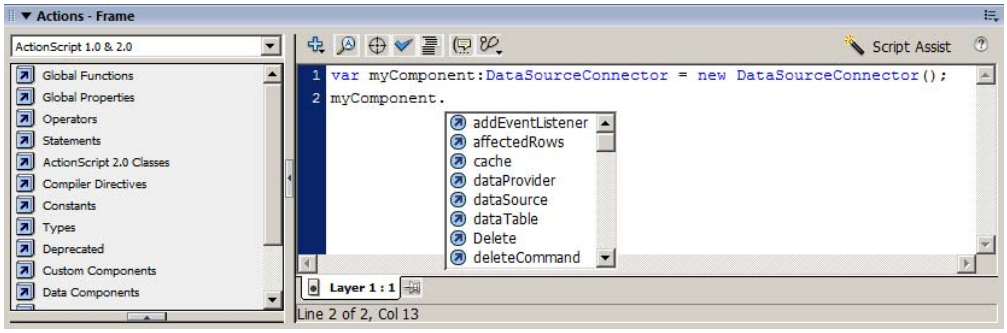
Tidigare deklarerades en händelse, men för att upprätta en faktisk koppling mellan en händelse och ett block kod krävs att komponenten programmerats att ge ifrån sig deklarerad händelse. Detta åstadkoms genom att använda en fördefinierad metod vid namn *dispatchEvent* vilken samtliga komponenter baserade på superklassen UIObject har i arv.

```
var event:Object = {target:Object, type:"EventName"};
component.dispatchEvent (event);
```

(9)

Detta utgör dock endast grunden för händelsehantering. Kännedom om hur händelser deklareras och inträffar är dock en fullgod bas att bygga vidare på. För komplett händelsedeklaration och emittering hänvisas ni till källkoden för utvecklade komponenter.

3.3.3 Kodtips och syntaxfärg

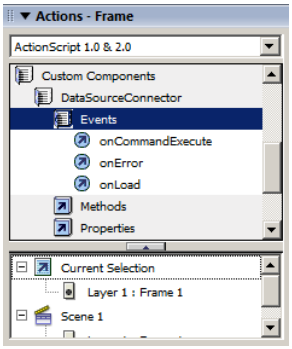


Figur 3. Kodreferenspanelen

Vid programmering i Flash finns möjlighet till kodtips och syntaxfärgning (se *Figur 3*). Dessa är visuella hjälpmedel vilka underlättar programmering i utvecklingsmiljön.

Att instruera utvecklingsmiljön att tillämpa syntaxfärgning och kodtips för egenutvecklade komponenter är förhållandevis enkelt. Ett enkelt XML-dokument författas, efter förbestämd utformning, där komponentspecifika metoder och egenskaper listas. Dokumentet placeras sedan vid installation i en av programmets systemmappar. Vid programstart läses dokumentet in. På så sätt har programmet den information som krävs för att ge de visuella hjälpmedel som önskas.

(13)



Figur 4. Kodreferenspanelen

Dokumentet instruerar även programmet att visa specificerade metoder, händelser och egenskaper i användargränssnittet. Det är i programmets kodreferenspanel (Actions panel) denna listning sker.

(9) (4)

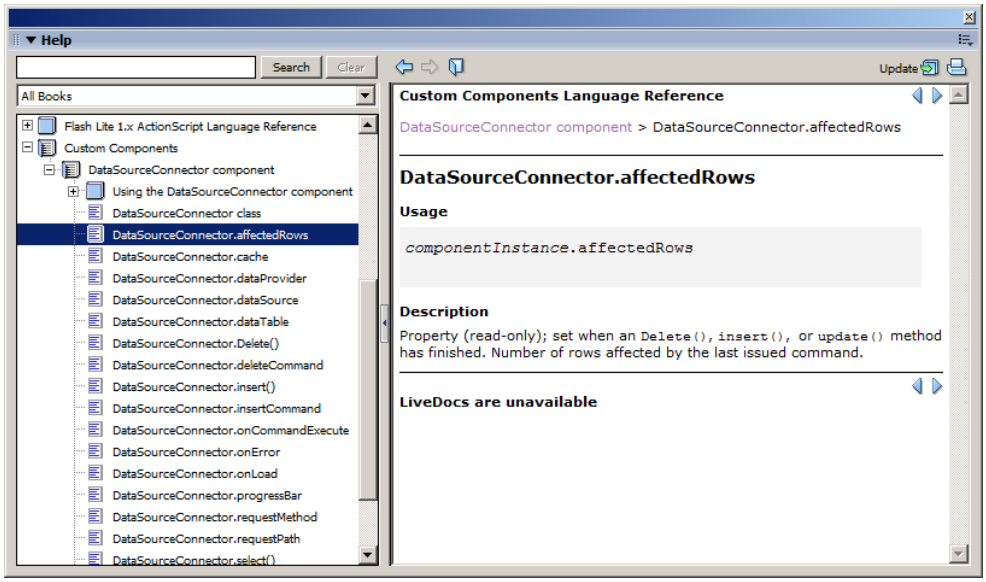
Önskvärt är att varje enskild post i kodreferenspanelen är kopplat till motsvarande avsnitt i komponentens dokumentation. Detta visade sig mycket svårt att åstadkomma. Då information om kodreferenspanelens uppbyggnad inte fanns att hitta, var det tvunget att gå igenom programmets konfigurationsfiler för att där söka ett samband mellan programmets paneler.

Ett samband hittades mellan attributet *helpid* och en fil vid namn *help.map*. Dessvärre kunde detta samband inte klarläggas fullt ut.

Programdokumentationen, hjälpfilerna, består huvudsakligen av ett hundratal HTML-dokument. Filerna har namngetts efter ett decimalt schema och samtliga filnamn utgörs av åtta siffror. Filen help.map utgör ett kopplingsschema där samtliga hjälpfiler, för gällande dokumentsektion, räknas upp i kronologisk ordning. Därtill har varje fil tilldelats ett unikt hexadecimalt identifikationsnummer med prefixet *x2*. Anledningen till detta prefix är okänt. Inte heller efter vilken formel de hexadecimala identifikationsnumren utdelas har kunnat klargöras. Möjligheten fanns att identitetsnumren var filnamnen uttryckta i en talbas om 16. Detta visade sig dock bara stämma för en bråkdel av listade filer.

Kopplingen skapades i stället genom att använda attributet *helpurl*. Attributet har sitt ursprung i en äldre version av kodreferenspanelen och fungerar även i Flash 8. (14)

3.3.4 Dokumentation

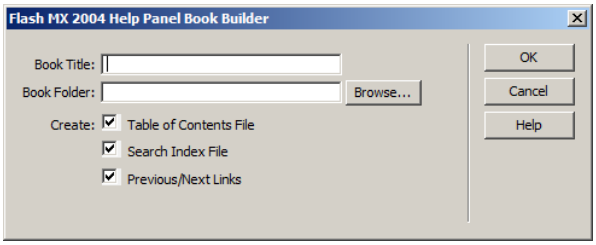


Figur 5. Hjälpgränssnittet

Flash är ett väl dokumenterat program, lika så de komponenter vilka som standard finns att bruka i utvecklingsmiljön. Utvecklade komponenter kan inledningsvis vara svåra att använda. Ett av de krav vilka finns specificerade i kravspecifikationen är enkelhet. Komponentdokumentationen avser möta detta kortkommande gällande användarvänlighet.

Instruktioner till hur hjälpfiler bör författas är i Flash obefintliga. I stället har Internet använts som informationskälla där mycket information varit utdaterat. Mycket av dokumentationsarbetet har således skett genom att manuell granska befintlig dokumentation och därigenom söka dess utformning och interna sammankoppling.

Vid dokumentation av utbyggnader till Flash finns en etablerad utformning och formatering att följa. Därtill finns utbyggnader och en dokumentmall för *Dreamweaver*, ett program för webbplatsproduktion, vilka underlättar dokumenteringen något.



Figur 6. Flash MX 2004 Help Panel Book Builder

Vid dokumentering har utbyggnaden *Flash MX 2004 Help Panel Book Builder* använts (se *Figur 6*). Utbyggnaden är en officiell sådan utvecklad och distribuerad av Adobe. Utbyggnaden automatgenererar en innehållsförteckning och sökindexeringsfil. Utbyggnaden skapar dessutom en linjär länkrelation dokumenten emellan, om sådan inte redan existerar.

(15)

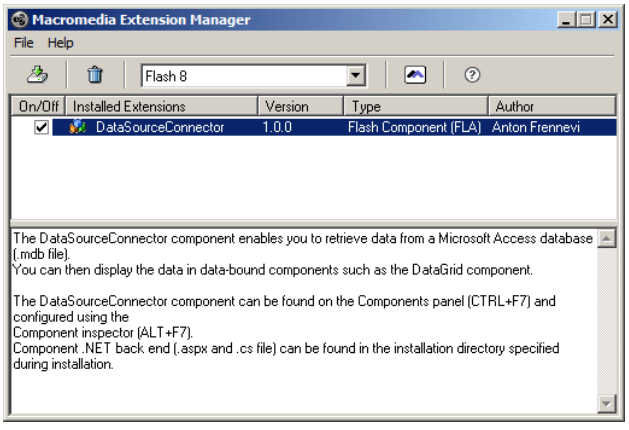
Formatmall använd är även den utvecklad och distribuerad av Adobe. Mallen är dessvärre utvecklad utefter MX 2004-standard och inte visuellt likvärdig den använd i dag. Formatmallen vidareutvecklades till en kopia av rådande standard. Utbyggnaden uppdaterades däremot inte vilket innebar att länkrelationer inte längre kunde upprättas automatiskt.

Dokumentationen utgörs av en serie HTML-dokument. Om dokumenten finns förekommande i en korrekt utformad innehållsförtäckning kan dokumentationen integreras i hjälpgränssnittet i Flash, se *Figur 5*. Sökindexeringsfilen möjliggör sökning i dokumentationen genom hjälpgränssnittet i Flash.

3.3.5 Paketering

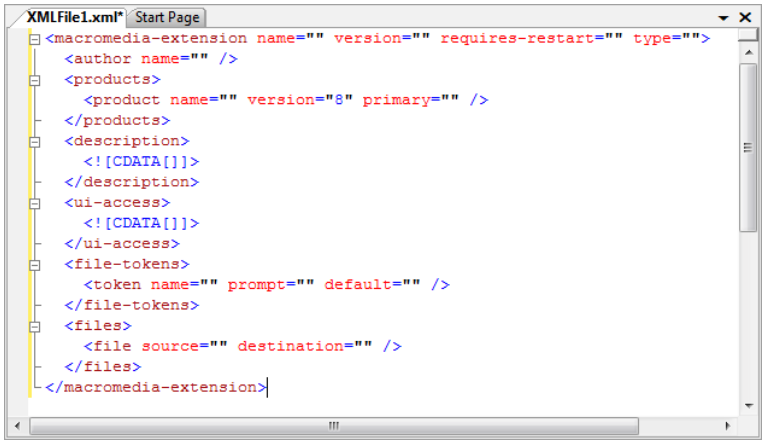
En komponent består av ett flertal olika filer som vid installation måste placeras i fördefinierade systemmappar. Detta för att Flash vid programstart automatiskt skall kunna läsa in tillagd komponent, tillhörande dokumentation med mera och möjliggöra komponentåtkomst genom programgränssnittet.

(9)



Figur 7. Programgränssnittet i Extension Manager

En komponent och dess tillhörande filer kan enkelt paketeras i en och samma fil med hjälp av programmet *Extension Manager*, se *Figur 7*. Extension Manager arbetar med två filformat; *MXI* och *MXP*. Programmet levereras tillsammans med Flash.



Figur 8. MXI

En MXI-fil är en fil författad i XML (se *Figur 8*) där komponentdata anges. I filen deklareras bland annat för vilket program utbyggnaden, i detta fall komponenten, är ämnad samt vilken programversion som krävs, hur utbyggnaden bör användas med mera. Vidare deklareras även relativa och symboliska sökvägar till de filer vilka skall paketeras och, vid installation, paketeras upp.

En symbolisk sökväg kallas i Extension Manager för en *token* och används för att adressera systemmappar relevanta för det program för vilket en utbyggnad är ämnad. Då gällande katalogstruktur kan variera mellan system kan tokens användas för att försäkra att paketerade filer packas upp i avsedda kataloger.

En MXP-fil är en utbyggnad i paketerad form. MXP-filen bygger på de instruktioner vilka angetts i tillhörande MXI-fil och innehåller all relevant data.

Av uppenbara skäl tillämpas ickeförstörande komprimering till filer paketerade i MXP-formatet. Kod komprimerad efter en förstörande algoritm riskerar att sluta fungera.

(16)

3.3.6 Symbolöverföring

Från och med version MX 2004 använder Flash *UTF-8*. (17) Det är därför viktigt för säker symbolöverföring att servern, vid anrop, returnerar data kodad efter samma teckentabell.

Det finns tecken som till följd av förhållanden utom kontroll inte kan överföras direkt. För att datatransmittering skall fungera tillförlitligt måste därför en lösning till denna begränsning utvecklas och implementeras.

Datatransmittering till och från Flash-applikationer sker med hjälp av par om variabelnamn och tillhörande data; data formaterad som en kontinuerlig teckensträng. För att dela upp denna teckensträng i par om variabelnamn och data används reserverade tecken. En teckensträng kan se ut som följer:

```
&variableName=variableData
```

Et-tecknet anger att följande tecken utgör ett variabelnamn. Likhetstecknet anger att vad som följer är data tillhörande före kommande variabel. Därtill finns flera andra reserverade tecken. (9)

För att möjliggöra överföring av reserverade tecken har jag valt att använda ett mycket enkelt, men fullt fungerande tillvägagångssätt. Reserverade tecken skrivs om, vid överföring, som ett procenttecken och den hexadecimala motsvarigheten till det reserverade tecknet i fråga. När teckensträngen tas emot är processen den omvända; en subrutin bearbetar teckensträngen och översätter hexadecimal data tillbaka till tecken. (18) Processen kallas procentkodning. Ett exempel på procentkodning följer nedan:

```
escape ("ActionScript - it's Good fun!");
unescape ("ActionScript%20%2D%20it%27s%20Good%20fun%21");
```

För procentkodning i Flash används metoderna *escape* och *unescape* vilka är definierade i ActionScript 1.0. (9)

För att försäkra säker teckenöverföring har samtliga komponenter, vilka använder par om variabelnamn och data, testats grundligt. En rad dataöverföringar har genomförts där samtliga tecken, angivna i bilaga 1, förekommit i skickad och mottagen data.

4 Resultat

Resultatet av detta examensarbete är tre kompletta komponenter för användning i utvecklingsmiljön Adobe Flash. Komponenterna bygger samtliga på ActionScript 2.0 och två av dem även på Microsoft .NET.

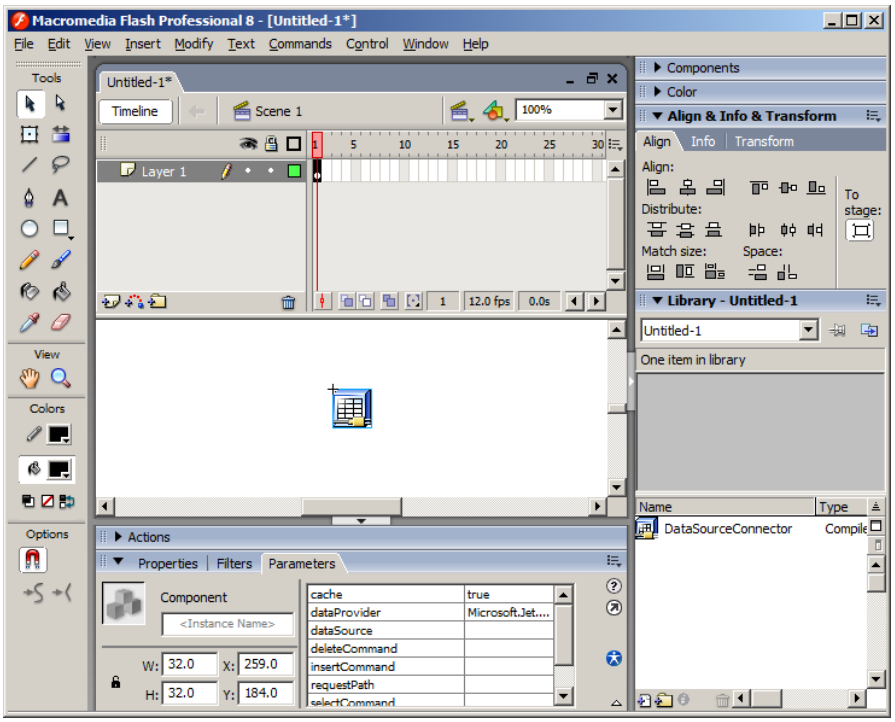
Resultatet finns att hitta i bifogad komponentkod samt på medföljande CD-skiva. Se bilagorna 2-7.

Samtliga utvecklade komponenter ärver ifrån UIComponent. Jag har genomgående försökt följa de riktlinjer och rekommendationer vilka finns att hitta i dokumentation rörande utveckling av komponenter. Ett mindre antal undantag har gjorts, vilka redogörs för i detta avsnitt.

Detta rapportavsnitt utgörs av redogörelser för de tre komponenter vilka tagits fram. På medföljande CD-skiva (bilaga 7) finns samtliga komponenter i paketerad form, det vill säga som MXP-filer. För att installera; dubbelklicka på någon av de tre MXP-filerna och följ anvisningarna som visas på skärmen.

4.1 Komponentbeskrivning

4.1.1 DataSourceConnector



Figur 9. DataSourceConnector-komponenten i Flash

Denna komponent utvecklades för att underlätta arbetet vid utveckling av datadrivna applikationer i Flash; se bilagorna 2 och 3 samt *Figur 9*. Som nämnts tidigare saknar Flash naturligt databasstöd. Komponenten utgör en beprövad lösning till denna begränsning. Komponenten syftar till att möjliggöra enkel interaktion mellan en Microsoft Access-databas och Flash.

Komponentgrunden utgörs av *LoadVars*; en klass tillgänglig i ActionScript 1.0 och Flash Player 6. (9) Klassen är ett alternativ till funktionen *loadVariables* och kan användas för att verifiera och övervaka datanerladdning. Klassen tillåter variabler att skickas till och/eller laddas ner från specificerad plats genom anrop till funktionerna *send*, *load* eller *sendAndLoad*. Denna komponent bygger på den senare.

sendAndLoad är en sammanfogning av funktionerna send och load. Funktionssyntaxen ser ut som följer: (9)

```
sendAndLoad(url:String, target:Object, [method:String])
```

Komponenten fungerar på så sätt att specificerad data skickas till applikationens mellanhand. Mellanhanden använder dessa data för att utföra specificerad databasaktion. Aktionsresultatet returneras sedan till applikationen vilken i sin tur bearbetar data och agerar därefter.

Vid användning av LoadVars samt funktionen sendAndLoad skickas och mottas datapar, det vill säga data formaterad som en kontinuerlig sträng. Hur variabelnamn och tillhörande data paras samman har redogjorts för tidigare.

Några uppenbara nackdelar med detta tillvägagångssätt är att all data skickas och tas emot som strängar. Ursprunglig typ av data spelar således ingen roll då all data vid transmittering konverteras till dess strängrepresentation. Transmitterade strängar kan därtill växa sig stora. Äldre versioner av Flash Player har visat sig ha svårigheter att hantera längre strängar.

Vid användning av LoadVars och funktionen sendAndLoad skickas och mottas data normalt i form av par; variabelnamn och tillhörande data. Att skapa par av data för varje enskilt fält i en databastabell vid selektion innebär en stor förlust i prestanda. Normal transmission är alltså inte att föredra.

Table1

Kolumnnamn 0	Kolumnnamn 1	Kolumnnamn 2	Kolumnnamn 3
Kolumn 0 Rad 0	Kolumn 1 Rad 0	Kolumn 2 Rad 0	Kolumn 3 Rad 0
Kolumn 0 Rad 1	Kolumn 1 Rad 1	Kolumn 2 Rad 1	Kolumn 3 Rad 1
Kolumn 0 Rad 2	Kolumn 1 Rad 2	Kolumn 2 Rad 2	Kolumn 3 Rad 2
Kolumn 0 Rad 3	Kolumn 1 Rad 3	Kolumn 2 Rad 3	Kolumn 3 Rad 3

Figur 10. Exempeltabell

Som exempel, betrakta *Figur 10*. Tabellen består av fyra kolumner och fyra rader (inledande rubrikrad bortses ifrån). Varje fält innehåller en sträng om 14 tecken. Tabellen innehåller totalt 224 tecken vilka vid selektion skall skickas från server till klienten. Önskas en fullständig kopia av tabellen blir den totala teckenmängden 272 tecken.

Vid standardtillämpning sänds data i formation om par. För varje fältsträng krävs en unik variabel föranledd av ett avskiljningstecken. För att kunna återskapa tabellen krävs att varje fältsträng märks upp på sådant sätt att de efter sändning kan placeras på rätt plats i tabellkopian. Detta är möjligt om varje variabelnamn innehåller numeriska pekare för kolumn- och radnummer. För exemplets skull tillåts fältkoordinaterna fungera variabelnamn. Uppräkning sker från tabellens översta vänstra hörn.

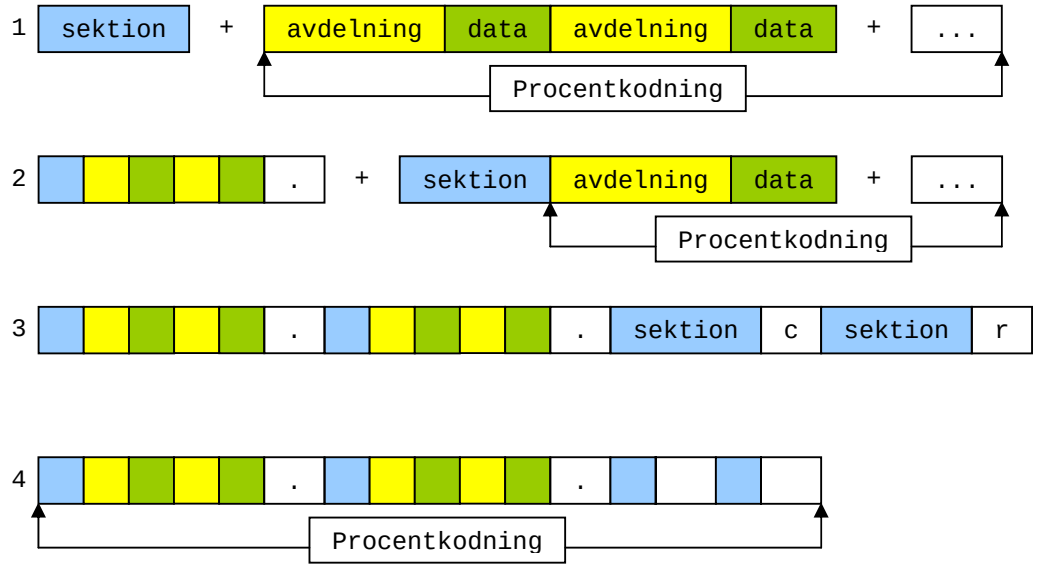
20 variabler har nu skapats. De fyra första variablerna innehåller kolumnnamn och de återstående 16 variablerna fältsträngar. Antalet tecken är nu 312 stycken. Därefter tillkommer 20 separationstecken vilket ger en summa på 332 tecken. Vid eventuell procentkodning ger varje förekommande mellanslag i exemplet en ökning om två tecken, men för enkelhetens skull bortses detta ifrån.

Denna komponent använder en annan typ än den av standard för att sända data. Exemplet tidigare skulle, bortsett ifrån kodning, ge en total mängd om i stället 277 tecken, det vill säga omkring 17 % färre tecken än med ovanstående metod. En större tabell ger högre effektivitet i jämförelse med ovan.

För förbättrad prestanda använder denna komponent teckenseparerad variabeldata. Data att returnera paras inte samman utan delas upp i mindre enheter genom användning av reserverade tecken.

Två specialtecken reserveras; ett tecken för sektionsindelning och ett för avdelningsindelning. Genom lagervis procentkodning hindras eventuella specialtecken, vilka kan förekomma bland det data som skall returneras, från att tolkas som reserverade tecken.

Serverprocessen beskrivs i detalj i figuren som följer.

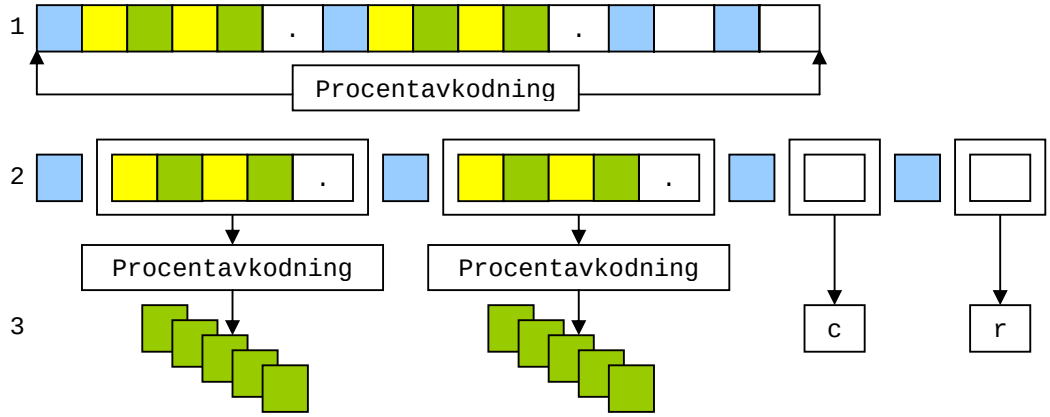


Figur 11. Serverprocessen vid komponentkommunikation

Data hämtas genom selektion. En kontinuerlig sträng, uppdelad i fyra sektioner föranledda av ett reserverat sektionstecken, skapas där data skiljs från varandra genom avdelningsindelning. Antalet kolumner och rader i aktuell tabell bifogas till strängens bakdel i form av två slutgiltiga strängsektioner.

Data i de två främre sektionerna skrivs om enskilt genom procentkodning. När strängen färdigställts kodas den slutligen som en helhet med procentkodning och sänds därefter till klienten.

Klientprocessen beskrivs i detalj i figuren som följer.



Figur 12. Klientprocessen vid komponentkommunikation

Klienten tar emot och avkodar försändelsen. Strängen delas upp i fyra sektioner. De två bakre sektionerna omvandlas till heltal och de två främre delas upp i avdelningar. Varje avdelning avkodas och tabellen kan därefter återskapas exakt.

4.1.1.1 Mellanhand

Den första prototypen för denna komponent programmerades i Visual Basic. Vid ett senare skede kom jag dock att översätta min prototyp till C# då jag fann språkets syntax mer lik den av ActionScript.

Därtill programmerades en prototyp för kommunikation genom XML-data. Fördelen med XML-data gentemot datapar är att XML grupperar data med dess innebörd. En mellanhand, utformad för hantering av XML, kan även användas för att returnera data till klienter av okänd utformning. (19)

XML-prototypen kom dock inte att vidareutvecklas då fördelarna med XML inte vägde tyngre än dess nackdelar. XML visade sig svårare att implementera och dess fördelar saknade betydelse. Då information utöver faktiska data transmitteras innebär XML i själva verket en försämring av prestanda (vid överföring av data).

Mellanhanden bygger på *Object Linking and Embedding Database*. Serverklassen gör skillnad på två typer av frågor där selektion utgör den första gruppen och infogning, uppdatering samt radering utgör den andra. En databaskoppling upprättas baserad på värden definierade i komponentinspektorn. Därefter körs angiven fråga och aktionsdata returneras. Om angivet kommando är av första gruppen returneras tabellstruktur och tabelldata. Om angivet kommando är av andra gruppen returneras endast antalet påverkade rader.

En enkel funktion för felloggning har programmerats i mellanhanden. Skulle ett fel uppstå noteras felet i ett enkelt textdokument vilket senare kan granskas.

4.1.1.2 Prestanda

Vid användning skickas data till en mellanhand. Innan data kan returneras genomförs en databasaktion. Därefter omstruktureras resultatdata för returnering. Flera separata steg vilka har en negativ inverkan på komponentens prestanda. Komponentens utformning presenterar flera olika potentiella flaskhalsar; klientdatorn, uppkopplingshastighet, rådande nät- och serverbelastning för att nämna några möjliga.

Genom att använda teckenseparerad data och inte XML- eller parbaserad sparas prestanda vid användning. Därtill lagras efterfrågad data temporärt på servern varje gång en urvalssats körs. En selektionssats returnerar samma data varje gång satsen körs, förutsatt att källan inte ändras. Att temporärt lagra data innebär att data inte behöver hämtas var gång selektionskommandot efterfrågas.

4.1.1.3 Begränsningar

Komponenten stöder inte tabeller med uppslag. (20)

4.1.1.4 Säkerhet

Data som skickas mellan server och klient skickas i klartext. Detta innebär att överförd data inte kan betraktas som säker. Information som önskas vara säker bör inte hanteras av denna komponent i dess nuvarande form. Det finns dock teoretiska lösningar till problemet.

I dagsläget finns ett flertal krypteringsklasser för ActionScript. Det finns klasser för *MD5* (21), *SHA1* (22) och *RSA* för att nämna några. Data som ämnas skickas till servern kan krypteras hos klienten och dekrypteras hos servern. Data som sedan skall returneras från servern kan i sin tur krypteras för att sedan dekrypteras hos klienten. Avsaknaden av säkerhet kan alltså åtgärdas.

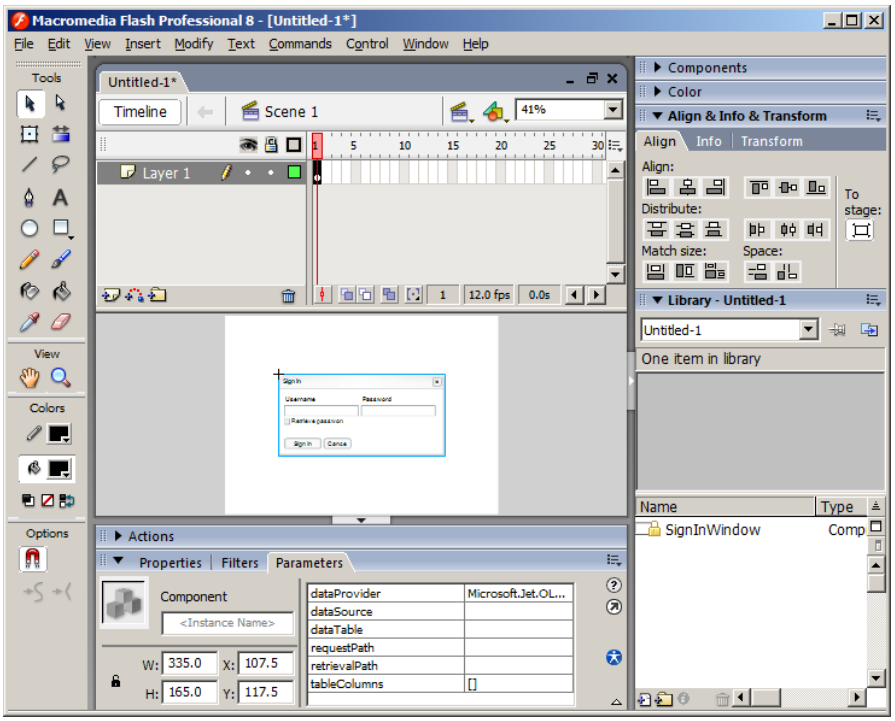
4.1.1.5 Utvecklingsmöjligheter

Som nämnts tidigare så stöder denna komponent inte databaser med uppslag. Uppslag är en viktig funktion i Microsoft Access och därför ett område efter vilket komponenten bör vidareutvecklas. Idag vet jag inte säkert varför uppslag utgör sig vara problematiskt för komponenten att hantera. Vidare tester och praktiska tillämpningar av komponenten kommer säkerligen att röja dess orsak. Dessvärre finns inte tid för detta inom ramarna för detta examensarbete.

Säkerhet är även det ett problemområde som tagits upp tidigare. En lösning till detta finns. Även denna lösning är dock för tidskrävande att implementera i dagsläget.

Möjligt vore att utveckla ytterligare, alternativa, mellanhänder. Det är fullt möjligt att skriva om nuvarande serverklass i exempelvis PHP. Därtill kan komponentens användningsområde utökas till att även omfatta MyISAM-databaser (MySQL). Då fordras dock större förändringar till komponentklassen på serversidan.

4.1.2 SignInWindow



Figur 13. SignInWindow-komponenten i Flash

Detta är en komponent för användarinloggning; se bilagorna 5 och 6 samt *Figur 13*. Det finns ett flertal scenarion där tillgänglighetsrestriktioner är en nödvändighet. Detta är en komponent för att enkelt kunna upprätta sådana restriktioner i Flash-applikationer.

Komponenten består enkelt av ett fönster i vilket användare kan ange sitt användarnamn och lösenord för inloggning. Komponenten kommunicerar med en bakomliggande mellanhand för att granska angivna uppgifter.

Komponenten bygger på i stort samma tillvägagångssätt som för tidigare nämnd komponent; data skickas till servern som datapar, bearbetas och returneras därefter som datapar. Processen är enkel.

4.1.2.1 Mellanhand

Mellanhanden för denna komponent bygger i stora drag på den för tidigare nämnd komponent.

4.1.2.2 Begränsningar

Komponenten bygger på ett antagande om att restriktioner är av binär natur. Som användare av applikationen i fråga tilldelas full eller begränsad åtkomst.

Därtill har ytterligare ett antagande gjorts rörande vilka uppgifter som krävs för att initiera ett inloggningsförsök. För åtkomst krävs ett giltigt användarnamn och ett lösenord identiskt med det kopplat till angivet användarnamn. Detta innebär att matchning sker mot två separata fält i den tabell vilken innehåller godkända användaruppgifter. Referenser till dessa kolumner kan av användaren anges för ökad konfigurationsmöjlighet.

I stort är den inloggningsfråga vilken komponenten använder redigeringslåst. Detta innebär att komponenten inte kan skräddarsys mer ingående.

4.1.2.3 Säkerhet

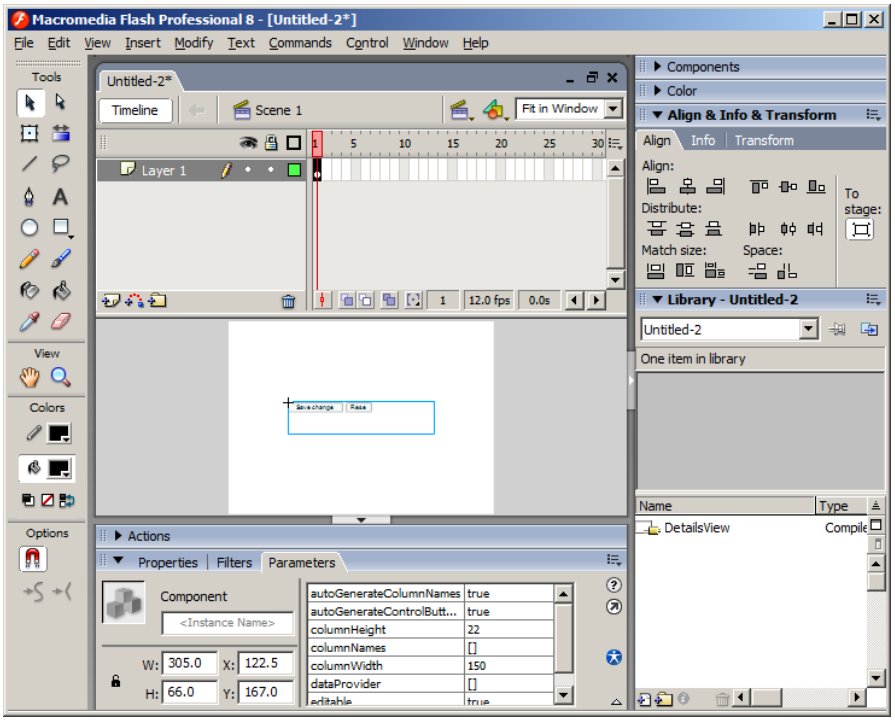
Likt tidigare nämnd komponent skickas data mellan klient och server i klartext, något som inte är att föredra om säkerhet är av vikt. Då denna komponent är tänkt att hantera känsliga uppgifter så som användarnamn och tillhörande lösenord kan komponenten inte användas praktiskt i dess nuvarande form. För full säkerhet krävs bland annat att data krypteras hos servern så väl som hos klienten. Ett säkert protokoll vore att föredra.

Så som kommunikationen mellan klient och server fungerar i nuläget är komponenten sårbar för så kallad *frågeinjektion*. Den fråga vilken körs mot databasen kan lätt manipuleras genom att använda reserverade tecken i de datafält vilka används för inloggning.

Ett sätt att avhjälpa problemet vore att använda lagrade procedurer. Här har användarvänlighet vägs mot säkerhet. Då komponenten i dess nuvarande utformning är allt annat än säker används för närvarande inte lagrade procedurer.

Ytterligare ett tillvägagångssätt vore att införa restriktioner till vilka tecken som kan anges av användaren vid inloggning. Detta är dock inte gångbart då mängden möjliga tecken ett lösenord kan bestå av i sig bidrar till högre säkerhet.

4.1.3 DetailsView



Figur 14. DetailsView-komponenten i Flash

Detta är en visuell komponent vilken kan kopplas till datakällor, se bilaga 4 samt *Figur 14*. Komponenten begränsar den visuella representationen av datakällan genom att bara presentera källans första rad av data. Oavsett datakällans storlek kan således denna komponent användas för att ge en grafisk representation av kopplad data.

Till komponenten har redigeringsmöjligheter lagts. Data som presenteras genom användandet av denna komponent kan direkt eller genom ActionScript-kod enkelt redigeras. Ändringar gjorda reflekteras i en redigeringsfråga som automatiskt sätts samman av komponenten. Frågan kan enkelt användas för att uppdatera en extern datakälla med gjorda ändringar.

Komponenten saknar helt mellanhand då dess funktion saknar behov av sådan. Objektet kan kopplas till mellanhandsberoende komponenter. Detta är dock en möjlighet, inte ett krav, för att komponenten skall fungera.

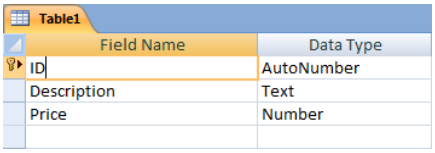
I programdokumentationen för Flash finns en rekommendation rörande den obligatoriska metoden *createChildren*. Metoden är ett krav för komponenter vilka utökar UIComponent och syftar till renderingen av till komponenten barnobjekt. Att skapa barnobjekt utanför denna metod är inte i enlighet med givna riktlinjer. I stället skall flaggor sättas och metoden createChildren anropas genom att ogiltigförklara komponenten, detta genom anrop till metoden *invalidate*. (9) Jag valde att bortse ifrån detta och skapa barnobjekt utanför denna metod.

4.1.3.1 Begränsningar

Vid redigering av data återspeglas gjorda ändringar i en av komponenten automatgenererad redigeringsfråga. All data visualiserad genom denna komponent omvandlas till strängform. Detta oavsett ursprunglig datatyp. Flash är mycket flexibelt i detta avseende; tal i strängform kan användas vid matematiska beräkningar.

Problem uppstår då redigeringsfrågan skall konstrueras. Då hänsyn inte kan tas till ursprunglig datatyp (då denna information gått förlorad), sammanfogas en fråga om strängvärden.

Problemet kan illustreras genom ett enkelt exempel. Genom en komponentinstans selekteras all förekommande data i en tabell av strukturen i *Figur 15*.



Figur 15. Tabellstruktur

Vid överföring av selekterad data, från mellanhanden till SWF-filen i fråga, går all data rörande varje fälts datatyp förlorad. Att tabellens första fält är av datatypen *räknare* är data som vid överföring faller bort. Selekterad data presenteras sedan i en instans av aktuell komponent. Om information rörande ursprunglig datatyp bibehållits vid överföring går den i stället förlorad vid presentation.

När redigeringsfrågan genereras utformas den för redigering av fält av strängbaserad datatyp (i *Figur 15* datatypen *text*). När frågan skickas till mellanhanden för att utföras kommer instruktionen att misslyckas. Strängdata kan inte infogas i fält av talbaserad datatyp. Således kan redigeringsfrågan inte appliceras på datatyperna *AutoNumber* och *Number*.

En lösning till detta problem vore att vidareutveckla komponenten så att hänsyn tas till ursprunglig datatyp av de data som skall presenteras. För att exemplet som getts skall fungera fullt ut måste även mellanhanden vidareutvecklas så att även information rörande datatyp överförs.

Ytterligare en brist i komponentens utformning är utförandet av metoden *updateCommand*. Metoden genererar en redigeringsfråga och ser då till alla representerade fält – även de vilka inte redigerats. Detta innebär att mer data än absolut nödvändigt inkluderas. Redigeringsfrågan är inte heller möjlig att redigera i den bemärkelsen att fält att inkludera inte kan ändras.

5 Slutsats och diskussion

Arbetet har resulterat i tre komponenter med grundläggande funktioner. Tiden räckte inte till och därför lämpar sig inte komponenterna för praktisk användning. Komponenterna är till grunden fungerande, men saknar delar vilka försäkrar fullgod funktionalitet. Således krävs ytterligare arbete innan komponenterna kan användas fullt ut inom den egna företagsverksamheten.

Det finns flera olika metoder för att möjliggöra kommunikation mellan en SWF-fil och en databas. Tillvägagångssättet vilket används inom detta examensarbete är tekniskt okomplicerat. Dessvärre är metoden klumpig och medför flera begränsningar; stöd för databaser med uppslag saknas och typ av data går förlorad vid överföring.

Flash Remoting är en teknik vilken ger en näst intill sömlös koppling mellan klient och server. Flash Remoting använder binär kommunikation och möjliggör anrop till externa funktioner som om deklarerade på lokal nivå. (2) Detta är en mycket kraftfull men krävande teknik och därför olämplig för mindre applikationer.

En av Macromedia utvecklad tillämpning av Flash Remoting är *Flash Remoting MX* som kan användas tillsammans med ColdFusion, *JRun*, .NET och *Java*. (2) Då Flash Remoting MX utgör ett licensierat programtillägg valde jag att inte utveckla tidigare nämnda komponenter utefter den funktionalitet tillägget innebär. Samtliga komponenter har istället byggts på basfunktioner fritt tillgängliga i Flash.

Under arbetets gång och vid närmare studier av möjligheterna i Flash fann jag flera andra sätt på vilka komponenterna kunnat utvecklas. Jag fann en utökad komponent i programmet av vilka mina komponenter med fördel kunnat ärva. Inte bara hade detta inneburit skillnader till komponentklasserna utan även till mellanhänderna. I nuläget används webbformulär för att styra mellanhandskommunikationen. Komponenten skulle inneburit att webbtjänster skulle kunnat användas. Nuvarande mellanhänder kan ses som en imitation till webbtjänsternas utformning.

Ser man till hur programmerade mellanhänder fungerar är det uppenbart att de med fördel skulle kunnat utformas i form av webbtjänster. Detta var insikt jag saknade vid projektets början.

Flash har i senare versioner blivit mer och mer orienterat kring användningen av XML. Jag har medvetet valt att undvika XML av flera olika skäl vilka redogjorts för tidigare. XML skulle dock inneburit en mer skal- och portbar utformning.

När detta examensarbete påbörjades hade jag ingen tidigare erfarenhet av Flash-komponenter. Jag har sedan flera år tillbaka använt mig av Flash vid webbutveckling. Under denna tid har jag aldrig sett till fördelarna med komponenter. Under arbetets gång har jag blivit medveten om hur mycket som finns att vinna genom objektorienterad programmering.

Komponenter i Flash innebär en betydande arbetslättnad vid utveckling av Flash-media. Objekt kan återanvändas och förändringar till en överordnad klass tillåts flöda neråt i klasshierarkin. Fördelarna med Flash-komponenter är alltså många. Generellt definierade komponenter kan konfigureras om och användas i flera skilda projekt.

Genom att kombinera möjligheterna Flash-media ger och de fördelar vilka användandet av databaser medför, kan applikationer med ett stort funktionsomfång skapas. Ett spel kan lagra poäng och statistik, ett forum kan lagra trådar och inlägg, en webbshop kan lagra produkter. Genom användandet av databaser kan dynamik läggas till ett annars statiskt medieinnehåll. Därför föll det för mig naturligt att omvandla redogjord kommunikationsmetod till komponentform.

Som nämnts tidigare finns det flera områden med möjlighet till förbättring. Komponenterna är i nuläget begränsade och lämpar sig inte för användning. Arbetet skulle med fördel kunnat begränsas till endast en komponent.

Ser vi till rådande begränsningar och brister finns mycket att åtgärda:

- Komponenten för inloggning saknar kryptering och kan därför inte anses säker. Tillhörande mellanhand är bland annat sårbar för frågeinjektion. Därtill är mellanhanden utformad kring antaganden rörande strukturen av databasen innehållande uppgifter för inloggning.
- Komponenten för datapresentation har även den brister vilka skulle kunnat åtgärdas om mer tid funnits till hand. Som komponenten fungerar i nuläget görs ingen skillnad på indata i form av tal och strängar.
- Komponenten för databaskommunikation bortser ifrån datatyp vid selektion. En betydlig förbättring av komponenten vore att arbeta bort denna brist.

Slutligen vill jag säga att detta examensarbete varit mycket lärorikt. Jag känner att jag till stor del har lyckats uppnå mina mål, men än finns brister vilka bör åtgärdas innan utvecklade komponenter tas i bruk.

Det var ibland svårt att hitta den information jag behövde. Mycket arbete har därför skett enligt *trial and error*-metoden. Detta tillvägagångssätt har inneburit mycket merarbete. Samtidigt anser jag att jag i och med detta arbetssätt fått en djupare förståelse för hur komponenter fungerar och hur de skapas.

6 Referenser

1. **Adobe.** *Adobe Press Room.* [Online] [Citat: den 6 februari 2007.] http://www.adobe.com/pressrom/.
2. —. *Adobe.* [Online] [Citat: den 30 januari 2007.] http://www.adobe.com/.
3. —. *Adobe TechNote.* [Online] [Citat: den 30 januari 2007.] http://www.adobe.com/support/. 14482.
4. —. *Adobe Components Learning Guide.* [Online] [Citat: den 24 april 2007.] http://www.adobe.com/devnet/flash/components/components.html.
5. **Eriksson, Hans-Erik och Penker, Magnus.** *Objektorientering - handbok och lexikon.* Lund : Studentlitteratur, 2000. 91-44-49801-2.
6. **Adobe.** *Adobe XML News Aggregator.* [Online] [Citat: den 25 januari 2007.] http://weblogs.adobe.com/.
7. **Meyne, Hank och Davis, Scott.** *Developing Web Applications with ASP.NET and C#.* New York : John Wiley & Sons, Inc., 2002. 0-471-12090-1.
8. **Microsoft.** *MSDN.* [Online] Microsoft. [Citat: den 24 april 2007.] http://msdn.microsoft.com/.
9. **Adobe.** *Adobe Flash 8 LiveDocs.* [Online] [Citat: den 26 januari 2007.] http://livedocs.adobe.com/flash/8/.
10. **Sahlin, Doug.** *Flash ActionScript i ett nötskal.* Sundbyberg : Pagina Förlags AB, 2002. 91-7241-076-0.
11. **De Donatis, Antonio.** *AdvancED ActionScript Components: Mastering the Flash Component Architecture.* Berkeley : Friends of Ed, 2006. 1590595939.
12. **Elst, Peter, o.a.** *Object-Oriented ActionScript for Flash 8.* Berkeley : Friends of Ed, 2006. 1590596196.
13. **Adobe.** *Adobe Developer Center.* [Online] [Citat: den 6 maj 2007.] http://www.adobe.com/devnet/flash/articles/cybersage.html.
14. —. *Adobe Support.* [Online] [Citat: den 6 maj 2007.] http://www.adobe.com/support/documentation/en/flash/mx2004/extending_help/extending_help5.html.
15. —. *Adobe Support.* [Online] [Citat: den 6 maj 2007.] http://www.adobe.com/support/documentation/en/flash/mx2004/extending_help/extending_help6.html.
16. —. *Adobe Exchange.* [Online] [Citat: den 6 maj 2007.] http://download.macromedia.com/pub/exchange/mxi_file_format.pdf.
17. —. *Adobe TechNote.* [Online] [Citat: den 5 februari 2007.] http://www.adobe.com/support/. 1077744.
18. —. *Adobe TechNote.* [Online] [Citat: den 27 mars 2007.] http://www.adobe.com/support/. tn_14143.
19. **Östlund, Anna och Helen, Hermundstad.** *XML, DTD:er och XSL.* Sundbyberg : Docendo Sverige AB, 2001. 91-7882-562-8.
20. **Microsoft.** *Microsoft Office.* [Online] Microsoft. [Citat: den 7 juni 2007.] http://office.microsoft.com/sv-se/access/HA101637731053.aspx.
21. **Crugnola, Alessandro.** *Sephiroth.* [Online] den 7 oktober 2003. [Citat: den 7 juni 2007.] http://www.sephiroth.it/file_detail.php?id=69.
22. **Eismann, Dirk.** *RichInternet.* [Online] den 12 oktober 2004. [Citat: den 7 juni 2007.] http://www.richinternet.de/blog/index.cfm?entry=8C8B659E-0858-D8EC-61DECA2DFF048623.

7 Sökord

8 Bilagor

Bilaga 1 Procentkodningstabell för teckenöverföring

Bilaga 2 Källkod för DataSourceConnector (AS)

Bilaga 3 Källkod för DataSourceConnector (C#)

Bilaga 4 Källkod för DetailsView (AS)

Bilaga 5 Källkod för SignInWindow (AS)

Bilaga 6 Källkod för SignInWindow (C#)

Bilaga 7 CD-skiva innehållande källkod, paketerade komponenter och exempelfiler

Symbol	Kod	Symbol	Kod	Symbol	Kod	Symbol	Kod
©		G	%47	u	%75	Ó	%D3
®		H	%48	v	%76	Ô	%D4
™		I	%49	w	%77	Õ	%D5
backsteg	%08	J	%4A	x	%78	Ö	%D6
tabb	%09	K	%4B	y	%79	Ø	%D8
	%0A	L	%4C	z	%7A	Ù	%D9
	%0D	M	%4D	{	%7B	Ú	%DA
mellanslag	%20	N	%4E	|	%7C	Û	%DB
!	%21	O	%4F	}	%7D	Ü	%DC
”	%22	P	%50	~	%7E	Ý	%DD
#	%23	Q	%51	¢	%A2	Þ	%DE
$	%24	R	%52	£	%A3	ß	%DF
%	%25	S	%53	¥	%A5	à	%E0
&	%26	T	%54	¦	%A6	á	%E1
'	%27	U	%55	§	%A7	â	%E2
(	%28	V	%56	«	%AB	ã	%E3
)	%29	W	%57	¬	%AC	ä	%E4
*	%2A	X	%58	¯	%AD	å	%E5
+	%2B	Y	%59	º	%B0	æ	%E6
,	%2C	Z	%5A	±	%B1	ç	%E7
-	%2D	[	%5B	ª	%B2	è	%E8
.	%2E	\	%5C	,	%B4	é	%E9
/	%2F	]	%5D	µ	%B5	ê	%EA
0	%30	^	%5E	»	%BB	ë	%EB
1	%31	_	%5F	¼	%BC	ì	%EC
2	%32	`	%60	½	%BD	í	%ED
3	%33	a	%61	¿	%BF	î	%EE
4	%34	b	%62	À	%C0	ï	%EF
5	%35	c	%63	Á	%C1	ð	%F0
6	%36	d	%64	Â	%C2	ñ	%F1
7	%37	e	%65	Ã	%C3	ò	%F2
8	%38	f	%66	Ä	%C4	ó	%F3
9	%39	g	%67	Å	%C5	ô	%F4
:	%3A	h	%68	Æ	%C6	õ	%F5
;	%3B	i	%69	Ç	%C7	ö	%F6
<	%3C	j	%6A	È	%C8	÷	%F7
=	%3D	k	%6B	É	%C9	ø	%F8
>	%3E	l	%6C	Ê	%CA	ù	%F9
?	%3F	m	%6D	Ë	%CB	ú	%FA
@	%40	n	%6E	Ì	%CC	û	%FB
A	%41	o	%6F	Í	%CD	ü	%FC
B	%42	p	%70	Î	%CE	ý	%FD
C	%43	q	%71	Ï	%CF	þ	%FE
D	%44	r	%72	Đ	%D0	ÿ	%FF

E	%45	s	%73	Ñ	%D1		
F	%46	t	%74	Ò	%D2		

DataSourceConnector.as

```
import mx.core.UIComponent;
[IconFile("icon-19x19.png")]
[Event("onCommandExecute")]
[Event("onError")]
[Event("onLoad")]
[InspectableList("cache", "dataProvider", "dataSource", "deleteCommand",
"insertCommand", "requestPath", "selectCommand", "updateCommand")]
class com.remote.DataSourceConnector extends UIComponent
{
      static var symbolName:String = "DataSourceConnector";
      static var symbolOwner:Object = Object (DataSourceConnector);
      var className:String = "DataSourceConnector";
      private var __affectedRows:Number = 0;
      public function get affectedRows ():Number
      {
            return __affectedRows;
      }
      private var __cache:Boolean;
      public function get cache ():Boolean
      {
            return __cache;
      }
      [Inspectable(defaultValue=true,type=Boolean)]
       public function set cache (value:Boolean):Void
      {
            __cache = value;
      }
      private var __dataProvider:String;
      public function get dataProvider ():String
      {
            return __dataProvider;
      }
      [Inspectable(defaultValue="Microsoft.Jet.OLEDB.4.0",type=String)]
       public function set dataProvider (value:String):Void
      {
            __dataProvider = value;
      }
      private var __dataSource:String;
      public function get dataSource ():String
      {
            return __dataSource;
      }
      [Inspectable(type=String)]
       public function set dataSource (value:String):Void
      {
            __dataSource = value;
      }
      private var __dataTable:Array = new Array ();
```

```
	public function get dataTable ():Array
	{
		return __dataTable;
	}
	private var __deleteCommand:String;
	public function get deleteCommand ():String
	{
		return __deleteCommand;
	}
	[Inspectable(type=String)]
	 public function set deleteCommand (value:String):Void
	{
		__deleteCommand = value;
	}
	private var __insertCommand:String;
	public function get insertCommand ():String
	{
		return __insertCommand;
	}
	[Inspectable(type=String)]
	 public function set insertCommand (value:String):Void
	{
		__insertCommand = value;
	}
	private var __progressBar:Object;
	public function get progressBar ():Object
	{
		return __progressBar;
	}
	public function set progressBar (value:Object):Void
	{
		__progressBar = value;
	}
	private var __requestMethod:String = "POST";
	public function get requestMethod ():String
	{
		return __requestMethod;
	}
	private var __requestPath:String;
	public function get requestPath ():String
	{
		return __requestPath;
	}
	[Inspectable(type=String)]
	 public function set requestPath (value:String):Void
	{
		__requestPath = value;
	}
```

```
	private var __selectCommand:String;
	public function get selectCommand ():String
	{
		return __selectCommand;
	}
	[Inspectable(type=String)]
	 public function set selectCommand (value:String):Void
	{
		__selectCommand = value;
	}
	private var __updateCommand:String;
	public function get updateCommand ():String
	{
		return __updateCommand;
	}
	[Inspectable(type=String)]
	 public function set updateCommand (value:String):Void
	{
		__updateCommand = value;
	}
	private var progressInterval:Number;
	function DataSourceConnector ()
	{
	}
	function init ():Void
	{
		super.init ();
	}
	public function createChildren ():Void
	{
		_visible = false;
		size ();
	}
	function draw ():Void
	{
		super.draw ();
	}
	function size ():Void
	{
		super.size ();
		invalidate ();
	}
	private function item (colArr:Array, fldArr:Array):Void
	{
		/**
		* Function for table data insertion.
		*/
		for (var i:Number = 0; i < fldArr.length; i++)
```

```
            {
                  this[colArr[i]] = fldArr[i];
            }
      }
      public function select ():Void
      {
            /**
            * Function called on select command execution. Sends required
            * variables to the server resident class and processes returned
            * data.
            */
            var component:DataSourceConnector = this;
            var send:LoadVars = new LoadVars ();
            var load:LoadVars = new LoadVars ();
            try
            {
                  /**
                  * Variables to be passed to the server resident class are
                  * set.
                  */
                  send.cache = __cache;
                  send.command = __selectCommand;
                  send.selectMethod = true;
                  send.dataProvider = __dataProvider;
                  send.dataSource = __dataSource;
                  /**
                  * Validates variable data. If invalid an error is thrown.
                  */
                  var data:Array = unescape (send.toString ()).split
(String.fromCharCode (38));
                  for (var i:Number = 0; i < data.length; i++)
                  {
                        data[i] = data[i].substr (data[i].indexOf
(String.fromCharCode (61)) + 1);
                        if (data[i].length == 0)
                        {
                              throw new Error ();
                        }
                  }
                  /**
                  * Progress bar update interval. Run while waiting for
                  * server response.
                  */
                  load.onLoadProgress = function ():Void
                  {
                        component.__progressBar.setProgress
(this.getBytesLoaded (), this.getBytesTotal ());
                        if (this.getBytesLoaded () == this.getBytesTotal ())
                        {
```

```
                              clearInterval (component.progressInterval);
                        }
                  };
                  progressInterval = setInterval (load, "onLoadProgress", 5);
                  /**
                  * Data table is cleared.
                  */
                  __dataTable.removeAll ();
                  /**
                  * Component core method is called. A command event is
                  * dispatched.
                  * Waiting for server response.
                  */
                  send.sendAndLoad (__requestPath +
"DataSourceConnector.aspx", load, __requestMethod);
                  var event:Object = {target:this, type:"onCommandExecute"};
                  dispatchEvent (event);
                  load.onLoad = function (success:Boolean):Void
                  {
                        if (success)
                        {
                              /**
                              * Server data was returned. Proceeds with data
                              * process.
                              */
                              this = unescape (this);
                              this = this.substr (0, this.indexOf
(String.fromCharCode (61)));
                              /**
                              * Returned data is split into an array of four
                              * separate fields.
                              */
                              var arr:Array = this.split (String.fromCharCode
(38));
                              arr = arr.splice (1, arr.length);
                              /**
                              * First field is split additionally into a
                              * field array.
                              */
                              var fldArr:Array = arr[0].split
(String.fromCharCode (35));
                              fldArr.reverse ();
                              fldArr.pop ();
                              /**
                              * Each field array field is unescaped and
                              * stored in its initial field position.
                              */
                              for (var i:Number = 0; i < fldArr.length; i++)
                              {
```

```
                                    fldArr[i] = unescape (fldArr[i]);
                              }
                              /**
                              * Second field is split additionally into a
                              * column array.
                              */
                              var colArr:Array = arr[1].split
(String.fromCharCode (35));
                              colArr.reverse () ();
                              colArr.pop ();
                              /**
                              * Each column array field is unescaped and
                              * stored in its initial field position.
                              */
                              for (var i:Number = 0; i < colArr.length; i++)
                              {
                                    colArr[i] = unescape (colArr[i]);
                              }
                              /**
                              * Data table column and row count is set
                              * according to third and forth data field.
                              */
                              var colCnt:Number = Number (arr[2]);
                              var rowCnt:Number = Number (arr[3]);
                              /**
                              * Data table is updated according to values in
                              * field and column array.
                              */
                              for (var i:Number = 0; i < rowCnt; i++)
                              {
                                    component.__dataTable.addItem (new
component.item (colArr, fldArr.slice (i * colCnt, (i * colCnt) + colCnt)));
                              }
                              component.__dataTable.reverse ();
                              /**
                              * Server data has been loaded and processed. A
                              * load event is dispatched.
                              */
                              var event:Object = {target:component,
type:"onLoad"};
                              component.dispatchEvent (event);
                        }
                        else
                        {
                              /**
                              * No server data was returned. An error event
                              * is dispatched.
                              */
                              throw new Error ();
```

```
                        }
                  };
            }
            catch (Error)
            {
                  /**
                  * An error occurred. An error event is dispatched.
                  */
                  var event:Object = {target:component, type:"onError"};
                  component.dispatchEvent (event);
            }
      }
      private function deleteInsertUpdate (command:String,
selectMethod:Boolean):Void
      {
            /**
            * Function called on delete, insert and update command execution.
            * Sends required variables
            * to the server resident class and processes returned data.
            */
            var component:DataSourceConnector = this;
            var send:LoadVars = new LoadVars ();
            var load:LoadVars = new LoadVars ();
            try
            {
                  /**
                  * Variables to be passed to the server resident class are
                  * set.
                  */
                  send.command = command;
                  send.selectMethod = selectMethod;
                  send.dataProvider = __dataProvider;
                  send.dataSource = __dataSource;
                  /**
                  * Validates variable data. If invalid an error is thrown.
                  */
                  var data:Array = unescape (send.toString ()).split
(String.fromCharCode (38));
                  for (var i:Number = 0; i < data.length; i++)
                  {
                        data[i] = data[i].substr (data[i].indexOf
(String.fromCharCode (61)) + 1);
                        if (data[i].length == 0)
                        {
                              throw new Error ();
                        }
                  }
                  /**
                  * Progress bar update interval. Run while waiting for
```

```
                  * server response.
                  */
                  load.onLoadProgress = function ():Void
                  {
                        component.__progressBar.setProgress
(this.getBytesLoaded (), this.getBytesTotal ());
                        if (this.getBytesLoaded () == this.getBytesTotal ())
                        {
                              clearInterval (component.progressInterval);
                        }
                  };
                  progressInterval = setInterval (load, "onLoadProgress", 5);
                  /**
                  * Component core method is called. A command event is
                  * dispatched.
                  * Waiting for server response.
                  */
                  send.sendAndLoad (__requestPath +
"DataSourceConnector.aspx", load, __requestMethod);
                  var event:Object = {target:this, type:"onCommandExecute"};
                  dispatchEvent (event);
                  load.onLoad = function (success:Boolean):Void
                  {
                        if (success)
                        {
                              /**
                              * Server data was returned. Proceeds with data
                              * process.
                              */
                              this = unescape (this);
                              this = this.substr (0, this.indexOf
(String.fromCharCode (61)));
                              /**
                              * Command was successful. Number of affected
                              * rows is returned and a
                              * load event is dispatched.
                              */
                              component.__affectedRows = Number (this);
                              var event:Object = {target:component,
type:"onLoad"};
                              component.dispatchEvent (event);
                        }
                        else
                        {
                              /**
                              * No server data was returned. An error event
                              * is dispatched.
                              */
                              throw new Error ();
```

```
                        }
                };
        }
        catch (Error)
        {
                /**
                * An error occurred. An error event is dispatched.
                */
                var event:Object = {target:component, type:"onError"};
                component.dispatchEvent (event);
        }
    }
    public function Delete ():Void
    {
        /**
        * Function call to run delete command.
        */
        deleteInsertUpdate (__deleteCommand, false);
    }
    public function insert ():Void
    {
        /**
        * Function call to run insert command.
        */
        deleteInsertUpdate (__insertCommand, false);
    }
    public function update ():Void
    {
        /**
        * Function call to run update command.
        */
        deleteInsertUpdate (__updateCommand, false);
    }
}
```

```
1  namespace Com.Remote
2  {
3      using System;
4      using System.IO;
5      using System.Data;
6      using System.Data.OleDb;
7      using System.Configuration;
8      using System.Collections;
9      using System.Web;
10     using System.Web.Security;
11     using System.Web.UI;
12     using System.Web.UI.WebControls;
13     using System.Web.UI.WebControls.WebParts;
14     using System.Web.UI.HtmlControls;
15
16     public partial class DataSourceConnector : System.Web.UI.Page
17     {
18         private decimal[] d = { 35, 38 };
19         private string _buffer;
20         protected void Page_Load(object sender, EventArgs e)
21         {
22             /**
23             * A connection is created and opened.
24             */
25             OleDbConnectionStringBuilder c_str = new OleDbConnectionStringBuilder();
26             c_str.Provider = Request.Form["dataProvider"];
27             c_str.DataSource = Server.MapPath(Request.Form["dataSource"]);
28
29             OleDbConnection c = new OleDbConnection(c_str.ConnectionString);
30
31             try
32             {
33                 c.Open();
34
35                 if (Convert.ToBoolean(Request.Form["selectMethod"]))
36                 {
37                     /**
38                     * A select command has been issued.
39                     */
40                     if (Convert.ToBoolean(Request.Form["cache"]))
41                     {
42                         /**
43                         * Cache is to be used. If issued command has been executed ↙
    previously, cached data
44                         * is returned. Otherwise the command is executed.
45                         */
46                         object _cache = Cache[Request.Form["command"]];
47                         if (_cache == null)
48                         {
49                             select(c);
50                         }
51                         else
52                         {
53                             /**
54                             * Return buffer data to client class as string ↙
    representation.
55                             */
56                             Response.Write(_cache);
57                         }
58                     }
59                     else
60                     {
61                         /*
62                         * Cache is not to be used. Execute issued select command.
63                         */
64                         select(c);
65                     }
66                 }
67                 else
68                 {
69                     /**
70                     * A delete, insert or update command have been issued. Execute ↙
    issued command.
71                     */
```

```
72                      deleteInsertUpdate(c);
73                  }
74              }
75              catch (Exception _e)
76              {
77                  /**
78                  * An exception was thrown. The exception is logged.
79                  */
80                  writeLog(_e);
81              }
82              finally
83              {
84                  /**
85                  * A connection is closed.
86                  */
87                  c.Close();
88              }
89          }
90          private void select(OleDbConnection c)
91          {
92              /**
93              * Function called on select command execution.
94              */
95              OleDbDataAdapter adp = new OleDbDataAdapter(Request.Form["command"], c);
96              DataTable dt;
97              ArrayList arr;
98              /**
99              * The data table is populated with query result data.
100             */
101             dt = new DataTable();
102             adp.Fill(dt);
103             /**
104             * Data is extracted from the data table and the inserted into an data array↙
        list.
105             */
106             arr = new ArrayList();
107             for (int i = 0; i < dt.Rows.Count; i++)
108             {
109                 arr.AddRange(dt.Rows[i].ItemArray);
110             }
111             /**
112             * The data array list is checked for null values and then buffered.
113             */
114             writeBuffer(new Object[] { (char)d[1] });
115             foreach (Object obj in arr)
116             {
117                 if (obj is DBNull)
118                 {
119                     writeBuffer(new Object[] { (char)d[0], Server.UrlEncode("") });
120                 }
121                 else
122                 {
123                     writeBuffer(new Object[] { (char)d[0], Server.UrlEncode(obj.       ↙
        ToString()) });
124                 }
125             }
126             /**
127             * The data array list is cleared for repopulation.
128             */
129             arr.Clear();
130             /**
131             * New data is inserted into the data array list.
132             */
133             for (int i = 0; i < dt.Columns.Count; i++)
134             {
135                 arr.Add(dt.Columns[i].ColumnName);
136             }
137             /**
138             * The data array list is buffered.
139             */
140             writeBuffer(new Object[] { (char)d[1] });
141             foreach (Object obj in arr)
142             {
143                 writeBuffer(new Object[] { (char)d[0], Server.UrlEncode(obj.ToString())↙
```

```
     });
144              }
145              /**
146              * Data table column and row count is added to the buffer.
147              */
148              writeBuffer(new Object[] { (char)d[1], dt.Columns.Count });
149              writeBuffer(new Object[] { (char)d[1], dt.Rows.Count });
150              /**
151              * The buffer is cached for later reuse, if requested.
152              */
153              if (Convert.ToBoolean(Request.Form["cache"]))
154              {
155                  Cache[Request.Form[3]] = Server.UrlEncode(_buffer);
156              }
157              /**
158              * The buffer data is returned to the client class as string representation.
159              */
160              Response.Write(Server.UrlEncode(_buffer));
161          }
162          private void deleteInsertUpdate(OleDbConnection c)
163          {
164              /**
165              * Function called on delete, insert and update command execution.
166              */
167              OleDbCommand cmd = new OleDbCommand(Request.Form["command"], c);
168              /**
169              * Buffer result data.
170              */
171              writeBuffer(new Object[] { (char)d[1], cmd.ExecuteNonQuery() });
172              /**
173              * Return buffer data to client class as string representation.
174              */
175              Response.Write(_buffer);
176          }
177          private void writeBuffer(Object[] obj)
178          {
179              /**
180              * Function for result buffering. Arguments are added to global buffer
     variable.
181              */
182              for (int i = 0; i < obj.Length; i++)
183              {
184                  _buffer += obj[i].ToString();
185              }
186          }
187          private void writeLog(Exception _e)
188          {
189              /**
190              * Function for error logging. Log file is created and/or opened.
191              */
192              StreamWriter sw;
193              sw = File.AppendText(Server.MapPath("~/log.txt"));
194              /**
195              * Error log entry is written and the log file is closed.
196              */
197              sw.WriteLine("ERROR Component: DataSourceConnector Thrown: " + DateTime.Now
      + " Type: " + _e.GetType());
198              sw.Close();
199          }
200      }
201  }
```

DetailsView.as

```
import mx.core.UIComponent;
import mx.controls.Label;
import mx.controls.TextInput;
import mx.controls.Button;
[Event("onError")]
[Event("onLoad")]
[Event("onReset")]
[Event("onUpdate")]
[IconFile("icon-24x13.png")]
[InspectableList("autoGenerateColumnNames", "autoGenerateControlButtons",
"columnHeight", "columnNames", "columnWidth", "dataProvider", "editable",
"visible")]
class com.remote.DetailsView extends UIComponent
{
      static var symbolName:String = "DetailsView";
      static var symbolOwner:Object = Object (DetailsView);
      var className:String = "DetailsView";
      private var mcBoundingBox:MovieClip;
      private var __autoGenerateColumnNames:Boolean;
      public function get autoGenerateColumnNames ():Boolean
      {
            return __autoGenerateColumnNames;
      }
      [Inspectable(defaultValue=true; type=Boolean)]
       public function set autoGenerateColumnNames (value:Boolean):Void
      {
            __autoGenerateColumnNames = value;
            invalidate ();
      }
      private var __autoGenerateControlButtons:Boolean;
      public function get autoGenerateControlButtons ():Boolean
      {
            return __autoGenerateControlButtons;
      }
      [Inspectable(defaultValue=true; type=Boolean)]
       public function set autoGenerateControlButtons (value:Boolean):Void
      {
            __autoGenerateControlButtons = value;
            invalidate ();
      }
      private var btnReset:MovieClip;
      private var btnUpdate:MovieClip;
      private var __columnHeight:Number;
```

```
	public function get columnHeight ():Number
	{
		return __columnHeight;
	}
	[Inspectable(defaultValue=22; type=Number)]
	 public function set columnHeight (value:Number):Void
	{
		__columnHeight = value;
		invalidate ();
	}
	private var __columnNames:Array;
	public function get columnNames ():Array
	{
		return __columnNames;
	}
	[Inspectable(type=Array)]
	 public function set columnNames (value:Array):Void
	{
		__columnNames = value;
		invalidate ();
	}
	private var __columnWidth:Number;
	public function get columnWidth ():Number
	{
		return __columnWidth;
	}
	[Inspectable(defaultValue=150; type=Number)]
	 public function set columnWidth (value:Number):Void
	{
		__columnWidth = value;
		invalidate ();
	}
	private var __dataProvider:Array;
	public function get dataProvider ():Array
	{
		return __dataProvider;
	}
	[Inspectable(type=Array)]
	 public function set dataProvider (value:Array):Void
	{
		__dataProvider = value;
		providerSet = false;
		providerInterval = setInterval (this, "onLoadProgress", 5);
```

```
	}
	private var __editable:Boolean;
	public function get editable ():Boolean
	{
		return __editable;
	}
	[Inspectable(defaultValue=true; type=Boolean)]
	 public function set editable (value:Boolean):Void
	{
		__editable = value;
	}
	private var editableState:Boolean = false;
	private var editColumn:Number;
	private var mcBoundaries:MovieClip;
	var onLoadProgress:Function = function ():Void
	{
		if (__dataProvider.length > 0)
		{
			clearInterval (providerInterval);
			invalidate ();
		}
	};
	private var providerInterval:Number;
	private var providerSet:Boolean = false;
	public function updateCommand (dataTable:String):String
	{
		var query:String;
		if (providerSet)
		{
			if (editableState)
			{
				abortEdit ();
			}
			/**
			* Data is extracted for query construction.
			*/
			var dataRow:Object;
			var name:Array = new Array ();
			var newValue:Array = new Array ();
			var oldValue:Array = new Array ();
			dataRow = __dataProvider[0];
			/**
			* Column names are extracted and inserted into the name
```

```
                  * array.
                  */
                  var i:Number = 0;
                  for (var columnName in dataRow)
                  {
                        name[i] = columnName;
                        i++;
                  }
                  /**
                  * Column data is extracted and inserted into the oldValue
                  * array.
                  */
                  for (i = 0; i < name.length; i++)
                  {
                        oldValue[i] = __dataProvider[0][name[i]];
                  }
                  /**
                  * New column data is extracted from component controls and
                  * inserted into the newValue array.
                  */
                  for (i = 0; i < name.length; i++)
                  {
                        newValue[i] = mcBoundaries["lblValue" + i].text;
                  }
                  /**
                  * Update query string is constructed.
                  */
                  query = "UPDATE " + dataTable + " SET";
                  for (i = 0; i < name.length - 1; i++)
                  {
                        query += " " + name[i] + "='" + newValue[i] + "',";
                  }
                  query += " " + name[name.length - 1] + "='" +
newValue[newValue.length - 1] + "' WHERE";
                  for (i = 0; i < name.length - 1; i++)
                  {
                        query += " " + name[i] + "='" + oldValue[i] + "'
AND";
                  }
                  query += " " + name[name.length - 1] + "='" +
oldValue[oldValue.length - 1] + "';";
            }
            else
```

```
            {
                  /**
                  * The component has not yet been populated with data or has
                  * not finished creating
                  * its child objects. Therefore required data for query
                  * construction cannot be extracted.
                  */
                  var event:Object = {target:this, type:"onError"};
                  dispatchEvent (event);
            }
            return query;
      }
      public function addColumnNameAt (columnIndex:Number,
columnName:String):Void
      {
            /**
            * If column label exist at specified index, no label will be
            * rendered.
            * An onError event will be triggered instead.
            */
            if (mcBoundaries["lblName" + columnIndex] == undefined)
            {
                  mcBoundaries.createObject ("Label", "lblName" +
columnIndex, mcBoundaries.getNextHighestDepth (), {_width:__columnWidth,
_height:__columnHeight, _x:0, _y:columnIndex * __columnHeight,
text:columnName});
            }
            else
            {
                  /**
                  * Column label exists at specified index. An onError event
                  * is dispatched.
                  */
                  var event:Object = {target:this, type:"onError"};
                  dispatchEvent (event);
            }
      }
      private function change (event:Object):Void
      {
      }
      public function changeColumnNameAt (columnIndex:Number,
columnName:String):Void
      {
            mcBoundaries["lblName" + columnIndex].text = columnName;
```

```
	}
	public function clearAll ():Void
	{
	}
	private function click (event:Object):Void
	{
		event.target.event (event.target.component);
	}
	private function columnEdit (columnIndex:Number):Void
	{
		/**
		* All non selected column rows are reset to default
		* highlighting.
		*/
		if (editableState)
		{
			abortEdit ();
		}
		mcBoundaries["mcBack" + columnIndex]._alpha = 100;
		/**
		* Continues with column edit and enters an editable state.
		*/
		editableState = true;
		/**
		* Clicked label is referenced. This label will later on be
		* replaced with an input field.
		*/
		var columnToEdit:MovieClip;
		columnToEdit = mcBoundaries["lblValue" + columnIndex];
		/**
		* Column index is stored within the scope of the class
		* for later reference.
		*/
		editColumn = columnIndex;
		/**
		* An input field is created and positioned using coordinates of
		* clicked label.
		*/
		var editableField:MovieClip;
		editableField = mcBoundaries.createObject ("TextInput",
"txtValue" + columnIndex, mcBoundaries.getNextHighestDepth (),
{_width:__columnWidth, _height:__columnHeight, _x:columnToEdit._x,
_y:columnToEdit._y, text:columnToEdit.text, html:false, _c:columnIndex});
```

```
            editableField.addEventListener ("change", this);
            /**
            * Referenced label is destroyed.
            */
            mcBoundaries.destroyObject (columnToEdit._name);
      }
      public function columnEditableAt (columnIndex:Number):Void
      {
            if (providerSet)
            {
                  columnEdit (columnIndex);
            }
            else
            {
                  /**
                  * Data have not yet been loaded and can therefore  not
                  * be edited.
                  */
                  var event:Object = {target:this, type:"onError"};
                  dispatchEvent (event);
            }
      }
      private function destroyChildren ():Void
      {
            /**
            * The mcBoundaries movie clip is destroyed and with it all its
            * rendered children.
            * This method is run when the component is to be updated or
            * cleared.
            */
            destroyObject (mcBoundaries._name);
      }
      private function drawHitArea (obj:Object, __width:Number,
__height:Number):Void
      {
            /**
            * Method draws a solid rectangular hit area for each column row.
            * The coloration of this area will indicate row selection during
            * run-time.
            */
            with (obj)
            {
                  beginFill (0xE3FFD6, 100);
```

```
                moveTo (0, 0);
                lineTo (__width, 0);
                lineTo (__width, __height);
                lineTo (0, __height);
                lineTo (0, 0);
                endFill ();
          }
    }
    private function highlightColumn (columnIndex:Number,
highlight:Boolean):Void
    {
          /**
          * Method for column row highlighting. A row will be highlighted
          * on hover. If a
          * row is selected the highlighting will be locked until it's
          * unselected.
          * This method is somewhat scattered. Partial code that handles
          * highlighting can
          * be found in the columnEdit method.
          */
          if (highlight)
          {
                mcBoundaries["mcBack" + columnIndex]._alpha = 100;
          }
          else
          {
                if (editColumn != columnIndex)
                {
                      mcBoundaries["mcBack" + columnIndex]._alpha = 0;
                }
          }
    }
    public function removeColumnNameAt (columnIndex:Number):Void
    {
          /**
          * Column name label at specified index is destroyed.
          */
          destroyObject (mcBoundaries["lblName" + columnIndex]._name);
    }
    private function reset (component:DetailsView):Void
    {
          /**
          * Child objects are reset using set dataProvider.
```

```
            */
            component.invalidate ();
            /**
            * An onReset event is dispatched as result.
            */
            var event:Object = {target:component, type:"onReset"};
            component.dispatchEvent (event);
      }
      private function setHitArea (obj:Object):Void
      {
            /**
            * Method sets the column row events onRollOver, onRollOut and
            * onPress.
            * These events are set for the row hit areas and editable text
            * fields rendered on selection and calls a highlight method.
            */
            var component:DetailsView = this;
            obj.onRollOver = function ()
            {
                  component.highlightColumn (this._c, true);
            };
            obj.onRollOut = function ():Void
            {
                  component.highlightColumn (this._c, false);
            };
            if (__editable)
            {
                  obj.onPress = function ():Void
                  {
                        component.columnEdit (this._c);
                  };
            }
      }
      private function abortEdit ():Void
      {
            var i:Number = 0;
            for (var column in __dataProvider[0])
            {
                  mcBoundaries["mcBack" + i]._alpha = 0;
                  i++;
            }
            /**
            * A column value is currently being edited. Edited column will be
```

```
            * reset to label. Input field to reset is referenced.
            */
            var columnToReset:MovieClip;
            columnToReset = mcBoundaries["txtValue" + editColumn];
            /**
            * A label is created and positioned above referenced input field.
            */
            var columnLabel:MovieClip;
            columnLabel = mcBoundaries.createObject ("Label", "lblValue" +
editColumn, mcBoundaries.getNextHighestDepth (), {_width:__columnWidth,
_height:__columnHeight, _x:columnToReset._x, _y:columnToReset._y,
text:columnToReset.text, html:false, _c:editColumn});
            setHitArea (columnLabel);
            /**
            * Referenced input field is destroyed.
            */
            mcBoundaries.destroyObject (columnToReset._name);
            editableState = false;
            editColumn = null;
      }
      private function update (component:DetailsView):Void
      {
            /**
            * An update command is constructed using current component
            * values. This command can be used by the DataSourceConnector
            * to update an external data source.
            *
            * When the onUpdate event is dispatched, the updateCommand
            * method can be accessed.
            */
            var event:Object = {target:component, type:"onUpdate"};
            component.dispatchEvent (event);
      }
      function DetailsView ()
      {
            /**
            * A mouse listener is set. If a column value is being edited and
            * the
            * mouse is clicked outside of the component body, editing will be
            * aborted.
            */
            var component:DetailsView = this;
            var mouseListener:Object = new Object ();
```

```
            mouseListener.onMouseDown = function ()
            {
                  if (component.editableState)
                  {
                        var i:Number = 0;
                        var fieldToEdit:MovieClip;
                        for (var column in component.__dataProvider[0])
                        {
                              if (component.mcBoundaries["txtValue" + i] !=
undefined)
                              {
                                    fieldToEdit =
component.mcBoundaries["txtValue" + i];
                                    if (component._xmouse < fieldToEdit._x -
component.__columnWidth - 5 || component._xmouse > fieldToEdit._x +
fieldToEdit._width)
                                    {
                                          component.abortEdit ();
                                    }
                                    else if (component._ymouse <
fieldToEdit._y || component._ymouse > fieldToEdit._y + fieldToEdit._height)
                                    {
                                          component.abortEdit ();
                                    }
                              }
                              i++;
                        }
                  }
            };
            Mouse.addListener (mouseListener);
      }
      function init ():Void
      {
            super.init ();
            mcBoundingBox._visible = false;
            mcBoundingBox._width = 0;
            mcBoundingBox._height = 0;
      }
      public function createChildren ():Void
      {
            if (__autoGenerateControlButtons)
            {
```

```
                  btnUpdate = createObject ("Button", "btnUpdate",
this.getNextHighestDepth (), {_width:115, _height:22, _x:0, _y:0, label:"Save
changes", event:update, component:this});
                  btnReset = createObject ("Button", "btnReset",
this.getNextHighestDepth (), {_width:55, _height:22, _x:120, _y:0,
label:"Reset", event:reset, component:this});
                  btnUpdate.addEventListener ("click", this);
                  btnReset.addEventListener ("click", this);
            }
            mcBoundaries = this.createEmptyMovieClip ("mcBoundaries",
this.getNextHighestDepth ());
            size ();
      }
      function draw ():Void
      {
            if (__dataProvider.length > 0)
            {
                  var event:Object = {target:this, type:"onLoad"};
                  dispatchEvent (event);
                  /**
                  * If editing whilst updating, reset; editing is aborted and
                  * component child objects
                  * are returned to their initial state.
                  */
                  if (editableState)
                  {
                        abortEdit ();
                  }
                  providerSet = true;
                  /**
                  * The component is cleared before it's updated using
                  * new data.
                  */
                  destroyChildren ();
                  /**
                  * Set dataProvider is limited to its first row of data.
                  * This is
                  * the very essence of the DetailsView component. The data
                  * is
                  * divided into two arrays; one containing column names and
                  * one containing column data.
                  */
                  var dataRow:Object;
                  var name:Array = new Array ();
```

```
                  var value:Array = new Array ();
                  dataRow = __dataProvider[0];
                  /**
                  * Column names are extracted and inserted into the name
                  * array.
                  */
                  var i:Number = 0;
                  for (var columnName in dataRow)
                  {
                        name[i] = columnName;
                        i++;
                  }
                  /**
                  * Column data is extracted and inserted into the value
                  * array.
                  */
                  for (i = 0; i < name.length; i++)
                  {
                        value[i] = __dataProvider[0][name[i]];
                  }
                  /**
                  * Child objects are created using extracted column names
                  * and corresponding data.
                  */
                  mcBoundaries = this.createEmptyMovieClip ("mcBoundaries",
this.getNextHighestDepth ());
                  var mcColumnBackground:MovieClip;
                  var lblColumnName:MovieClip;
                  var lblColumnValue:MovieClip;
                  for (i = 0; i < name.length; i++)
                  {
                        mcColumnBackground =
mcBoundaries.createEmptyMovieClip ("mcBack" + i,
mcBoundaries.getNextHighestDepth ());
                        /**
                        * Rectangular background is rendered.
                        */
                        drawHitArea (mcColumnBackground, width,
__columnHeight);
                        /**
                        * Movie clip properties are set.
                        */
                        mcColumnBackground._y = i * __columnHeight;
```

```
                        mcColumnBackground._c = i;
                        mcColumnBackground._alpha = 0;
                        mcColumnBackground.useHandCursor = false;
                        setHitArea (mcColumnBackground);
                        /**
                        * Column row labels are created. Labeling is variable
                        * dependent. If autoGenerateColumnNames
                        * is set to false the columnNames array will be used
                        * for proper labeling. If no values
                        * have been specified, labels will be marked as
                        * undefined.
                        */
                        lblColumnName = mcBoundaries.createObject ("Label",
"lblName" + i, mcBoundaries.getNextHighestDepth (), {_width:__columnWidth,
_height:__columnHeight, _y:i * __columnHeight});
                        if (__autoGenerateColumnNames)
                        {
                              lblColumnName.text = name[i];
                        }
                        else
                        {
                              lblColumnName.text = __columnNames[i];
                        }
                        /**
                        * Column values are presented as labels. Events are
                        * then set.
                        */
                        lblColumnValue = mcBoundaries.createObject ("Label",
"lblValue" + i, mcBoundaries.getNextHighestDepth (), {_width:__columnWidth,
_height:__columnHeight, _x:__columnWidth + 5, _y:i * __columnHeight,
text:value[i], html:false, _c:i});
                        setHitArea (lblColumnValue);
                  }
                  /**
                  * This section is run if control buttons have been
                  * rendered.
                  */
                  if (__autoGenerateControlButtons)
                  {
                        btnReset._y = i * __columnHeight;
                        btnUpdate._y = i * __columnHeight;
                  }
            }
            super.draw ();
```

DetailsView.as

```
	}
	function size ():Void
	{
		super.size ();
		invalidate ();
	}
}
```

SignInWindow.as

```
import mx.core.UIComponent;
import mx.containers.Window;
import mx.managers.PopUpManager;
import mx.controls.Label;
import mx.controls.TextInput;
import mx.controls.Button;
import mx.controls.CheckBox;
[IconFile("icon-24x13.png")]
[Event("onError")]
[Event("onLoad")]
[Event("onSigned")]
[InspectableList("dataProvider", "dataSource", "dataTable", "requestPath",
"retrievalPath", "tableColumns", "visible")]
class com.remote.SignInWindow extends UIComponent
{
      static var symbolName:String = "SignInWindow";
      static var symbolOwner:Object = Object (SignInWindow);
      var className:String = "SignInWindow";
      private var mcBoundingBox:MovieClip;
      private var winSgn:MovieClip;
      private var lblUsr:MovieClip;
      private var lblPwd:MovieClip;
      private var txtUsr:MovieClip;
      private var txtPwd:MovieClip;
      private var chkRetr:MovieClip;
      private var btnSgn:MovieClip;
      private var btnCnl:MovieClip;
      private var __dataProvider:String;
      public function get dataProvider ():String
      {
            return __dataProvider;
      }
      [Inspectable(defaultValue="Microsoft.Jet.OLEDB.4.0", type=String)]
       public function set dataProvider (value:String):Void
      {
            __dataProvider = value;
      }
      private var __dataSource:String;
      public function get dataSource ():String
      {
            return __dataSource;
      }
      [Inspectable(type=String)]
```

```
		 public function set dataSource (value:String):Void
		{
			__dataSource = value;
		}
		private var __dataTable:String;
		public function get dataTable ():String
		{
			return __dataTable;
		}
		[Inspectable(type=String)]
		 public function set dataTable (value:String):Void
		{
			__dataTable = value;
		}
		private var __password:String;
		public function get password ():String
		{
			return __password;
		}
		public function set password (value:String):Void
		{
			__password = value;
			winSgn.txtPwd.text = __password;
		}
		private var __requestMethod:String = "POST";
		public function get requestMethod ():String
		{
			return __requestMethod;
		}
		private var __requestPath:String;
		public function get requestPath ():String
		{
			return __requestPath;
		}
		[Inspectable(type=String)]
		 public function set requestPath (value:String):Void
		{
			__requestPath = value;
		}
		private var __retrievalPath:String;
		public function get retrievalPath ():String
		{
			return __retrievalPath;
```

```
	}
	[Inspectable(type=String)]
	 public function set retrievalPath (value:String):Void
	{
		__retrievalPath = value;
	}
	private var __retrieve:Boolean = false;
	public function get retrieve ():Boolean
	{
		return __retrieve;
	}
	public function set retrieve (value:Boolean):Void
	{
		__retrieve = value;
	}
	private var __signed:Boolean = false;
	public function get signed ():Boolean
	{
		return __signed;
	}
	private var __tableColumns:Array;
	public function get tableColumns ():Array
	{
		return __tableColumns;
	}
	[Inspectable(type=Array)]
	 public function set tableColumns (value:Array):Void
	{
		__tableColumns = value;
	}
	private var __username:String;
	public function get username ():String
	{
		return __username;
	}
	public function set username (value:String):Void
	{
		__username = value;
		winSgn.txtUsr.text = __username;
	}
	private var __visible:Boolean = true;
	public function get visible ():Boolean
	{
```

```
            return __visible;
      }
      public function set visible (value:Boolean):Void
      {
            __visible = value;
            winSgn._visible = __visible;
      }
      public function sign ():Void
      {
            Sign (this);
      }
      public function unsign ():Void
      {
            __signed = false;
      }
      private function click (event:Object):Void
      {
            /**
            * Call to event specific function using component reference as an
            * argument.
            * The component reference is used to access private variables set
            * at runtime.
            */
            event.target.event (event.target.component);
      }
      private function check (component:SignInWindow):Void
      {
            /**
            * Event function called on checkbox check and unchecks. Sets
            * state for
            * password text field and changes sign in button label.
            */
            component.txtPwd.editable = !component.chkRetr.selected;
            component.__retrieve = component.chkRetr.selected;
            if (component.chkRetr.selected)
            {
                  component.btnSgn.label = "Retrieve";
            }
            else
            {
                  component.btnSgn.label = "Sign In";
            }
      }
```

```
	private function Sign (component:SignInWindow):Void
	{
		/**
		* Event function called on sign in. Sends required variables to
		* the
		* server resident class and processes returned data.
		*/
		var send:LoadVars = new LoadVars ();
		var load:LoadVars = new LoadVars ();
		component.__username = component.txtUsr.text;
		component.__password = component.txtPwd.text;
		try
		{
			/**
			* Variables to be passed to the server resident class are
			* set.
			*/
			send.retrievalPath = component.__retrievalPath;
			send.retrieve = component.__retrieve;
			send.password = component.__password;
			send.username = component.__username;
			send.tableColumns = component.__tableColumns;
			send.dataTable = component.__dataTable;
			send.dataProvider = component.__dataProvider;
			send.dataSource = component.__dataSource;
			/**
			* Validates variable data. If invalid an error is thrown.
			*/
			var data:Array = unescape (send.toString ()).split
(String.fromCharCode (38));
			for (var i:Number = 0; i < data.length - 1; i++)
			{
				data[i] = data[i].substr (data[i].indexOf
(String.fromCharCode (61)) + 1);
				if (data[i].length == 0)
				{
					throw new Error ();
				}
			}
			/**
			* If retrieve is set to true, a retrievalPath must be
			* specified. Otherwise an error will occur.
			*/
```

```
                  if (component.__retrieve &&
component.__retrievalPath.length == 0)
                  {
                        trace ("ERROR");
                        throw new Error ();
                  }
                  /**
                  * Component core method is called. Waiting for server
                  * response.
                  */
                  send.sendAndLoad (component.__requestPath +
"SignInWindow.aspx", load, component.__requestMethod);
                  load.onLoad = function (success:Boolean):Void
                  {
                        if (success)
                        {
                              /**
                              * Server data was returned. Proceeds with data
                              * process.
                              */
                              component.__signed = Boolean (Number
(this.signed));
                              if (component.__signed)
                              {
                                    /**
                                    * Sign in was successful. A sign event is
                                    * dispatched and
                                    * the component window is closed.
                                    */
                                    event = {target:component,
type:"onSigned"};
                                    component.dispatchEvent (event);
                                    component.visible = false;
                              }
                              /**
                              * Server data has been loaded and processed. A
                              * load event is dispatched.
                              */
                              var event:Object = {target:component,
type:"onLoad"};
                              component.dispatchEvent (event);
                        }
                        else
                        {
```

```
                              /**
                              * No server data was returned. An error event
                              * is dispatched.
                              */
                              throw new Error ();
                        }
                  };
            }
            catch (Error)
            {
                  /**
                  * An error occurred. An error event is dispatched.
                  */
                  var event:Object = {target:component, type:"onError"};
                  component.dispatchEvent (event);
            }
      }
      private function close (component:SignInWindow):Void
      {
            /**
            * An event function. The component window is closed.
            */
            component.visible = false;
      }
      function SignInWindow ()
      {
      }
      function init ():Void
      {
            super.init ();
            mcBoundingBox._visible = false;
            mcBoundingBox._width = 0;
            mcBoundingBox._height = 0;
      }
      public function createChildren ():Void
      {
            /**
            * Creates component user controls.
            */
            winSgn = PopUpManager.createPopUp (this, Window, false,
{_width:335, _height:165, _x:_x, _y:_y, title:"Sign In", _visible:__visible,
closeButton:true, event:close, component:this});
```

```
            lblUsr = winSgn.createObject ("Label", "lblUsr",
winSgn.getNextHighestDepth (), {_width:150, _height:22, _x:12, _y:42,
text:"Username"});
            lblPwd = winSgn.createObject ("Label", "lblPwd",
winSgn.getNextHighestDepth (), {_width:150, _height:22, _x:167, _y:42,
text:"Password"});
            txtUsr = winSgn.createObject ("TextInput", "txtUsr",
winSgn.getNextHighestDepth (), {_width:150, _height:22, _x:12, _y:64,
html:false});
            txtPwd = winSgn.createObject ("TextInput", "txtPwd",
winSgn.getNextHighestDepth (), {_width:150, _height:22, _x:167, _y:64,
html:false, password:true});
            chkRetr = winSgn.createObject ("CheckBox", "chkRetr",
winSgn.getNextHighestDepth (), {_width:150, _height:22, _x:12, _y:86,
label:"Retrieve password", event:check, component:this});
            btnSgn = winSgn.createObject ("Button", "btnSgn",
winSgn.getNextHighestDepth (), {_width:75, _height:22, _x:12, _y:130,
label:"Sign In", event:Sign, component:this});
            btnCnl = winSgn.createObject ("Button", "btnCnl",
winSgn.getNextHighestDepth (), {_width:55, _height:22, _x:92, _y:130,
label:"Cancel", event:close, component:this});
            /**
            * Initiates component user controls events.
            */
            winSgn.addEventListener ("click", this);
            chkRetr.addEventListener ("click", this);
            btnSgn.addEventListener ("click", this);
            btnCnl.addEventListener ("click", this);
            size ();
      }
      function draw ():Void
      {
            super.draw ();
      }
      function size ():Void
      {
            super.size ();
            invalidate ();
      }
}
```

```
1  namespace Com.Remote
2  {
3      using System;
4      using System.IO;
5      using System.Data;
6      using System.Data.OleDb;
7      using System.Configuration;
8      using System.Collections;
9      using System.Web;
10     using System.Web.Security;
11     using System.Web.UI;
12     using System.Web.UI.WebControls;
13     using System.Web.UI.WebControls.WebParts;
14     using System.Web.UI.HtmlControls;
15
16     public partial class SignInWindow : System.Web.UI.Page
17     {
18         protected void Page_Load(object sender, EventArgs e)
19         {
20             if (!Convert.ToBoolean(Request.Form["retrieve"]))
21             {
22                 /**
23                 * A connection is created and opened.
24                 */
25                 OleDbConnectionStringBuilder c_str = new OleDbConnectionStringBuilder();
26                 c_str.Provider = Request.Form["dataProvider"];
27                 c_str.DataSource = Server.MapPath(Request.Form["dataSource"]);
28
29                 OleDbConnection c = new OleDbConnection(c_str.ConnectionString);
30
31                 try
32                 {
33                     c.Open();
34
35                     string[] tableColumns = Request.Form["tableColumns"].Split((char)44)↙
   ;
36                     /**
37                     * A select query is composed. A select command is then created and  ↙
   run.
38                     * Command output is returned to client class as string             ↙
   representation.
39                     */
40                     string query = "SELECT COUNT(*) FROM " + Request.Form["dataTable"] +↙
    " WHERE " + tableColumns[0] + " = '" + Request.Form["username"] + "' AND " +       ↙
   tableColumns[1] + " = '" + Request.Form["password"] + "'";
41
42                     OleDbCommand cmd = new OleDbCommand(query, c);
43
44                     Response.Write((char)38 + "signed=" + cmd.ExecuteScalar());
45                 }
46                 catch (Exception _e)
47                 {
48                     /**
49                     * An exception was thrown. The exception is logged.
50                     */
51                     writeLog(_e);
52                 }
53                 finally
54                 {
55                     /**
56                     * A connection is closed.
57                     */
58                     c.Close();
59                 }
60             }
61             else
62             {
63                 /*
64                 * Retrieve account password. A function has to be declared.
65                 */
66                 Response.Redirect(Request.Form["retrievalPath"]);
67             }
68         }
69         private void writeLog(Exception _e)
```

```
70          {
71              /**
72              * Function for error logging. Log file is created and/or opened.
73              */
74              StreamWriter sw;
75              sw = File.AppendText(Server.MapPath("~/log.txt"));
76              /**
77              * Error log entry is written and the log file is closed.
78              */
79              sw.WriteLine("ERROR Component: SignInWindow Thrown: " + DateTime.Now + "
    Type: " + _e.GetType());
80              sw.Close();
81          }
82      }
83  }
```